

SYBASE®

Programmer's Reference for COBOL

Mainframe Connect™ Server Option

15.0

IBM CICS, IMS, and MVS

DOCUMENT ID: DC36520-01-1500-01

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](http://www.sybase.com/detail?id=1011207) 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	vii	
CHAPTER 1	Introduction	1
	What is Open ServerConnect?	1
	What is Gateway-Library?	2
	Gateway-Library functions	2
	Using Client/Server connections	3
	SNA connections	4
	TCP/IP connections	4
	Initializing the Gateway-Library environment	4
	Starting and ending a conversation	5
	Handling client requests	5
	Processing client requests	6
	Differences between CICS, IMS TM, and MVS	7
	General processing procedures	8
	Processing an RPC	9
	Processing a SQL language request	10
	Processing a cursor request	11
	Processing a dynamic SQL request	12
	Processing a long-running transaction	13
	Additional processing options	14
	Tracing and accounting functions	15
CHAPTER 2	Topics	17
	Character sets	17
	Supported workstation character sets	18
	Supported mainframe character sets	19
	Communication states	19
	Cursors	20
	What is a cursor?	21
	Benefits of using cursors	21
	How cursors work in Open ServerConnect	21
	Types of cursor commands	22

CURSOR-DESC structure.....	26
Customization	35
Datatypes	36
Datatype descriptions and correspondences	36
Character datatypes	41
Binary and decimal datatypes	42
Graphic datatypes	45
Unsupported datatypes	45
Dynamic SQL support	45
Events	53
The login packet.....	53
Long-running transactions.....	54
Calls in a long-running transaction	55
Mixed-mode applications	56
Rules for writing mixed-mode applications	56
Native languages	57
Processing Japanese client requests.....	58
The Japanese Conversion Module.....	58
Japanese character sets	59
Datatypes used with Japanese characters.....	60
Summary of datatypes used with Japanese characters.....	62

CHAPTER 3

Functions.....	67
List of functions	67
General information about functions	69
TDACCEPT	70
TDCONVRT	77
TDCURPRO.....	83
TDESCRIB	87
TDFREE	95
TDGETREQ	98
TDGETSOI.....	104
TDGETUSR	108
TDINFACT	112
TDINFBCD	116
TDINFLOG	121
TDINFPGM	125
TDINFPRM.....	129
TDINFRPC	134
TDINFSPT.....	136
TDINFUDT	140
TDINIT.....	142
TDLOCPRM.....	146
TDLSTSPT.....	149

	TDNUMPRM	152
	TDRCVPRM	154
	TDRCVSQL	162
	TDRESULT	167
	TDSETACT	170
	TDSETBCD	174
	TDSETLEN	180
	TDSETLOG	183
	TDSETPRM	188
	TDSETPT	193
	TDSETSOI	196
	TDSETSPT	200
	TDSETUDT	204
	TDSNDDON	206
	TDSNDMSG	213
	TDSNDROW	220
	TDSQLLEN	224
	TDSTATUS	227
	TDTERM	232
	TDYNAMIC	234
	TDWRTLOG	238
APPENDIX A	Gateway-Library Quick Reference	243
APPENDIX B	Sample RPC Application for CICS.....	253
	Sample program SYCCSAR2	254
	Sample program SYCCSAU2	268
	Sample program SYCCSAW2	277
	Sample program SYCCSAY2	287
	Sample program SYCCSAZ2.....	298
APPENDIX C	Sample Language Application for CICS	307
	Sample program SYCCSAL2.....	308
APPENDIX D	Sample RPC Application for IMS TM (Implicit).....	319
	Sample program SYICSAD2.....	319
APPENDIX E	Sample RPC Application for IMS TM (Explicit)	333
	Sample program SYIXSAM2.....	333
APPENDIX F	Sample Mixed-Mode Application	347

	Sample program SYCTSAX5.....	347
APPENDIX G	Sample Tracing and Accounting Program	385
	Sample program SYCCSAS2	386
Index		407

About This Book

This guide references information for the COBOL version of Open ServerConnect™ Gateway-Library.

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

Audience

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

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

How to use this book

This guide contains these chapters;

- Chapter 1, “Introduction,” gives 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,” has a sample COBOL application program that processes client RPC requests under CICS, as well as three COBOL programs that are Open ServerConnect versions of the RSP3C, RSP4C and RSP8C remote stored procedures
- Appendix C, “Sample Language Application for CICS,” contains a sample COBOL application program that processes client language requests under CICS.
- Appendix D, “Sample RPC Application for IMS TM (Implicit),” provides a sample COBOL application program that processes client RPC requests under the IMS TM implicit API.
- Appendix E, “Sample RPC Application for IMS TM (Explicit),” has a sample COBOL application program that processes client RPC requests under the IMS TM explicit API.
- Appendix F, “Sample Mixed-Mode Application,” shows a sample COBOL 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 COBOL 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 this release of the Mainframe Connect™ Integrated Product Set.

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

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

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

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

Related documents

The documentation set consists of

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

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

- *Mainframe Connect 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/I* – describes writing Client Option programs that call PL/I Client-Library functions. This guide contains reference pages for Client-Library routines and descriptions of the underlying concepts for PL/I programmers.
 - Mainframe Connect Server Option *Programmer's Reference for PL/I* – provides reference material for writing Server Option programs that call PL/I Gateway-Library functions. This guide contains reference pages for Gateway-Library routines and descriptions of the underlying concepts for PL/I programmers.
 - Mainframe Connect Client Option *Programmer's Reference for 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 *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	<i>/usr/bin directory, interfaces file</i>
Variables	<i>n bytes</i>
Adaptive Server datatypes	datetime, float

This type of information	Looks like this
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	<i>BUFFER, RETCODE</i>
Client-Library function arguments that are input (I) or output (O)	<i>COMMAND</i> – (I) <i>RETCODE</i> – (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
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

This chapter includes the following topics:

- What is Open ServerConnect?
- What is Gateway-Library?
- Gateway-Library functions
- Using Client/Server connections
- General processing procedures

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/I databases, as well as 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.

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 “Differences between CICS, IMS TM, and MVS” on page 7.

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

Open ServerConnect uses the Sybase Tabular Data Stream™ (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 to 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.

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 explanation of all Gateway-Library functions, see 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

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.

Note 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 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 structure
- TDPROC structure

IHANDLE structure

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.

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 structure

The *TDPROC* structure corresponds to the *DBPROCESS* structure in DB-Library and 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 *TDPROC* 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 TDRCVxxx function you code in your server application is responding to a corresponding send function issued by the client or TRS, and that the data you send with a TDSNDxxx 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 (SYCCSAL2) 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 TM, and MVS

For the most part, the use of Gateway-Library functions in CICS, IMS TM, and MVS 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	Difference between CICS, IMS TM, and MVS
TDINFRPC TDSTATUS	<p>The action taken when the communication state (<i>COMM-STATE</i>) is TDS-RESET can differ between CICS, MVS and the IMS TM implicit API:</p> <ul style="list-style-type: none"> • 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	<p>The first argument differs between CICS and IMS TM:</p> <ul style="list-style-type: none"> • 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 (<i>IO-PCB</i>). • Under MVS, a null pointer should be used.
TDSETPT	<p>Used with IMS TM only, to indicate the type of IMS TM transaction.</p>
TDSNDDON	<p>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 54.</p>
TDINFACT TDSETACT	<p>Accounting records are written to different logs under CICS, IMS TM, and MVS:</p> <ul style="list-style-type: none"> • Under CICS, accounting functions use VSAM files as log files. The default file name is <i>SYTACCTI</i>. • Under IMS TM, accounting functions use the IMS TM log. • Under MVS, the records are written to a sequential file. The <i>DDNAME</i> of this file is specified as a parameter in TDCUSTOM.
TDINFLOG TDINFSPT TDLSTSPT TDSETLOG TDSETSPT TDWRTLOG	<p>Trace records are written to different logs under CICS, IMS TM, and MVS:</p> <ul style="list-style-type: none"> • Under CICS, tracing functions use VSAM files as log files. The default file name is <i>SYTDLOGI</i>. • Under IMS TM, tracing functions use the IMS TM log. • Under MVS, the records are written to a sequential file. The <i>DDNAME</i> of this file is specified as a parameter in TDCUSTOM.

General processing procedures

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

- 1 Prepares the environment.

- 2 Accepts the request and retrieves the language, cursor, or dynamic request or RPC parameters.
- 3 Performs the requested action.
- 4 Returns results to the requesting client.
- 5 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 contained in the appendices of this book 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 in Table 1-2.

Table 1-2: Functions to process RPCs

Task	Function
<i>1. Prepare for incoming requests.</i>	
Initialize the Gateway-Library environment.	TDINIT
Specify the type of IMS TM transaction (used with IMS TM transactions only).	TDSETPT
<i>2. Accept the incoming request.</i>	
Accept the incoming request.	TDACCEPT
<i>3. Handle incoming parameters.</i>	
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
<i>4. Process the request.</i>	
Perform the requested task(s).	

Task	Function
<i>5. Prepare to return results to the client.</i>	
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
<i>6. Return data to the client.</i>	
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.	
<i>7. End the conversation.</i>	
Free the <i>TDPROC</i> 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 in Table 1-3.

Table 1-3: Functions to process language requests

Task	Function
<i>1. Prepare for incoming requests.</i>	
Initialize the Gateway-Library environment.	TDINIT
Specify the type of IMS TM transaction (used with IMS TM transactions only).	TDSETPPT
<i>2. Accept the incoming request.</i>	
Accept the incoming request.	TDACCEPT
<i>3. Handle incoming SQL statements.</i>	
Determine the length (in bytes) of the incoming SQL string.	TDSQLLEN
Retrieve the SQL string (Loop until all parameters are retrieved).	TDRCVSQL
<i>4. Process the request.</i>	
Retrieve the requested data from the database.	
<i>5. Prepare to return results to the client.</i>	
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

Task	Function
<i>6. Return data to the client.</i>	
Send data to the client, one row at a time (Loop until all rows are sent). Send the return parameters, tell the client when results are finished, and close the connection.	TDSNDROW
<i>7. End the conversation.</i>	
Free the <i>TDPROC</i> structure. Free the MVS storage (required with IMS TM; optional but recommended with CICS).	TDFREE TDTERM

Processing a cursor request

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

Table 1-4: Functions to process cursor requests

Task	Function
<i>1. Prepare for incoming requests.</i>	
Initialize the Gateway-Library environment. Specify the type of IMS TM transaction (used with IMS TM transactions only).	TDINIT TDSETPT
<i>2. Accept the incoming request.</i>	
Accept the incoming request.	TDACCEPT
<i>3. Determine the type of request.</i>	
Determine the type of client request: RPC, language, dynamic, or cursor.	TDINFPGM
<i>4. Determine the type of cursor request.</i>	
Retrieve <i>CURSOR-COMMAND</i> , <i>CURSOR-ID</i> , and <i>COMMAND-OPTIONS</i> .	TDCURPRO
<i>5. Process the cursor request.</i>	
Get the SQL statement, number of parameters, table name, and parameters format. Either receive input parameters or update columns.	TDRCVSQL TDNUMPRM TDINFPRM TDRCVPRM
<i>6. Describe the rows.</i>	
Describe the rows. Send rows.	TDESCRIB TDSNDROW
<i>7. Send return information.</i>	
Send acknowledgment, <i>CURSOR-STATUS</i> , <i>CURSOR-INFO</i> .	TDCURPRO

Task	Function
8. <i>Send DONE.</i>	
Send a DONE package.	TDSNDDON
9. <i>Accept the next request.</i>	
Accept the incoming request.	TDGETREQ
10. <i>End the conversation.</i>	
Send final DONE package.	TDSNDDON
Free the <i>TDPROC</i> structure.	TDFREE
Free the MVS storage.	TDTERM

Processing a dynamic SQL request

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

Table 1-5: Functions to process dynamic requests

Task	Function
1. <i>Prepare for incoming requests.</i>	
Initialize the Gateway-Library environment.	TDINIT
Specify the type of IMS TM transaction (used with IMS TM transactions only).	TDSETPT
2. <i>Accept the incoming request.</i>	
Accept the incoming request.	TDACCEPT
3. <i>Determine the event type.</i>	
Determine the type of client request: RPC, language, dynamic, or cursor.	TDINFRPC
4. <i>Determine the type of dynamic operation.</i>	
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. <i>Process dynamic operation.</i>	
Get the statement (for prepare).	TDYNAMIC
Get the statement ID (for all dynamic requests).	

Task	Function
<i>6. Handle the incoming parameters.</i>	
Determine number of parameters (for execute).	TDNUMPRM
Retrieve number of parameters (for execute).	TDINFPRM
Retrieve input parameters (for execute).	TDRCVPRM
<i>7. Describe the data.</i>	
Describe input/output parameters, or rows to be returned.	TDSNDROW
<i>8. Return data to the client.</i>	
Send result rows for execute or execute immediate.	TDYNAMIC
Send an acknowledge request.	TDSNDDON
Send a done package.	
<i>9. Get the next request type.</i>	
Accept the incoming request.	TDGETREQ
<i>10. End the conversation.</i>	
Send final DONE package.	TDSNDDON
Free the <i>TDPROC</i> structure.	TDFREE
Free the MVS storage.	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 in Table 1-6. The arrows in the table indicate code loops.

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

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

Task	Function
<i>3. Handle incoming parameters.</i>	
Determine how many parameters were sent. → Loop until all parameters are retrieved.	TDNUMPRM TDINFPRM TDRCVPRM
<i>4. Process the request.</i>	
Perform the requested task(s).	
<i>5. Prepare to return results to the client.</i>	
→ Loop to describe return parameters and columns in return rows.	TDSETPRM TDESCRIB
<i>6. Return data to the client</i>	
→ 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
<i>7. Accept next request.</i>	
Accept the incoming request.	TDGETREQ
<i>8. Repeat steps 3-6.</i>	
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).	
<i>9. End the conversation.</i>	
Free the <i>TDPROC</i> structure. Free the MVS storage (required with IMS TM; optional but recommended with CICS).	TDFREE TDTERM

Additional processing options

Table 1-7 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

Task	Function
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.	TDSEACT
Determine whether accounting is on and the name of the accounting log file.	TDINFACT

Topics

This chapter contains information about Open ServerConnect concepts and procedures that are grouped by the following listed topics:

- Character sets
- Communication states
- Cursors
- Customization
- Datatypes
- Dynamic SQL support
- Events
- The login packet
- Long-running transactions
- Mixed-mode applications
- Native languages
- Processing Japanese client requests

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, conversion from ASCII 8 to EBCDIC or from Shift-JIS to IBM-Kanji).

Gateway-Library uses translation 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 translation tables or rename the JCM at your site. See the Mainframe Connect Server Option *Installation and Administration Guide* for details. This section contains the following subsections:

- 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 shows which single-byte character sets (SBCS) are supported at the workstation.

Table 2-1: SBCS 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 shows which double-byte character sets (DBCS) 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 shows which character sets 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 54 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. This section contains the following subsections:

- What is a cursor?
- Benefits of using cursors

- How cursors work in Open ServerConnect
- Types of cursor commands
- CURSOR-DESC structure

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

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 prepared 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, 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: <ul style="list-style-type: none"> • 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-LibraryThinSpace 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 <i>n</i> 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-LibraryThinSpace transaction calls:

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` (`CS-CUR-STATUS`)), 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: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • 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 *SYGWC0B* as follows:

```

CURSOR-ID PIC S9(9) USAGE COMP SYNC.
NUMBER-OF-UPDATE-COLUMNS PIC S9(9) USAGE COMP SYNC.
FETCH-COUNT PIC S9(9) USAGE COMP SYNC.
CURSOR-STATUS PIC S9(9) USAGE COMP SYNC.
CURSOR-COMMAND PIC S9(9) USAGE COMP SYNC.
COMMAND-OPTIONS PIC S9(9) USAGE COMP SYNC.
FETCH-TYPE PIC S9(9) USAGE COMP SYNC.
ROW-OFFSET PIC S9(9) USAGE COMP SYNC.
CURSOR-NAME-LENGTH PIC S9(9) USAGE COMP SYNC.
CURSOR-NAME PIC X(30).
TABLE-NAME-LENGTH PIC S9(9) USAGE COMP-SYNC.
TABLE-NAME PIC X(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.	<p>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 <i>CURSOR-NAME</i> as a required parameter.</p> <p>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.</p> <p>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.</p>
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.	<p>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:</p> <ul style="list-style-type: none"> - 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 <p>Requests to Open ServerConnect always have a FETCH-TYPE of TDS-NEXT.</p>
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*)

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.
TDS-CURSTAT-ROWCNT	The cursor specified the number of rows that should be returned for the TDS-CURSOR-FETCH command.

Value	Meaning
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 the following table. TDCURPRO can update this field with the next command to process for a given cursor. Table 2-7 on page 29 also lists the relevant *COMMAND-OPTIONS* values.

Table 2-7: Values for *CURSOR-COMMAND* and *COMMAND-OPTIONS*

<i>CURSOR-COMMAND</i> value	Meaning	Legal values for <i>COMMAND-OPTIONS</i>
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.	<p>TDS-CURSOR-SETROWS when the client is describing the current row fetch count. The FETCH-COUNT field contains the requested fetch count.</p> <p>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.</p> <p>TDS-CURSOR-INFORMSTATUS when the Open ServerConnect application is responding to a TDS-CURSOR-INFO command. The CURSOR-STATUS field contains the cursor status.</p>
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 handler 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 on page 32 summarizes the valid exchange of cursor requests and responses between a client and an Open ServerConnect application.

The forward arrows indicate that *ACTION* is set to TDS-GET and the application is retrieving information from the client. The backward arrows indicate that *ACTION* is set to TDS-SET and the application is sending information to the client.

Table 2-8: Valid cursor requests and responses

Client action	Open ServerConnect application response
<p>Declares a cursor.</p> <p>(CURSOR-COMMAND field of <i>CURSOR-DESC</i> contains TDS-CURSOR-DECLARE).</p>	<p>→ Retrieve CURSOR-COMMAND value from <i>CURSOR-DESC</i>. (TDCURPRO)</p> <p>→ Retrieve number of cursor parameters, if any. (TDNUMPRM)</p> <p>→ Retrieve format of cursor parameters, if any. (TDINFPRM)</p> <p>→ Retrieve actual text of cursor command. (TDRCVSQL)</p> <p>← Set cursor ID. Set CURSOR-ID field to unique cursor ID. (TDCURPRO)</p> <p>← Send a DONE packet. (TDSNDDON with <i>STATUS</i> argument set to TDS-DONE-FINAL)</p>
<p>Requests the status of the current cursor or sends a fetch count.</p> <p>(CURSOR-COMMAND field of <i>CURSOR-DESC</i> contains TDS-CURSOR-INFO).</p>	<p>→ Retrieve CURSOR-COMMAND, CURSOR-ID, and COMMAND-OPTIONS values from <i>CURSOR-DESC</i> structure. (TDCURPRO)</p> <p>← Send number of rows to be returned per fetch, if client set COMMAND-OPTIONS field to TDS-CURSOR-SETROWS. (TDCURPRO)</p> <p>← 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).</p> <p>← Send a DONE packet. (TDSNDDON with <i>STATUS</i> argument set to TDS-DONE-FINAL).</p> <p><i>Note:</i></p> <p>If the client request is <i>ct_cmd_props</i> with cursor options, then CURSOR-COMMAND field is TDS-CURSOR-INFO with TDS-CURSOR-ASKSTATUS option.</p> <p>If the client request is <i>ct_cursor</i> (CS-CURSOR-ROWS), then CURSOR-COMMAND field is TDS-CURSOR-INFO with TDS-CURSOR-SETROWS option.</p>

Client action	Open ServerConnect application response
<p>Opens a cursor.</p> <p>(CURSOR-COMMAND field of <i>CURSOR-DESC</i> contains TDS-CURSOR-OPEN).</p>	<p>→ Retrieve CURSOR-COMMAND and CURSOR-ID values from <i>CURSOR-DESC</i> structure. (TDCURPRO)</p> <p>→ Retrieve number of cursor parameters, if any. (TDNUMPRM)</p> <p>→ Retrieve format of cursor parameters and actual parameters, if any. (TDINFPRM, TDRCVPRM)</p> <p>← Send cursor status. Set <i>CURSOR-ID</i> to current cursor ID. (TDCURPRO)</p> <p>← Describe result row formats. (TDESCRIB with <i>TYPE</i> argument set to TDS-ROWDATA).</p> <p>← Send a DONE packet. (TDSNDDON with <i>STATUS</i> argument set to TDS-DONE-FINAL).</p>
<p>Fetches rows.</p> <p>(CURSOR-COMMAND field of <i>CURSOR-DESC</i> contains TDS-CURSOR-FETCH).</p>	<p>→ Retrieve CURSOR-COMMAND and CURSOR-ID values from <i>CURSOR-DESC</i> structure. (TDCURPRO)</p> <p>← Send result rows, <i>FETCH-COUNT</i> times. (TDSNDROW)</p> <p>← Send a DONE packet. (TDSNDDON with <i>STATUS</i> argument set to TDS-DONE-FINAL).</p>
<p>Sends a cursor close command.</p> <p>(CURSOR-COMMAND field of <i>CURSOR-DESC</i> contains TDS-CURSOR-CLOSE).</p>	<p>→ Retrieve CURSOR-COMMAND and CURSOR-ID values from <i>CURSOR-DESC</i> structure. (TDCURPRO)</p> <p>← Send cursor status. Open ServerConnect sets cursor status, not the application. (TDCURPRO)</p> <p>← Send a DONE packet. (TDSNDDON with <i>STATUS</i> argument set to TDS-DONE-FINAL).</p>
<p>Updates a cursor.</p> <p>CURSOR-COMMAND field of <i>CURSOR-DESC</i> contains TDS-CURSOR-UPDATE).</p>	<p>→ Retrieve CURSOR-COMMAND and CURSOR-ID values from <i>CURSOR-DESC</i> structure. (TDCURPRO)</p> <p>→ Retrieve actual text of cursor command. (TDRCVSQL)</p> <p>← Send a DONE packet. (TDSNDDON with <i>STATUS</i> argument set to TDS-DONE-FINAL).</p>

Client action	Open ServerConnect application response
Deletes a cursor.	→ Retrieve CURSOR-COMMAND and CURSOR-ID values from <i>CURSOR-DESC</i> structure. (TDCURPRO)
CURSOR-COMMAND field of <i>CURSOR-DESC</i> contains TDS-CURSOR-DELETE).	← Send a DONE packet. (TDSNDDON with <i>STATUS</i> 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-INT, 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 SBCS to be used at the mainframe.
- Whether DB2 LONG VARCHAR data strings with lengths greater than 255 bytes are truncated or rejected when sent to a client
- 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. The section “Binary and decimal datatypes” on page 42 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 lists those datatypes and their Gateway-Library equivalents.

Table 2-9: Open ServerConnect datatypes

Open ServerConnect datatype and COBOL data descriptions	Client-Library/C and DB-Library datatypes	Datatype descriptions
TDSBINARY 01 BINVAL PIC X(1) 01 BINVALMAX PIC X(255)	CS_BINARY DBBINARY	Fixed binary type. No translations are performed on this datatype.
TDSCHAR 01 CHARVAL PIC X(1) 01 CHVALMAX PIC X(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 DATTIM PIC X(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 PIC X(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 PIC S9(4) COMP-1	CS_REAL DBREAL	4-byte single precision type.
TDSFLT8 01 FLT8VAL PIC S9(9) COMP-2	CS_FLOAT DBFLT8	8-byte double precision type.
TDSGRAPHIC 01 GRAPHVAL PIC X(254) or 01 GRAPHVAL PIC G(127)	(Not applicable)	1- to 127-character fixed character type. Used at the mainframe only to represent Japanese double-byte characters.
TDSIMAGE 01 IMAGEVALPIC X(1) 01 IMGVALMAXPIC X(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.
TDSINT2 01 INT2VAL PIC X(2) or 01 INT2VAL PIC S9(4) COMP	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.

Open ServerConnect datatype and COBOL data descriptions	Client-Library/C and DB-Library datatypes	Datatype descriptions
<p>TDSINT4</p> <p>01 INT4VAL PIC X(4) or 01 INT4VAL PIC S9(9) COMP</p>	<p>CS_INT DBINT</p>	<p>4-byte integer. Can be declared as numeric or character. When declared as numeric, it has a maximum value of 2,147,483,648.</p>
<p>TDSLONGBINARY</p> <p>01 LONGBINVALPIC X(1) 01 LBINVALMAXPIC X(32000)</p>	<p>CS_LONGBINARY</p>	<p>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.</p>
<p>TDSLONGBINARY</p> <p>01 LONGCHARVAL PIC X(1) 01 LCHARVALMAXPIC X(32000)</p>	<p>CS_LONGCHAR</p>	<p>Long variable character type. The default maximum length for this datatype is 32K. Does not include the 2-byte ("LL") length specification prefix.</p>
<p>TDSMONEY</p> <p>01 MONEY-GROUP 05 MON-HIPIC S9(9) COMP 05 MON-LOPIC S9(9) COMP or 01 MONEYVALPIC X(8)</p>	<p>CS_MONEY DBMONEY</p>	<p>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. <i>Note:</i> This datatype can be used with client data only.</p>
<p>TDSMONEY4</p> <p>01 MONEY4VAL PIC X(4)</p>	<p>CS_MONEY4 DBMONEY4</p>	<p>4-byte double precision type. The range of legal values for this datatype is: -\$214,748.3648 to +\$214,748.3647 <i>Note:</i> This datatype can be used with client data only.</p>

Open ServerConnect datatype and COBOL data descriptions	Client-Library/C and DB-Library datatypes	Datatype descriptions
TDSNUMERIC 01 NUMVAL 05 PRECISION PIC X(1) 05 SCALE PIC X(1) 05 ARR PIC X(33)	CS_NUMERIC	Numeric type. Support for numbers with: <i>Precision</i> – the precision of the numeric value (1 to 77, default 18). <i>Scale</i> – the scale of the numeric value (0 to 77, default 0). <i>Note:</i> This is a Sybase datatype. This datatype can be used for client data only.
TDS-PACKED-DECIMAL 01 PDECVALPIC S9(N) V9(M) COMP-3	CS_PACKED370	IBM/370 packed decimal type. 31-byte precision is supported. <i>Note:</i> This is a mainframe datatype and can be used for mainframe data only.
TDS-SYBASE-DECIMAL 01 NUMVAL 05 PRECISION PIC X(1) 05 SCALE PIC X(1) 05 ARR PIC X(33)	CS_DECIMAL	Decimal type. Support for numbers with: <i>Precision</i> – the precision of the numeric value (1 to 77, default 18). <i>Scale</i> – scale of the numeric value (0 to 77, default 0). <i>Note:</i> This is a Sybase datatype. This datatype can be used for client data only.
TDSTEXT 01 TEXTVAL PIC X(1) 01 TEXTVALMAX PIC X(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.

Open ServerConnect datatype and COBOL data descriptions	Client-Library/C and DB-Library datatypes	Datatype descriptions
<p>TDSVARYBIN</p> <p>01 VARBIN-GROUPVAL 05 VBIN-LEN PIC S9(4) COMP 05 VBIN-VAL PIC X(1)</p> <p>01 VARBIN-GROUPVALMAX 05 VBMAX-LEN PIC S9(4) COMP 05 VBMAX-VAL PIC X(255)</p>	<p>CS_VARBINARY DBVARYBIN</p>	<p>1- to 255-byte variable binary type.</p> <p>The field length is stored in a 2-byte (“LL”) prefix, as in DB2. No translations are performed on this datatype.</p>
<p>TDSVARYCHAR</p> <p>01 VARCHAR-GROUPVAL 05 VCHAR-LEN PIC S9(4) COMP 05 VCHAR-VAL PIC X(1)</p> <p>01 VARCHAR-GROUPVALMAX 05 VCMAX-LEN PIC S9(4) COMP 05 VCMAX-VAL PIC X(255)</p>	<p>CS_VARCHAR DBVARYCHAR</p>	<p>1- to 255-byte variable character type.</p> <p>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.</p>
<p>TDSVARYGRAPHIC</p>	<p>(Not applicable)</p>	<p>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.</p> <p><i>Note:</i> This is a mainframe datatype and can be used with mainframe data only.</p>
<p>TDSVOID</p>	<p>(Not applicable)</p>	<p>NULL or nonexistent parameter. Used to denote parameters omitted in stored procedures.</p> <p>For example, in the following RPC: exec rpc a, b, , d, e</p> <p>The missing c parameter is represented as TDSVOID.</p> <p>No translations are done on this datatype.</p>

Character datatypes

The following subsections contain additional information about character datatypes.

TDSVARYCHAR

Always use TDSVARYCHAR rather than TDSVARCHAR. DBVARCHAR 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 PIC X(1) .
   05 SCALE     PIC X(1) .
   05 ARR       PIC 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:

$$\text{Result Length} = (2 * \text{Source Length}) - 1.$$

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 1 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

Table 2-10 lists decimal-to-character conversions.

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

Decimal value	Precision	Scale	Calculation $P+s+d+z = \text{result length}$	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

Decimal value	Precision	Scale	Calculation P+s+d+z = result length	Character-type result	Result length
.123	3	3	3+1+1+1 = 6	ThinSpace0.123	6
-.123	3	3	3+1+1+1 = 6	ThinSpace-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, TDSOVERFLOW 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 “Character set length requirements” on page 62.

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 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)	<ol style="list-style-type: none">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)	<ol style="list-style-type: none">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)	<ol style="list-style-type: none">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)	<ol style="list-style-type: none">1. Get operation type (TDS-GET) (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) <p>[application logic: execute client request]</p> <ol style="list-style-type: none">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)	<ol style="list-style-type: none">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) <p>[application logic: execute client request]</p> <ol style="list-style-type: none">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)	<ol style="list-style-type: none">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 request, 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 108 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 IMS TM Users: IMS TM WFI transactions can accept additional RPCs after receiving TDS-RESET. See TDSTATUS on page 227 for details.

Calls in a long-running transaction

The pattern of calls in a long-running transaction follows the proceeding subsection descriptions.

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: 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 the *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.
Sybase recommends 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 notabend 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, SYCTSAX5, 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 53 for details.

The following options are set during customization:

- The native language used at the mainframe
- DBCS: whether double-byte kanji characters are used
- Information about SBCSs

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 translation 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 the 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.

See Table 2-14 on page 63 and Table 2-15 on page 64 for information on character set differences in tabular form.

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:

- CHAR
- 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

Table 2-13 lists the datatypes that 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.	<i>IBM Kanji</i> : Uses SO/SI with double-byte characters. <i>EUC-JIS</i> : 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.

- 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 on page 63 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	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	EUC-JIS hankaku katakana	character	8
IBM Kanji hankaku katakana	character	4	Shift-JIS hankaku katakana	character	4
IBM Kanji kanji	character	10	EUC-JIS 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	EUC-JIS 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 196 for details.

See TDGETSOI on page 104 for information about how to query the SO/SI processing settings for a column or parameter.

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

Open ServerConnect supports the following Gateway-Library functions listed in Table 3-1.

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	Retrieves or sets information about a cursor.
TDESCRIB	Describes a row column and binds its associated host program variable.
TDFREE	Frees up the <i>TDPROC</i> 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 a 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 are 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.
TDSQLEN	Gets the length of a SQL statement string received from a remote client.
TDSTATUS	Retrieves status information about the client/server connection.
TDTERM	Terminates a program and frees up all MVS storage.
TDYNAMIC	Reads or responds 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 function, topics, and documentation. All arguments of each function are required unless designated as optional.

Most examples in this chapter are taken from the sample programs in the appendices. These programs show how individual functions are coded in context. Refer to the appendices to learn how to set up your WORKING STORAGE SECTION 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 COBOL Gateway-Library interface. To include these constants in a COBOL program, include the copybook SYGWCOB 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, including 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

```
COPY SYGWCOB.
01 TDPROC      PIC S9(9) USAGE COMP SYNC.
01 RETCODE    PIC S9(9) USAGE COMP SYNC.
01 IHANDLE    PIC S9(9) USAGE COMP SYNC.
01 ACCEPT-CONNECTION-NAME PIC X(8) VALUE IS SPACES.
01 ERROR-SUBCODE      PIC S9(9) USAGE COMP SYNC.

CALL 'TDACCEPT' USING 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 their *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 70.

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)	<p>Gateway-Library found the DBCS specified by the client, but the corresponding double-byte module was not loaded at the mainframe.</p> <p>This code is returned 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.</p> <p>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.</p>
TDS-CHARSETSRV-NOT-SBCS (-264)	<p>The client character set was not found; DBCS specified as default.</p> <p>This code represents two problems:</p> <ol style="list-style-type: none"> 1 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). 2 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)	<p>Connection abended. The client/server connection abnormally ended (for example, the LU 6.2 session crashed or the remote transaction abended).</p>
TDS-DBCS-CHARSET-NOTFOUND (-263)	<p>Gateway-Library could not find the DBCS specified in the client login packet.</p> <p>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.</p>
TDS-DEFAULT-CHARSET-NOTFOUND (-262)	<p>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 single-byte character sets only.</p>
TDS-GWLIB-UNAVAILABLE (-15)	<p>Could not load SYGWCICS (the Gateway-Library phase).</p>
TDS-INVALID-IHANDLE (-19)	<p>Invalid <i>IHANDLE</i> specification. Error in specifying a value for the <i>IHANDLE</i> argument.</p>

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 <i>TDPROC</i> 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, TDRESULT, TDSNDDON, and TDFREE at the beginning and end of a Gateway-Library program. This example is taken from the sample program, SYCCSAR2, in Appendix B, "Sample RPC Application for CICS."

```

*   Establish gateway environment

CALL 'TDINIT' USING DFHEIBLK, GWL-RC, GWL-INIT-HANDLE.

*   Accept client request

CALL 'TDACCEPT' USING GWL-PROC, GWL-RC, GWL-INIT-HANDLE, SNA-CONNECTION-
NAME, SNA-SUBC.

*   TDRESULT to make sure we were started via RPC request

CALL 'TDRESULT' USING GWL-PROC, GWL-RC.
    IF GWL-RC NOT = TDS-PARM-PRESENT
    THEN PERFORM TDRESULT-ERROR
    GO TO END-PROGRAM
    END-IF.

*   -----
*   body of program
*   -----
*   -----
END-PROGRAM.
```

```

* -----
  IF SEND-DONE-OK
    MOVE TDS-DONE-COUNT TO WRK-DONE-STATUS
  ELSE
    MOVE TDS-DONE-ERROR TO WRK-DONE-STATUS
    MOVE ZERO           TO PARM-RETURN-ROWS
  END-IF.
  CALL 'TDSNDDON' USING GWL-PROC, GWL-RC, WRK-DONE-STATUS,
                    PARM-RETURN-ROWS, TDS-ZERO,
                    TDS-ENDRPC.
  CALL 'TDFREE' USING GWL-PROC, GWL-RC.
  EXEC CICS RETURN END-EXEC.

```

Example 2

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

```

* -----
*   establish gateway environment
* -----
  CALL 'TDINIT' USING IO-PCB, GWL-RC, GWL-INIT-HANDLE.

[check return code]

* -----
*   set program type to MPP
* -----
  CALL 'TDSETPT' USING  GWL-INIT-HANDLE, GWL-RC, GWL- PROG-TYPE,
                    GWL-SPA-PTR, TDS-NULL, TDS- NULL.

[check return code]

* -----
*   accept client request
* -----
  CALL 'TDACCEPT' USING  GWL-PROC, GWL-RC, GWL-INIT-HANDLE,
                    SNA-CONNECTION-NAME,
                    SNA-SUBC.

* -----
  READ-IN-USER-PARM.
* -----

```

Usage

- A mainframe server application uses TDACCEPT 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), or the name of the Open Server from the *interfaces* file (in a two-tier environment).

Your program can retrieve 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 translations

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

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

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 double-byte character sets (DBCS) such as kanji are supported.
- The treatment of single-byte character sets, when DBCS are supported. This determines 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.

When TDACCEPT 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 translate 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 double-byte character set 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.

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

When the character set is double-byte:

- If the login packet does not specify a character set or specifies one that Gateway-Library cannot match, TDACCEPT fails and returns TDS-DBCS-CHARSET-NOTFOUND.

- If 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.

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 double-byte character set 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 95
- TDINIT on page 142

Related topics

- “Character sets” on page 17
- “The login packet” on page 53
- “Processing Japanese client requests” on page 58
- “Customization” on page 58

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 single-byte character sets. Do not use it with double-byte character sets.

Syntax

COPY SYGWCORB.

```
01 TDPROC          PIC S9(9)  USAGE COMP SYNC.
01 RETCODE         PIC S9(9)  USAGE COMP SYNC.
01 NUM-DECIMAL-PLACES PIC S9(9)  USAGE COMP SYNC.
01 SOURCE-TYPE     PIC S9(9)  USAGE COMP SYNC.
01 SOURCE-LENGTH   PIC S9(9)  USAGE COMP SYNC.
01 SOURCE-VARIABLE PIC X(n).
01 RESULT-TYPE     PIC S9(9)  USAGE COMP SYNC.
01 RESULT-LENGTH   PIC S9(9)  USAGE COMP SYNC.
01 RESULT-VARIABLE PIC X(n).
01 OUTLEN          PIC S9(9)  USAGE COMP SYNC.
                    (optional)
```

```
CALL 'TDCONVRT' USING 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 78.

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 and 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.
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 <i>RESULT-LENGTH</i> argument is too short. The length must be greater than zero.

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 <i>TDPROC</i> argument.
TDS-INVALID-VAR-ADDRESS (-175)	Specified variable address is invalid. No variable with the specified name exists. A <i>NULL</i> 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 shows two methods of converting datatypes. One method uses `TDESCRIB` to convert data from the DB2 datatype `DECIMAL` (`TDSDECIMAL`) to `TDSFLT8`. The other method uses `TDCONVRT` to convert data from the DB2 datatype `DECIMAL` (`TDSDECIMAL`) to the DB-Library datatype `DBMONEY` (`TDSMONEY`).

This example is taken from the sample program, `SYCCSAR2`, in Appendix B, “Sample RPC Application for CICS” which runs under CICS.

* Here we let `TDESCRIB` convert from `TDSDECIMAL` to `TDSFLT8`.

```
CALL 'SYGETAD' USING DB-DESCRIBE-HV-PTR, EMPLOYEE-JC.
CALL 'SYGETAD' USING DB-COLUMN-NAME-HV-PTR, CN-JC.
MOVE LENGTH OF EMPLOYEE-JC TO WRKLEN1.
MOVE LENGTH OF CN-JC      TO WRKLEN2.
MOVE TDSDECIMAL          TO DB-HOST-TYPE.
MOVE TDSFLT8             TO DB-CLIENT-TYPE.
PERFORM DESCRIBE-COLUMN.
```

* We must inform the Server Library how many decimal places
* are in the `EMPLOYEE-JC` column.

```
CALL 'TDSETBCD' USING GWL-PROC, GWL-RC, TDS-OBJECT-COL,
CTR-COLUMN, TDS-DEFAULT-LENGTH,
GWL-SETBCD-SCALE.
```

* Demonstrate getting decimal column information.

Usage

- A server application uses this function to convert from a mainframe datatype to a datatype that can be used by an Open Client 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 87 and TDSETPRM on page 188.

- 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 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.
TDSCHAR	TDSLONGVARCHAR	
TDSCHAR	TDSMONEY	Pads TDSCHAR fields with blanks.
TDSCHAR	TDS-SYBASE-DECIMAL	When converting TDSCHAR to Sybase numeric and decimal, specify 35 as destination length.
TDSCHAR	TDS-PACKED-DECIMAL	
TDSCHAR	TDSCHAR	
TDSVARYCHAR	TDSLONGVARCHAR	<i>OUTLEN</i> shows the actual length.
TDSVARYCHAR	TDSMONEY	
TDSVARYCHAR	TDSCHAR	
TDSLONGVARCHAR	TDSTEXT	
TDSLONGVARCHAR	TDSVARYCHAR	
TDSLONGVARCHAR		

Source datatype	Result datatype	Notes
TDSDATETIME	TDSCHAR	
TDSDATETIME4	TDSCHAR	
TDSFLT4	TDSFLT8	Pads with zeros.
TDSFLT4	TDSMONEY	
TDSFLT4	TDSMONEY4	
TDSFLT8	TDSFLT4	Truncates low order digits.
TDSFLT8	TDSMONEY	
TDSFLT8	TDSMONEY4	
TDSGRAPHIC	TDSCHAR	Used with Japanese double-byte character sets.
TDSGRAPHIC	TDSVARYCHAR	
TDSVARYGRAPHIC	TDSCHAR	Pads TDSCHAR fields with blanks.
TDSVARYGRAPHIC	TDSVARYCHAR	
TDSLONGVARBIN	TDSIMAGE	
TDSNUMERIC	TDSCHAR	When converting Sybase numeric and decimal to char, specify destination length as precision +2, or precision +3 if precision=scale for leading zero.
TDSNUMERIC	TDSPACKED-DECIMAL	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 S9(10) v9(5).
TDS-PACKED-DECIMAL	TDS-CHAR	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	TDSNUMERIC	
TDS-PACKED-DECIMAL	TDSFLT4	
TDS-PACKED-DECIMAL	TDSFLT8	
TDS-PACKED-DECIMAL	TDS-SYBASE-DECIMAL	When converting TDS-PACKED-DECIMAL to Sybase numeric and decimal, specify 35 as the destination length. <i>OUTLEN</i> shows the actual length of the numeric field.
TDS-SYBASE-DECIMAL	TDSCHAR	When converting Sybase numeric and decimal to char, specify destination length as precision +2, or precision +3 if precision=scale for leading zero.
TDS-SYBASE-DECIMAL	TDSPACKED-DECIMAL	

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.

If you are using DB2:

- 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 TDSLONGBINARY. When converting long varchar data to a client datatype, you have three options:

Option 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.

Option 2: If the client is an Open Client 10.0 program, you can send the data as TDSLONGBINARY. The Client-Library datatype CS-LONGBINARY has a maximum length of 2,147,483,647 bytes.

Option 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 long 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 TDSLONGBINARY. The Client-Library datatype CS-LONGBINARY has a maximum length of 2,147,483,647 bytes.

See also

Related functions

- TDESCRIB on page 87

- TDRCVPRM on page 154
- TDSETBCD on page 174
- TDSETPRM on page 188

Related topics

- “Datatypes” on page 36

Related documents

- Open Client DB-Library *Reference Manual*

TDCURPRO

Description Retrieves or sets information about a cursor.

Syntax COPY SYGWCOB.

```
01 TDPROC PIC S9(9) USAGE COMP SYNC.
01 RETCODE PIC S9(9) USAGE COMP SYNC.
01 ACTION PIC S9(9) USAGE COMP SYNC.
01 CURSOR-DESC FROM SYGWCOB.
```

```
CALL 'TDCURPRO' USING TDPROC, RETCODE, ACTION,
CURSOR-DESC.
```

The CURSOR-DESC structure is defined in SYGWCOB as follows:

```
CURSOR-ID          PIC S9(9) USAGE COMP SYNC.
NUMBER-OF-UPDATE-COLUMNS PIC S9(9) USAGE COMP SYNC.
FETCH-COUNT        PIC S9(9) USAGE COMP SYNC.
CURSOR-STATUS      PIC S9(9) USAGE COMP SYNC.
CURSOR-COMMAND     PIC S9(9) USAGE COMP SYNC.
COMMAND-OPTIONS    PIC S9(9) USAGE COMP SYNC
FETCH-TYPE         PIC S9(9) USAGE COMP SYNC.
ROW-OFFSET         PIC S9(9) USAGE COMP SYNC.
CURSOR-NAME-LENGTH PIC S9(9) USAGE COMP SYNC.
CURSOR-NAME        PIC X(30).
TABLE-NAME-LENGTH  PIC S9(9) USAGE COMP-SYNC.
TABLE-NAME         PIC X(30).
```

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 84.

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.

Return value	Meaning
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 <i>ACTION</i> 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.
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 <i>OPTION</i> 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 <i>OPTION</i> 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.

Return value	Meaning
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 <i>TDPROC</i> 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

- An Open ServerConnect application uses this function 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 their responses are discussed under “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 which 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,

- Specify TDS-SET for the *ACTION* argument.

- Specify the new cursor ID in the CURSOR-ID field in the *CURSOR-DESC* structure.
- Return this information to the client.

Note Both the client and the Open ServerConnect applications must subsequently refer to this cursor by its ID rather than 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.

Note This is done by Open ServerConnect, not by the application.

- 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 70
- TDGETREQ on page 98

TDESCRIB

Description	Describes a column in a result row and the mainframe server program variable where it is stored.
Syntax	COPY SYGWCOD. 01 TDPROC PIC S9(9) USAGE COMP SYNC. 01 RETCODE PIC S9(9) USAGE COMP SYNC. 01 COLUMN-NUMBER PIC S9(9) USAGE COMP SYNC. 01 HOST-VARIABLE-TYPE PIC S9(9) USAGE COMP SYNC. 01 HOST-VARIABLE-MAXLEN PIC S9(9) USAGE COMP SYNC.

```
01 HOST-VARIABLE-NAME    PIC X(n).
01 NULL-INDICATOR-VARIABLE PIC S9(4) USAGE COMP SYNC.
01 NULLS-ALLOWED        PIC S9(9) USAGE COMP SYNC.
01 COLUMN-TYPE           PIC S9(9) USAGE COMP SYNC.
01 COLUMN-MAXLEN         PIC S9(9) USAGE COMP SYNC.
01 COLUMN-NAME           PIC X(n).
01 COLUMN-NAME-LENGTH    PIC S9(9) USAGE COMP SYNC.

CALL 'TDESCRIB' USING 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 90.

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 (TDSINTn, TDSFLTn), this argument is ignored.

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.
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 <i>COLUMN-MAXLEN</i> 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 <i>TDPROC</i> 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.

This example is taken from the sample program, SYCCSAR2, in Appendix B, “Sample RPC Application for CICS.”

```
* Here we let TDESCRIB convert from DB2 varchar (TDSVARYCHAR)
* to DECHAR.

CALL 'SYGETAD' USING DB-DESCRIBE-HV-PTR, EMPLOYEE-ED.
CALL 'SYGETAD' USING DB-COLUMN-NAME-HV-PTR, CN-ED.
MOVE LENGTH OF EMPLOYEE-ED TO WRKLEN1.
MOVE LENGTH OF CN-ED      TO WRKLEN2.
MOVE TDSINT2              TO DB-HOST-TYPE.
MOVE TDSINT2              TO DB-CLIENT-TYPE.
PERFORM DESCRIBE-COLUMN.

* Get the user defined datatype of EMPLOYEE-ED column.

CALL 'TDINFUDT' USING GWL-PROC, GWL-RC, CTR-COLUMN,
                  GWL-INFUDT-USER-TYPE.

* Set the user defined datatype of EMPLOYEE-ED column.

CALL 'TDSETUDT' USING GWL-PROC, GWL-RC, CTR-COLUMN,
                  GWL-INFUDT-USER-TYPE.
*-----
DESCRIBE-COLUMN.
*-----

SET ADDRESS OF LK-DESCRIBE-HV      TO DB-DESCRIBE-HV-PTR.
SET ADDRESS OF LK-COLUMN-NAME-HV  TO DB-COLUMN-NAME-HV-PTR.
ADD 1                             TO CTR-COLUMN.

CALL 'TDESCRIB' USING GWL-PROC, GWL-RC, CTR-COLUMN,
                    DB-HOST-TYPE, WRKLEN1, LK-DESCRIBE-HV,
                    DB-NULL-INDICATOR, TDS-FALSE,
                    DB-CLIENT-TYPE, WRKLEN1,
LK-COLUMN-NAME-HV, WRKLEN2.
```

Usage

- 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.
- You must use a separate TDESCRIB call for each column in a row. Thus, if a row has 12 columns of data, you must call TDESCRIB 12 times, once for each column.
- Columns can be described in any order.

- The maximum number of columns that can be returned to a client is 255.

Note Applications should check the return code after each TDESCRIB to see whether any data conversion errors occurred. This is especially important with applications that convert decimal or floating point data before returning it to the client.

- There can be only one TDESCRIB call for each column. If you try to describe the same column twice, the operation fails and returns TDS-DUPLICATE-ENTRY.
- After all the columns in a row are described, the server application calls TDSNDROW once for each row of data to be sent to the client.

Each TDSNDROW call retrieves the data from every variable named in the *HOST-VARIABLE-NAME* arguments of the preceding TDESCRIB calls, and returns that data to the client in the associated columns. Do not call TDSNDROW until all columns in the row are associated with a variable, using TDESCRIB.

- An application can only call TDESCRIB before it sends rows to a client. Do not call TDESCRIB once your program starts to send rows.
- The length of columns with datatypes that have “VARY” in the name (TDSVARYCHAR, TDSVARYBIN, TDSVARYGRAPHIC) is derived from the 2-byte “LL” length specification in the named variable structure.

TDSLONGVARCHAR columns do not have “LL” length specifications.

Always specify a column length of 35 when describing Sybase numeric and decimal columns.

- TDESCRIB automatically converts some mainframe datatypes to Open Client datatypes before returning data to a Sybase client. TDESCRIB sets up the conversion and then performs the conversion when the row is sent to the client by TDSNDROW.

TDESCRIB pads binary-type columns with zeros and character-type columns with blanks; no default padding is set for columns of other datatypes.

You can perform additional datatype conversions (to text and image datatypes, for example) by calling TDCONVRT.

- TDS versions earlier than 4.2 do not support short float (TDSFLT4), short money (TDSMONEY4), or short datetime (TDSDATETIME4) datatypes. Gateway-Library automatically converts short float and money values to TDSFLT8 and TDSMONEY before returning data to a client using earlier versions of TDS. Gateway-Library does not convert short datetime datatypes.

TDINFPGM returns the version of TDS in use.

Datatype conversions

Table 3-7 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	When converting TDSCHAR to Sybase numeric and decimal, specify 35 as destination length.
TDSCHAR	TDS-SYBASE-DECIMAL	
TDSVARYCHAR	TDSCHAR	<i>OUTLEN</i> shows the actual length.
TDSVARYCHAR	TDSLONGVARCHAR	
TDSVARYCHAR	TDSMONEY	
TDSLONGVARCHAR	TDSCHAR	
TDSLONGVARCHAR	TDSTEXT	
TDSLONGVARCHAR	TDSVARYCHAR	
TDSDATETIME	TDSCHAR	
TDSDATETIME4	TDSCHAR	
TDSFLT4	TDSFLT8	Pads with zeros.
TDSFLT4	TDSMONEY	
TDSFLT4	TDSMONEY4	
TDSFLT8	TDSFLT4	Truncates low order digits.
TDSFLT8	TDSMONEY	
TDSFLT8	TDSMONEY4	
TDSGRAPHIC	TDSCHAR	Used with Japanese double-byte character sets.
TDSGRAPHIC	TDSVARYCHAR	
TDSVARYGRAPHIC	TDSCHAR	Pads TDSCHAR fields with blanks.
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 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 S9(10) v9(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. <i>OUTLEN</i> 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:

$$\text{Result Length} = (2 * \text{Source Length}) - 1$$

- 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 may be different from the length of the column name at the mainframe.

To learn more, see the discussion of length considerations in “Processing Japanese client requests” on page 58.

See also

Related functions

- TDCONVRT on page 77
- TDSNDROW on page 220

Related topics

- “Datatypes” on page 36
- “Processing Japanese client requests” on page 58

TDFREE

Description	Frees up a previously allocated <i>TDPROC</i> structure after returning results to a client.
Syntax	COPY SYGWCOB. 01 TDPROC PIC S9(9) USAGE COMP SYNC. 01 RETCODE PIC S9(9) USAGE COMP SYNC. CALL 'TDFREE' USING TDPROC, RETCODE.
Parameters	<i>TDPROC</i> (I) Handle for this client–server connection. This must be the same value specified in the associated TDACCEPT call. The <i>TDPROC</i> 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.

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 <i>TDPROC</i> 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, SYCCSAR2, in Appendix B, "Sample RPC Application for CICS."

```

*   Establish gateway environment
CALL 'TDINIT' USING DFHEIBLK, GWL-RC, GWL-INIT-HANDLE.
*   Accept client request
CALL 'TDACCEPT' USING GWL-PROC, GWL-RC, GWL-INIT-HANDLE,
SNA-CONNECTION-NAME, SNA-SUBC.
*   TDRESULT to make sure we were started via RPC request
CALL 'TDRESULT' USING GWL-PROC, GWL-RC.
IF GWL-RC NOT = TDS-PARM-PRESENT THEN
    PERFORM TDRESULT-ERROR
    GO TO END-PROGRAM
END-IF.

* -----
* body of program
* -----
* -----
END-PROGRAM.
* -----
IF SEND-DONE-OK
    MOVE TDS-DONE-COUNT TO WRK-DONE-STATUS
ELSE
    MOVE TDS-DONE-ERROR TO WRK-DONE-STATUS
    MOVE ZERO           TO PARM-RETURN-ROWS
END-IF.
CALL 'TDSNDDON' USING GWL-PROC, GWL-RC, WRK-DONE-STATUS,
                    PARM-RETURN-ROWS, TDS-ZERO,
                    TDS-ENDRPC.

```

```
CALL 'TDFREE' USING GWL-PROC, GWL-RC.
EXEC CICS RETURN END-EXEC.
```

Example 2

This code fragment shows the use of TDFREE and TDTERM in a transaction that uses the IMS TM implicit API. This transaction processes multiple client requests, using TDGETREQ to call each request after the first. This example is taken from the sample program in Appendix D, “Sample RPC Application for IMS TM (Implicit).”

```
* -----
* Get next client request
* -----
MOVE TDS-TRUE TO GWL-WAIT-OPTION.
MOVE ZEROES TO GWL-REQ-TYPE.
MOVE SPACES TO GWL-RPC-NAME.
CALL 'TDGETREQ' USING GWL-PROC, GWL-RC, GWL-WAIT-OPTION,
                      GWL-REQ-TYPE, GWL-RPC-NAME.
EVALUATE GWL-RC
  WHEN ZEROES
    GO TO READ-IN-USER-PARM
  WHEN TDS-RESULTS-COMPLETE
    PERFORM FREE-ALL-STORAGE
  WHEN TDS-CONNECTION-TERMINATED
    PERFORM FREE-ALL-STORAGE
  WHEN OTHER
    MOVE 'TDGETREQ' TO CALL-ERROR
    PERFORM DISPLAY-CALL-ERROR
END-EVALUATE.
GOBACK.

* -----
FREE-ALL-STORAGE.
* -----
CALL 'TDFREE' USING GWL-PROC, GWL-RC.
CALL 'TDTERM' USING GWL-INIT-HANDLE, GWL-RC.
```

Usage

An application calls TDFREE 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.

Typically, a transaction calls TDFREE either at the end of a transaction, or after TDRESULT returns TDS-CONNECTION-TERMINATED or TDS-CONNECTION-FAILED.

Under IMS TM and MVS

The transaction must call TDTERM to free the *IHANDLE*.

The last call in an IMS TM program, after it has processed all requests, must be TDTERM, which frees all resources, including the *IHANDLE*, in preparation for program termination. Sybase strongly recommends ending all programs with a TDTERM call.

See also

Related functions

- TDACCEPT on page 70
- TDGETREQ on page 98
- TDINIT on page 142
- TDRESULT on page 167
- TDTERM on page 232

TDGETREQ

Description

Accepts the next request in a long-running transaction.

Syntax

COPY SYGWCOB.

```
01 TDPROC      PIC S9(9) USAGE COMP SYNC.
01 RETCODE    PIC S9(9) USAGE COMP SYNC.
01 WAIT-OPTION PIC S9(9) USAGE COMP SYNC.
01 REQUEST-TYPE PIC S9(9) USAGE COMP SYNC.
01 TRAN-NAME  PIC X(30).
```

```
CALL 'TDGETREQ'
USING 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 this 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 100.

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: Sybase recommends 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. This will be to 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 <i>TDPROC</i> 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 TDSNDDON and TDGETREQ in a Gateway-Library long-running transaction using the IMS TM explicit API. This example is taken from the sample program in Appendix E, "Sample RPC Application for IMS TM (Explicit)."

```

*-----
SEND ROWS TO CLIENT, MOVE ZEROES TO CTR-ROWS.
*-----

IF PARM-NR-ROWS = ZEROES THEN
MOVE 'Y' TO ALL-DONE-SW
ELSE
PERFORM SEND-ROWS
UNTIL ALL-DONE OR CTR-ROWS >= PARM-NR-ROWS.
IF SEND-DONE-OK
MOVE TDS-DONE-COUNT TO WRK-DONE-STATUS
ELSE
MOVE TDS-DONE-ERROR TO WRK-DONE-STATUS
MOVE ZERO TO CTR-ROWS
END-IF.

```

```

SEND-DONE.
    IF PARM-NR-ROWS = ZEROES THEN
        MOVE TDS-ENDRPC TO GWL-SEND-DONE
    ELSE
        MOVE TDS-ENDREPLY TO GWL-SEND-DONE.
*-----
    ISSUE SEND DONE TO CLIENT
*-----
    CALL 'TDSNDDON' USING GWL-PROC, GWL-RC,
                        WRK-DONE-STATUS,
                        CTR-ROWS,
                        TDS-ZERO,
                        GWL-SEND-DONE.

[check return code]

    IF PARM-NR-ROWS = ZEROES THEN
        PERFORM FREE-ALL-STORAGE
        GOBACK.
*-----
    GET NEXT CLIENT REQUEST
*-----
    MOVE TDS-TRUE TO GWL-WAIT-OPTION.
    MOVE ZEROES TO GWL-REQ-TYPE.
    MOVE SPACES TO GWL-RPC-NAME.
    CALL 'TDGETREQ' USING GWL-PROC, GWL-RC, GWL-WAIT-OPTION
                        GWL-REQ-TYPE, GWL-RPC-NAME.

[check return code]

    PERFORM FREE-ALL-STORAGE.
    GOBACK.

```

Example 2

The following code fragment illustrates the use of TDSNDDON and TDGETREQ in a Gateway-Library transaction using the IMS TM implicit API. This example is taken from the sample program in Appendix D, “Sample RPC Application for IMS TM (Implicit).”

```

*-----
SEND-ROWS
*-----
    PERFORM FETCH-AND-SEND-ROWS
        UNTIL ALL-DONE.
    FINISH-REPLY.

```

TDGETREQ

```
CALL 'TDSNDDON' USING GWL-PROC, GWL-RC,  
                    WRK-DONE-STATUS, CTR-ROWS,  
                    TDS-ZERO, TDS-ENDRPC.
```

[check return code]

```
* -----  
* Get next client request  
* -----  
MOVE TDS-TRUE TO GWL-WAIT-OPTION.  
MOVE ZEROES TO GWL-REQ-TYPE.  
MOVE SPACES TO GWL-RPC-NAME.  
CALL 'TDGETREQ' USING GWL-PROC, GWL-RC, GWL-WAIT-OPTION,  
                    GWL-REQ-TYPE, GWL-RPC-NAME.  
EVALUATE GWL-RC  
    WHEN ZEROES  
        GO TO READ-IN-USER-PARM  
    WHEN TDS-RESULTS-COMPLETE  
        PERFORM FREE-ALL-STORAGE  
    WHEN TDS-CONNECTION-TERMINATED  
        PERFORM FREE-ALL-STORAGE  
    WHEN OTHER  
        MOVE 'TDGETREQ' TO CALL-ERROR  
        PERFORM DISPLAY-CALL-ERROR  
END-EVALUATE.
```

Usage

Note IMS TM Users: Transactions running under the IMS TM implicit API do not support true long-running transactions. See “For IMS TM users” on page 103 in this section for IMS TM-specific information.

- Use TDGETREQ 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.
 - In a long-running transaction: To keep the connection open after TDSNDDON returns results for the previous client request, 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. When no request is present, the TDGETREQ action depends on the value of *WAIT-OPTION*:
 - When *WAIT-OPTION* is TDS-FALSE, TDGETREQ returns TDS-CONNECTION-TERMINATED.
 - When *WAIT-OPTION* is TDS-TRUE, TDGETREQ waits for another request; if the transaction stops, TDGETREQ returns TDS-CONNECTION-TERMINATED.

For IMS TM users

- Using the implicit API:

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.

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

- Using the explicit API:

Programs using the explicit API use the same Gateway-Library functions and parameters as CICS programs. Comments in this section apply to explicit IMS TM transactions and CICS transactions.

See also

Related functions

- TDSNDDON on page 206

Related topics

- “Communication states” on page 19
- “The login packet” on page 53
- “Long-running transactions” on page 54

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

COPY SYGWCOB.

```
01 TDPROC      PIC S9(9) USAGE COMP SYNC.
01 RETCODE     PIC S9(9) USAGE COMP SYNC.
01 OBJECT-TYPE PIC S9(9) USAGE COMP SYNC.
01 OBJECT-NUMBER PIC S9(9) USAGE COMP SYNC.
01 STRIP-SOSI  PIC S9(9) USAGE COMP SYNC.
```

```
CALL 'TDSETSOI' USING 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 105.

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.

Return value	Meaning
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 <i>TDPROC</i> argument.

Examples

The following code fragment uses TDGETSOI to replace SO/SI codes with blanks before retrieving parameters and again before returning data to the client. This example is not included on the Open ServerConnect API tape, but is available to Japanese customers on the Japanese Conversion Module tape.

```

*****
PROCEDURE DIVISION.
*****
    CALL 'TDINIT'    USING DFHEIBLK, GWL-RC, GWL-INIT-HANDLE.
*
    CALL 'TDACCEPT' USING GWL-PROC, GWL-RC, GWL-INIT-HANDLE,
                        SNA-CONNECTION-NAME,
                        SNA-SUBC.
*
    CALL 'TDRESULT' USING GWL-PROC, GWL-RC.
*
*   get the information of sosi
*
    MOVE TDS-OBJECT-PARM TO PRM-01-OBJ-TYPE.
    MOVE PRM-01-ID       TO PRM-01-OBJ-ID.
    CALL 'TDGETSOI' USING GWL-PROC, GWL-RC,
                        PRM-01-OBJ-TYPE,
                        PRM-01-OBJ-ID,
                        PRM-01-STRIP-SOSI.
*
    IF PRM-01-STRIP = TDS-STRIP-SOSI
    THEN
*
*   specify the embedded blanks to the parameter
*
    MOVE TDS-BLANK-SOSI TO PRM-01-STRIP-SOSI
    CALL 'TDSETSOI' USING GWL-PROC, GWL-RC,
                        PRM-01-OBJ-TYPE,
                        PRM-01-OBJ-ID,

```



```

                                PRM-01-STRIP-SOSI
*
END-IF
*
MOVE TDSCHAR                TO PRM-01-HOST-TYPE.
*
MOVE LENGTH OF PRM-01-DATA TO PRM-01-MAX-LEN.
*
CALL 'TDRCVPRM' USING GWL-PROC, GWL-RC,
                    PRM-01-ID,
                    PRM-01-AREA,
                    PRM-01-HOST-TYPE,
                    PRM-01-MAX-LEN,
                    PRM-01-ACT-LEN.
*
CALL 'TDESCRIB' USING GWL-PROC, GWL-RC,
                    COL-01-NUM,
                    COL-01-HOST-TYPE,
                    COL-01-HOST-LEN,
                    COL-01-AREA,
                    COL-01-NULL-INDICATOR,
                    TDS-FALSE,
                    COL-01-CLIENT-TYPE,
                    COL-01-CLIENT-LEN,
                    COL-01-NAME,
                    COL-01-NAME-LEN.
*
*
*   get the information of sosi
*
MOVE TDS-OBJECT-COL TO COL-01-OBJ-TYPE.
MOVE COL-01-NUM     TO COL-01-OBJ-ID.
CALL 'TDGETSOI' USING GWL-PROC, GWL-RC,
                    COL-01-OBJ-TYPE,
                    COL-01-OBJ-ID,
                    COL-01-STRIP-SOSI.
*
IF COL-01-STRIP-SOSI = TDS-STRIP-SOSI
THEN
*
*   specify the embedded blanks to the column
*
MOVE TDS-BLANK-SOSI TO COL-01-STRIP-SOSI
CALL 'TDSETSOI' USING GWL-PROC, GWL-RC,
                    COL-01-OBJ-TYPE,
                    COL-01-OBJ-ID,
                    COL-01-STRIP-SOSI
END-IF
```

*
*

PERFORM FETCH-AND-SEND-ROWS UNTIL ALL-DONE.

- Usage
- 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 58.

See also

Related functions

- TDSETSOI on page 196

Related topics

- “Character sets” on page 17
- “Processing Japanese client requests” on page 58

TDGETUSR

Description Gets user login information from the client.

Syntax

```
COPY SYGWCOB.
01 TDPROC          PIC S9(9) USAGE COMP SYNC.
01 RETCODE         PIC S9(9) USAGE COMP SYNC.
01 ACCESS-CODE    PIC X(32).
01 USER-ID        PIC X(32).
01 PASSWORD       PIC X(32).
01 SERVER-NAME    PIC X(32).
01 CLIENT-CHARSET PIC X(32).
01 NATIONAL-LANGUAGE PIC X(32).
01 SERVER-CHARSET PIC X(32).01 SERVER-DBCS    PIC X(32).
01 APPNAME-ID     PIC X(32).CALL 'TDGETUSR' USING 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 110.

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 when logging 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.

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.

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 <i>TDPROC</i> 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. This example is taken from the sample program, SYCCSAR2, in Appendix B, “Sample RPC Application for CICS.”

```
*   Accept client request

CALL 'TDACCEPT' USING GWL-PROC, GWL-RC, GWL-INIT-HANDLE,
                    SNA-CONNECTION-NAME, SNA-SUBC.

*   TDRESULT to make sure we were started via RPC request

CALL 'TDRESULT' USING GWL-PROC, GWL-RC.

IF GWL-RC NOT = TDS-PARM-PRESENT THEN
    PERFORM TDRESULT-ERROR
    GO TO END-PROGRAM
END-IF.

*   Verify user login information

MOVE 'TOP SECRET' TO GU-ACCESS-CODE.

CALL 'TDGETUSR'  USING 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 NOT = TDS-OK THEN
    PERFORM TDGETUSR-ERROR
    GO TO END-PROGRAM
END-IF.
```

Usage

- 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 also 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 functions

- TDACCEPT on page 70

Related topics

- “Character sets” on page 17
- “The login packet” on page 53
- “Customization” on page 35

TDINFACT

Description

Retrieves information about Gateway-Library accounting.

Syntax

COPY SYGWCOB.

```

01 IHANDLE          PIC S9(9) USAGE COMP SYNC.
01 RETCODE          PIC S9(9) USAGE COMP SYNC.
01 ACCOUNTING-FLAG PIC S9(4) USAGE COMP SYNC.
01 ACCOUNTING-FILENAME PIC X(8) VALUE IS SPACES.
01 MAXNUM-ACCT-RECORDS PIC 9(9) USAGE COMP SYNC.

```

CALL 'TDINFACT' USING 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 114.

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 <i>IHANDLE</i> 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.
*-----
      MOVE LENGTH OF GWL-INFACT-STATUS  TO WRKLEN1.
      MOVE LENGTH OF CN-INFACT-STATUS   TO WRKLEN2.
      ADD +1                             TO CTR-COLUMN.
      MOVE 'TDESCRIB'                   TO MSG-SRVLIB-FUNC.
      CALL 'TDESCRIB' USING GWL-PROC, GWL-RC, CTR-COLUMN, TDSINT4,
                                WRKLEN1, GWL-INFACT-STATUS, TDS-ZERO,
                                TDS-FALSE, TDSINT4, WRKLEN1,
                                CN-INFACT-STATUS, WRKLEN2.

      IF GWL-RC NOT = TDS-OK THEN
          MOVE 'N' TO SEND-DONE-SW
          GO TO TDINFACT-EXIT
      END-IF.

      MOVE LENGTH OF GWL-INFACT-FILENAME TO WRKLEN1.
      MOVE LENGTH OF CN-INFACT-FILENAME  TO WRKLEN2.
      ADD +1                             TO CTR-COLUMN.
      CALL 'TDESCRIB' USING GWL-PROC, GWL-RC, CTR-COLUMN, TDSCHAR,
                                WRKLEN1, GWL-INFACT-FILENAME,
                                TDS-ZERO, TDS-FALSE, TDSCHAR, WRKLEN1,
                                CN-INFACT-FILENAME, WRKLEN2.

      IF GWL-RC NOT = TDS-OK THEN
          MOVE 'N' TO SEND-DONE-SW
          GO TO TDINFACT-EXIT
      END-IF.

      MOVE LENGTH OF GWL-INFACT-RECORDS  TO WRKLEN1.
      MOVE LENGTH OF CN-INFACT-RECORDS   TO WRKLEN2.
      ADD +1                             TO CTR-COLUMN.
      CALL 'TDESCRIB' USING GWL-PROC, GWL-RC, CTR-COLUMN, TDSINT4,
                                WRKLEN1, GWL-INFACT-RECORDS, TDS-ZERO,
                                TDS-FALSE, TDSINT4, WRKLEN1,

```



```

                                CN-INFACT-RECORDS, WRKLEN2.
IF GWL-RC NOT = TDS-OK THEN
    MOVE 'N' TO SEND-DONE-SW
    GO TO TDINFACT-EXIT
END-IF.
CALL 'TDINFACT' USING GWL-INIT-HANDLE, GWL-RC,
                    GWL-INFACT-STATUS,
                    GWL-INFACT-FILENAME,
                    GWL-INFACT-RECORDS.
IF GWL-RC NOT = TDS-OK THEN
    MOVE 'N'          TO SEND-DONE-SW
    MOVE 'TDINFACT' TO MSG-SRVLIB-FUNC
    GO TO TDINFACT-EXIT
END-IF.
CALL 'TDSNDROW' USING GWL-PROC, GWL-RC.
*-----
TDINFACT-EXIT.
*-----
EXIT.

```

Usage

- 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 70
- TDFREE on page 95
- TDSETACT on page 170

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

COPY SYGWCOB.

```
01 TDPROC          PIC S9(9)USAGE COMP SYNC.
01 RETCODE         PIC S9(9)USAGE COMP SYNC.
01 OBJECT-TYPE     PIC S9(9)USAGE COMP SYNC.
01 OBJECT-NUMBER   PIC S9(9)USAGE COMP SYNC.
01 BCD-LENGTH      PIC S9(9)USAGE COMP SYNC.
01 BCD-NUMBER-DECIMAL-PLACES PIC S9(9)USAGE COMP SYNC.
```

CALL 'TDINFBCD' USING 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 117.

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/decimal, this is a 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.

Return value	Meaning
TDS-INVALID-TDPROC (-18)	Error in specifying a value for the <i>TDPROC</i> argument.

Examples The following code fragment shows two methods of converting datatypes. One uses TDESCRIB to convert data from the DB2 datatype DECIMAL (TDSDECIMAL) to TDSFLT8. The other uses TDCONVRT to convert data from the DB2 datatype DECIMAL (TDSDECIMAL) to the DB-Library datatype DBMONEY (TDSMONEY). 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, SYCCSAR2, in Appendix B, "Sample RPC Application for CICS."

* Here we let TDESCRIB convert from TDSDECIMAL to TDSFLT8.

```
CALL 'SYGETAD' USING DB-DESCRIBE-HV-PTR, EMPLOYEE-JC.
CALL 'SYGETAD' USING DB-COLUMN-NAME-HV-PTR, CN-JC.
MOVE LENGTH OF EMPLOYEE-JC TO WRKLEN1.
MOVE LENGTH OF CN-JC      TO WRKLEN2.
MOVE TDSDECIMAL          TO DB-HOST-TYPE.
MOVE TDSFLT8             TO DB-CLIENT-TYPE.
PERFORM DESCRIBE-COLUMN.
```

* We must inform the Server Library how many decimal places
* are in the EMPLOYEE-JC column.

```
CALL 'TDSETBCD' USING GWL-PROC, GWL-RC, TDS-OBJECT-COL,
                   CTR-COLUMN, TDS-DEFAULT-LENGTH,
                   GWL-SETBCD-SCALE.
```

* Demonstrate getting decimal column information.

```
CALL 'TDINFBCD' USING 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.

```
CALL 'SYGETAD' USING DB-DESCRIBE-HV-PTR, WRK-EMPLOYEE-SAL.
CALL 'SYGETAD' USING DB-COLUMN-NAME-HV-PTR, CN-SAL.
MOVE LENGTH OF WRK-EMPLOYEE-SAL TO WRKLEN1.
MOVE LENGTH OF CN-SAL          TO WRKLEN2.
```

```

MOVE TDSMONEY                TO DB-HOST-TYPE.
MOVE TDSMONEY                TO DB-CLIENT-TYPE.
PERFORM DESCRIBE-COLUMN.

PERFORM FETCH-AND-SEND-ROWS
      UNTIL ALL-DONE.
*-----
  FETCH-AND-SEND-ROWS.
*-----
  EXEC SQL FETCH ECURSOR INTO :EMPLOYEE-FIELDS END-EXEC.

  IF SQLCODE = 0 THEN

*       Convert from DB2 decimal (TDSDECIMAL) to dblib MONEY.

      MOVE LENGTH OF EMPLOYEE-SAL      TO WRKLEN1
      MOVE LENGTH OF WRK-EMPLOYEE-SAL TO WRKLEN2

      CALL 'TDCONVRT' USING GWL-PROC, GWL-RC,
                          GWL-CONVRT-SCALE, TDSDECIMAL,
                          WRKLEN1, EMPLOYEE-SAL, TDSMONEY,
                          WRKLEN2, WRK-EMPLOYEE-SAL

*       send a row to the client

      CALL 'TDSNDROW' USING GWL-PROC, GWL-RC
      ADD 1 TO PARM-RETURN-ROWS

      IF GWL-RC = TDS-CANCEL-RECEIVED THEN
        MOVE 'Y' TO ALL-DONE-SW
      END-IF

      ELSE IF SQLCODE = +100 THEN
        MOVE 'Y' TO ALL-DONE-SW

      ELSE
        MOVE 'Y' TO ALL-DONE-SW
        PERFORM FETCH-ERROR
      END-IF.

*-----
  GET-PARM-INFO.
*-----
  CALL 'TDINFPRM' USING 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.

```
*-----
DESCRIBE-COLUMN.
*-----

SET ADDRESS OF LK-DESCRIBE-HV      TO DB-DESCRIBE-HV-PTR.
SET ADDRESS OF LK-COLUMN-NAME-HV  TO DB-COLUMN-NAME-HV-PTR.
ADD 1                               TO CTR-COLUMN.

CALL 'TDESCRIB' USING GWL-PROC, GWL-RC, CTR-COLUMN,
                    DB-HOST-TYPE, WRKLEN1, LK-DESCRIBE-HV,
                    DB-NULL-INDICATOR, TDS-FALSE,
                    DB-CLIENT-TYPE, WRKLEN1,
                    LK-COLUMN-NAME-HV, WRKLEN2.
```

Usage

- Packed decimal data is supported in COBOL, but not in DB-Library or Client-Library.
- Numeric and Sybase decimal are used as Client-Library decimal datatypes.

Note Although the name of this function implies BCD data, in COBOL 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 87
- TDSETBCD on page 174

TDINFLOG

Description Determines what types of mainframe server tracing have been set.

Syntax COPY SYGWCOB.

```
01 IHANDLE          PIC S9(9) USAGE COMP SYNC.
01 RETCODE          PIC S9(9) USAGE COMP SYNC.
01 GLOBAL-TRACE-FLAG PIC S9(9) USAGE COMP SYNC.
01 API-TRACE-FLAG   PIC S9(9) USAGE COMP SYNC.
01 TDS-HEADER-TRACE-FLAG PIC S9(9) USAGE COMP SYNC.
01 TDS-DATA-TRACE-FLAG PIC S9(9) USAGE COMP SYNC.
01 TRACE-ID         PIC S9(9) USAGE COMP SYNC.
01 TRACE-FILENAME   PIC X(8).
01 MAXNUM-TRACE-RECORDS PIC S9(9) USAGE COMP SYNC.
```

CALL 'TDINFLOG' USING 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-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 123.

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) The 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) The 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) The 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 on page 123.

Table 3-14: TDINFLOG return values

Return value	Meaning
TDS-OK (0)	Function completed successfully.
TDS-INVALID-IHANDLE (-19)	Invalid <i>IHANDLE</i> specification. Error in specifying a value for the <i>IHANDLE</i> 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 shows how to use TDINFLOG at the beginning of a program to determine which types of tracing are currently enabled.

This example is taken from the sample program in Appendix C, “Sample Language Application for CICS.”

```

*   Establish gateway environment
      CALL 'TDINIT' USING DFHEIBLK, GWL-RC, GWL-INIT-HANDLE.
*   Turn on local tracing if not on globally or locally
      CALL 'TDINFLOG' USING 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 NOT = TDS-TRACE-ALL-RPCS
      AND GWL-INFLOG-GLOBAL NOT = TDS-TRACE-SPECIFIC-RPCS THEN
          MOVE 1 TO TRACING-SET-SW
          PERFORM LOCAL-TRACING
      END-IF.
*   Accept client request
      CALL 'TDACCEPT' USING GWL-PROC, GWL-RC, GWL-INIT-HANDLE,
                          SNA-CONNECTION-NAME,
                          SNA-SUBC.

* -----
LOCAL-TRACING.
* -----
      CALL 'TDSETSPT' USING GWL-INIT-HANDLE, GWL-RC,
                          TRACING-SET-SW,
                          GWL-SETSPT-TRACE-LEVEL,
                          GWL-SETSPT-RPC-NAME,
                          GWL-SETSPT-RPC-NAME-L.

```

Usage

- You use this function 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. Traces Gateway-Library calls.
Under CICS: Uses the CICS auxiliary trace facility.
Under IMS TM: Uses the IMS TM system log.
Under MVS: Uses a sequential file.
 - TDS header tracing. Keeps track of the 8-byte TDS headers being sent to and from the mainframe server.
 - TDS data tracing. 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 trace facility, instructions for using it, and the layout of the CICS trace log.

See also

Related functions

- TDACCEPT on page 70
- TDFREE on page 95
- TDINFSPT on page 136
- TDSETLOG on page 183
- TDSETSPT on page 200
- TDWRTLOG on page 238

TDINFPGM

Description	Retrieves information about the current client request.
Syntax	<pre> COPY SYGWCORB. 01 TDPROC PIC S9(9) USAGE COMP SYNC. 01 RETCODE PIC S9(9) USAGE COMP SYNC. 01 TDS-VERSION PIC S9(9) USAGE COMP SYNC. 01 LONGVAR-TRUNC-FLAG PIC S9(9) USAGE COMP SYNC. 01 ROW-LIMIT PIC S9(9) USAGE COMP SYNC. 01 REMOTE-TRACE-FLAG PIC S9(9) USAGE COMP SYNC. 01 USER-CORRELATOR PIC S9(9) USAGE COMP SYNC. 01 DB2GW-OPTIONS PIC S9(9) USAGE COMP SYNC. 01 DB2GW-PID PIC X(8). 01 REQUEST-TYPE PIC S9(9) USAGE COMP SYNC. CALL 'TDINFPGM' USING TDPROC,RETCODE, TDS-VERSION, LONGVAR-TRUNC-FLAG,ROW-LIMIT, REMOTE-TRACE-FLAG, USER-CORRELATOR,DB2GW-OPTIONS, DB2GW-PID, REQUEST-TYPE. </pre>
Parameters	<p><i>TDPROC</i></p> <p>(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The <i>TDPROC</i> handle corresponds to the connection and command handles in Open Client Client-Library.</p> <p><i>RETCODE</i></p> <p>(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-16 on page 127.</p> <p><i>TDS-VERSION</i></p> <p>(O) Variable where the version of TDS being used is returned. The version value can be any of the following listed in Table 3-15 on page 126.</p>

Table 3-15: TDS-VERSION values

TDS-VERSION-20	PIC S9(9)	COMP VALUE 512
TDS-VERSION-34	PIC S9(9)	COMP VALUE 832
TDS-VERSION-40	PIC S9(9)	COMP VALUE 1024
TDS-VERSION-42	PIC S9(9)	COMP VALUE 1056
TDS-VERSION-46	PIC S9(9)	COMP VALUE 1120
TDS-VERSION-48	PIC S9(9)	COMP VALUE 1152
TDS-VERSION-49	PIC S9(9)	COMP VALUE 1168
TDS-VERSION-50	PIC S9(9)	COMP VALUE 1280
TDS-VERSION-51	PIC S9(9)	COMP VALUE 1296

This value must be the same as the version level specified at the client.

LONGVAR-TRUNC-FLAG

(O) Variable where the truncation indicator for TDSLONGVARCHAR fields is returned. It indicates what happens when 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 may 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) Variable where the type of client request is indicated. One of the following values is returned:

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 (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 <i>TDPROC</i> 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.”

```
*      Establish gateway environment.
      CALL 'TDINIT' USING DFHEIBLK, GWL-RC, GWL-INIT-HANDLE.
```

* Accept client request

```
CALL 'TDACCEPT' USING 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' USING 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 NOT = TDS-START-SQL
    MOVE MSG-NOT-LANG          TO MSG-TEXT
    MOVE LENGTH OF MSG-NOT-LANG TO MSG-TEXT-L
    PERFORM SEND-ERROR-MESSAGE
    GO TO END-PROGRAM
END-IF.
```

Usage

- A server application uses TDINFPGM 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, cursor 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.
- TDINFPGM looks at both the TRS and mainframe customization settings to determine whether truncation will occur, according to Table 3-17.

Table 3-17: TDSLONGVARCHAR truncation rule

When TRS truncation flag	When TDCUSTOM truncation flag	TDSLONGVARCHAR fields
ON	ON	Truncated

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

- The argument *USER-CORRELATOR* is available for sending site-specific information.

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

COPY SYGWCORB.

```
01 TDPROC          PIC S9(9) USAGE COMP SYNC.
01 RETCODE         PIC S9(9) USAGE COMP SYNC.
01 PARM-ID         PIC S9(9) USAGE COMP SYNC.
01 DATATYPE        PIC S9(9) USAGE COMP SYNC.
01 ACTUAL-DATA-LENGTH PIC S9(9) USAGE COMP SYNC.
01 MAX-DATA-LENGTH PIC S9(9) USAGE COMP SYNC.
01 PARM-STATUS     PIC S9(9) USAGE COMP SYNC.
01 PARM-NAME       PIC X(30).
01 PARM-NAME-LENGTH PIC S9(9) USAGE COMP SYNC.
01 USER-DATATYPE   PIC S9(9) USAGE COMP SYNC.
```

```
CALL 'TDINFPRM' USING 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 131.

PARAM-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.

PARAM-STATUS

(O) Variable where the parameter’s status is returned. This argument indicates whether the named parameter is a return parameter. It returns one of the following values, depending on the TDS version you are using.

- *For TDS 4.6:*

TDS-INPUT-VALUE (0)	Parameter is not a return parameter.
TDS-RETURN-VALUE (1)	Parameter is a return parameter.

- *For 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.

PARAM-NAME

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

PARAM-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.

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.
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 <i>TDPROC</i> argument.
TDS-NO-PARM-PRESENT (103)	No incoming parameters present. <i>TDRCVPRM</i> 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 <i>RECEIVE</i> state and could not send. The operation failed.

Examples The following code fragment illustrates a typical use of *TDINFPRM*. The transaction: calls *TDNUMPRM* to determine how many parameters to retrieve; calls *TDLOCPRM* to ascertain the number of the parameter whose information it wants; calls *TDINFPRM* for a description of the parameter; calls *TDRCVPRM* to retrieve the parameter data. This example is taken from the sample program, *SYCCSAR2*, in Appendix B, “Sample RPC Application for CICS.”

```
*      Get number of parameters ... should be two

      CALL 'TDNUMPRM' USING GWL-PROC, GWL-NUMPRM-PARMS.

      IF GWL-NUMPRM-PARMS NOT = 2 THEN
          PERFORM TDNUMPRM-ERROR
          GO TO END-PROGRAM
      END-IF.

*      Get return parameter information

      MOVE 1 TO GWL-INFPRM-ID.
      PERFORM GET-PARM-INFO.

      (IF GWL-INFPRM-STATUS NOT = TDS-RETURN-VALUE AND
      IF GWL-INFPRM-STATUS NOT = TDS-RETURN-VALUE-NULLABLE) THEN
          PERFORM TDINFPRM-NOT-RETURN-PARM-ERROR
          GO TO END-PROGRAM
      END-IF.

      MOVE GWL-INFPRM-USER-DATA TO GWL-SETPRM-USER-DATA.
      MOVE GWL-INFPRM-ID          TO GWL-SETPRM-ID.
      MOVE GWL-INFPRM-DATA-L     TO GWL-SETPRM-DATA-L.
      MOVE GWL-INFPRM-TYPE      TO GWL-SETPRM-TYPE.

*      Get department id parameter number from known name

      MOVE '@parm2' TO GWL-INFPRM-NAME.
      MOVE 6         TO GWL-INFPRM-NAME-L.

      CALL 'TDLOCPRM' USING GWL-PROC, GWL-INFPRM-ID,
                          GWL-INFPRM-NAME, GWL-INFPRM-NAME-L.

*      Get department parameter information

      PERFORM GET-PARM-INFO.

      IF GWL-INFPRM-TYPE NOT = TDSVARYCHAR THEN
          PERFORM TDINFPRM-NOT-CHAR-PARM-ERROR
          GO TO END-PROGRAM
      END-IF.

*      Get department parameter data

      CALL 'TDRCVPRM' USING GWL-PROC, GWL-RC, GWL-INFPRM-ID,
                          PARM-DEPT, GWL-INFPRM-TYPE,
```

```

                                GWL-INFPRM-MAX-DATA-L,
                                GWL-RCVPRM-DATA-L.
*-----
GET-PARM-INFO.
*-----
CALL 'TDINFPRM' USING 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.

```

Usage

- A server application uses this function to retrieve datatype 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 which 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.

See also*Related functions*

- TDACCEPT on page 70
- TDNUMPRM on page 152
- TDRCVPRM on page 154
- TDSETPRM on page 188

TDINFRPC

Description	Returns information about the current client request.								
Syntax	<pre>COPY SYGWCOB. 01 TDPROC PIC S9(9)USAGE COMP SYNC. 01 RETCODE PIC S9(9)USAGE COMP SYNC. 01 REQUEST-TYPE PIC S9(9)USAGE COMP SYNC. 01 RPC-NAME PIC X(30). 01 COMM-STATE PIC S9(9)USAGE COMP SYNC. CALL 'TDINFRPC' USING TDPROC, RETCODE, REQUEST-TYPE, RPC-NAME, COMM-STATE.</pre>								
Parameters	<p><i>TDPROC</i> (I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The <i>TDPROC</i> handle corresponds to the connection and command handles in Open Client Client-Library.</p> <p><i>RETCODE</i> (O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-19 on page 135.</p> <p><i>REQUEST-TYPE</i> (O) Type of request being accepted. Returns one of the following values:</p> <table><tr><td>TDS-LANGUAGE-EVENT (1)</td><td>Current request is a language request.</td></tr><tr><td>TDS-RPC -EVENT (3)</td><td>Current request is an RPC.</td></tr><tr><td>TDS-DYNAMIC-EVENT (4)</td><td>Current request is a dynamic SQL request.</td></tr><tr><td>TDS-CURSOR-EVENT (5)</td><td>Current request is a cursor request.</td></tr></table> <p>TDINFPGM and TDGETREQ also return this information.</p> <hr/> <p>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.</p> <hr/> <p><i>RPC-NAME</i> (O) Variable where the name of the current client RPC is returned. If the client request is not an RPC, this field contains blanks.</p>	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.
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.								

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 communication states.

Return value The *RETCODE* argument can contain any of the return values listed in Table 3-19 on page 135.

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

- 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. TDSTATUS also returns 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.

See also

Related functions

- TDGETREQ on page 98
- TDINFPGM on page 125
- TDSNDDON on page 206
- TDSTATUS on page 227

Related topics

- “Communication states” on page 19

TDINFSPT

Description

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

Syntax

COPY SYGWCOB.

```

01 IHANDLE          PIC S9(9) USAGE COMP SYNC.
01 RETCODE          PIC S9(9) USAGE COMP SYNC.
01 TRACE-STATUS    PIC S9(9) USAGE COMP SYNC.
01 TRACE-OPTION    PIC S9(9) USAGE COMP SYNC.
01 TRANSACTION-ID  PIC X(n).
01 TRANSACTION-ID-LENGTH PIC S9(9) USAGE COMP SYNC.

CALL 'TDINFSPT' USING 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 138.

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 on page 138.

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 <i>IHANDLE</i> 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.
*-----
*   Determine whether global tracing is on.
   CALL 'TDINFLOG' USING GWL-INIT-HANDLE, GWL-RC,
                           GWL-INFLOG-GLOBAL, GWL-INFLOG-API,
                           GWL-INFLOG-HEADER, GWL-INFLOG-DATA,
                           GWL-INFLOG-TRACEID,
                           GWL-INFLOG-FILENAME,
*   If specific tracing is on, see if it's on for this
*   transaction and turn on the tracing flag.
   IF GWL-INFLOG-GLOBAL NOT = TDS-TRACE-SPECIFIC-RPCS
       GO TO GET-TRACE-STATUS-EXIT
   END-IF.
   MOVE LENGTH OF WRK-RPC TO WRKLEN1.
   CALL 'TDINFSPT' USING GWL-INIT-HANDLE, GWL-RC,

```



```

                                GWL-INFSPPT-STATUS, GWL-INFSPPT-OPTIONS,
                                WRK-RPC, WRKLEN1.
IF GWL-RC NOT = TDS-OK AND
  GWL-RC NOT = TDS-ENTRY-NOT-FOUND THEN
  MOVE 'N' TO SEND-DONE-SW
  MOVE 'TDINFSPPT' TO MSG-SRVLIB-FUNC
  GO TO GET-TRACE-STATUS-EXIT
END-IF.
IF GWL-INFSPPT-STATUS = TDS-TRUE THEN
  MOVE 'Y' TO TRACING-SW
END-IF.

```

Usage

- TDINFSPPT 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 200.
- To learn how to get a list of all transactions for which tracing is currently enabled, see TDLSTSPT on page 149.
- TDINFSPPT 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 121
- TDLSTSPT on page 149
- TDSETLOG on page 183

- TDSETSPT on page 200
- TDWRTLOG on page 238

Related documents

- Mainframe Connect Server Option *Installation and Administration Guide*

TDINFUDT

Description	Retrieves information about the client-defined datatype for a column.
Syntax	<pre>COPY SYGWCOB. 01 TDPROC PIC S9(9) USAGE COMP SYNC. 01 RETCODE PIC S9(9) USAGE COMP SYNC. 01 COLUMN-NUMBER PIC S9(9) USAGE COMP SYNC. 01 USER-DATATYPE PIC S9(9) USAGE COMP SYNC. CALL 'TDINFUDT' USING TDPROC, RETCODE, COLUMN-NUMBER, USER-DATATYPE.</pre>
Parameters	<p>TDPROC (I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The <i>TDPROC</i> handle corresponds to the connection and command handles in Open Client Client-Library.</p> <p>RETCODE (O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-21.</p> <p>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.</p> <p>USER-DATATYPE (O) Variable where the user-defined datatype is returned. This can be any datatype assigned to the column by a client.</p>
Return value	The <i>RETCODE</i> argument can contain any of the return values listed in Table 3-21 on page 140.

Table 3-21: TDINFUDT return values

Return value	Meaning
TDS-OK (0)	Function completed successfully.

Return value	Meaning
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 <i>TDPROC</i> argument.

Examples

The following code fragment illustrates a typical use of TDINFUDT. This example is taken from the sample program, SYCCSAR2, in Appendix B, “Sample RPC Application for CICS.”

```

*   Here we let TDESCRIB convert from DB2 varchar (TDSVARYCHAR)
*   to DECHAR.

CALL 'SYGETAD' USING DB-DESCRIBE-HV-PTR, EMPLOYEE-ED.
CALL 'SYGETAD' USING DB-COLUMN-NAME-HV-PTR, CN-ED.
MOVE LENGTH OF EMPLOYEE-ED TO WRKLEN1.
MOVE LENGTH OF CN-ED      TO WRKLEN2.
MOVE TDSINT2              TO DB-HOST-TYPE.
MOVE TDSINT2              TO DB-CLIENT-TYPE.
PERFORM DESCRIBE-COLUMN.

*   Get the user defined datatype of EMPLOYEE-ED column.

CALL 'TDINFUDT' USING GWL-PROC, GWL-RC, CTR-COLUMN,
                  GWL-INFUDT-USER-TYPE.

*   Set the user defined datatype of EMPLOYEE-ED column.

CALL 'TDSETUDT' USING GWL-PROC, GWL-RC, CTR-COLUMN,
                  GWL-INFUDT-USER-TYPE.

*-----
DESCRIBE-COLUMN.
*-----

SET ADDRESS OF LK-DESCRIBE-HV      TO DB-DESCRIBE-HV-PTR.
SET ADDRESS OF LK-COLUMN-NAME-HV TO DB-COLUMN-NAME-HV-PTR.
ADD 1                              TO CTR-COLUMN.

CALL 'TDESCRIB' USING GWL-PROC, GWL-RC, CTR-COLUMN,
                  DB-HOST-TYPE, WRKLEN1, LK-DESCRIBE-HV,
                  DB-NUL-INDICATOR, TDS-FALSE,
                  DB-CLIENT-TYPE, WRKLEN1,

```

LK-COLUMN-NAME-HV, WRKLEN2.

- Usage
- 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 129
- TDSETPRM on page 188
- TDSETUDT on page 204

TDINIT

Description

Initializes the TDS environment for a connection.

Syntax

COPY SYGWCOB.

01 RETCODE PIC S9(9) USAGE COMP SYNC.

01 IHANDLE PIC S9(9) USAGE COMP SYNC.

For CICS:ThinSpace CALL 'TDINIT' USING DFHEIBLK, RETCODE, IHANDLE.

For IMS TM: CALL 'TDINIT' USING IO-PCB, RETCODE, IHANDLE.

For native MVS: CALL 'TDINIT' USING DUMMY, GWL-RC, GWL-INIT-HANDLE.

Note MVS does not need to use anything.

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.
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 <i>IHANDLE</i> specification. Error in specifying a value for the <i>IHANDLE</i> 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, SYCCSAR2, in Appendix B, “Sample RPC Application for CICS.”

```
*   Establish gateway environment

      CALL 'TDINIT' USING DFHEIBLK, GWL-RC, GWL-INIT-HANDLE.

*   Accept client request

      CALL 'TDACCEPT' USING GWL-PROC, GWL-RC, GWL-INIT-HANDLE,
```

SNA-CONNECTION-NAME, SNA-SUBC.

```

*   TDRESULT to make sure we were started via RPC request

CALL 'TDRESULT' USING GWL-PROC, GWL-RC.

IF GWL-RC NOT = TDS-PARM-PRESENT THEN
    PERFORM TDRESULT-ERROR
    GO TO END-PROGRAM
END-IF.
* -----
* body of program
* -----
* -----
END-PROGRAM.
* -----

IF SEND-DONE-OK
    MOVE TDS-DONE-COUNT TO WRK-DONE-STATUS
ELSE
    MOVE TDS-DONE-ERROR TO WRK-DONE-STATUS
    MOVE ZERO           TO PARM-RETURN-ROWS
END-IF.

CALL 'TDSNDDON' USING GWL-PROC, GWL-RC, WRK-DONE-STATUS,
                    PARM-RETURN-ROWS, TDS-ZERO,
                    TDS-ENDRPC.

CALL 'TDFREE' USING GWL-PROC, GWL-RC.

EXEC CICS RETURN END-EXEC.

```

Example 2

The following code fragment shows the use of TDINIT, TDSETPT, and TDACCEPT at the beginning of a program that uses the implicit API under IMS TM. This example is taken from the sample program in Appendix D, “Sample RPC Application for IMS TM (Implicit).”

```

* -----
* establish gateway environment
* -----
CALL 'TDINIT' USING IO-PCB, GWL-RC, GWL-INIT-HANDLE.
.
. [check return code]
.
* -----
* set program type to MPP

```

```

* -----
CALL 'TDSETPT' USING  GWL-INIT-HANDLE, GWL-RC,
                    GWL-PROG-TYPE, GWL-SPA-PTR,
                    TDS-NULL, TDS- NULL.
.
. [check return code]
.
* -----
* accept client request
* -----
CALL 'TDACCEPT' USING GWL-PROC, GWL-RC, GWL-INIT-HANDLE,
                    SNA-CONNECTION-NAME, SNA-SUBC.
* -----
READ-IN-USER-PARM.
* -----

```

Usage

- 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.
- TDINIT 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: The EXEC Interface Block (EIB). You must code “DFHEIBLK” exactly as shown in the first call in the Syntax example.
 - Under IMS TM: The I/O Program Communications Block. You must code “IO-PCB” exactly as shown 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 58 for more information about using Gateway-Library with Japanese characters.

See also

Related functions

- TDACCEPT on page 70
- TDFREE on page 95
- TDGETUSR on page 108

Related topics

- “Character sets” on page 17
- “Processing Japanese client requests” on page 58
- “Customization” on page 35

TDLOCPRM

Description

Returns the ID number of a parameter when the parameter name is received.

Syntax

COPY SYGWCOB.

```
01 TDPROC      PIC S9(9) USAGE COMP SYNC.
01 PARM-ID     PIC S9(9) USAGE COMP SYNC.
01 PARM-NAME   PIC X(n).
01 PARM-NAME-LENGTH PIC S9(9) USAGE COMP SYNC.
```

```
CALL 'TDLOCPRM' USING TDPROC, PARM-ID, PARM-NAME,
    PARM-NAME-LENGTH.
```


Parameters	<p><i>TDPROC</i></p> <p>(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The <i>TDPROC</i> handle corresponds to the connection and command handles in Open Client Client-Library.</p> <p><i>PARAM-ID</i></p> <p>(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.</p> <p><i>PARAM-NAME</i></p> <p>(I) The name associated with the desired parameter. This name corresponds to the parameter name in the Open Client DB-Library dbrpcparam routine.</p> <p><i>PARAM-NAME-LENGTH</i></p> <p>(I) The actual length of the <i>PARAM-NAME</i>.</p>
Return value	This function has no <i>RETCODE</i> argument. It returns the parameter ID in the <i>PARAM-ID</i> argument, or a 0 if it finds no parameter with the specified name.
Examples	<p>The following code fragment illustrates a typical use of TDLOCPRM. The transaction calls TDNUMPRM to determine how many parameters to retrieve, calls TDLOCPRM to ascertain the number of the parameter with the information it wants, calls TDINFPRM for a description of the parameter, and calls TDRCVPRM to retrieve the parameter data.</p> <p>This example is taken from the sample program, SYCCSAR2, in Appendix B, "Sample RPC Application for CICS."</p> <pre> * Get number of parameters ... should be two CALL 'TDNUMPRM' USING GWL-PROC, GWL-NUMPRM-PARMS. IF GWL-NUMPRM-PARMS NOT = 2 THEN PERFORM TDNUMPRM-ERROR GO TO END-PROGRAM END-IF. * Get return parameter information MOVE 1 TO GWL-INFPRM-ID. PERFORM GET-PARM-INFO. (IF GWL-INFPRM-STATUS NOT = TDS-RETURN-VALUE AND IF GWL-INFPRM-STATUS NOT = TDS-RETURN-VALUE-NULLABLE) THEN PERFORM TDINFPRM-NOT-RETURN-PARM-ERROR </pre>

```
GO TO END-PROGRAM
END-IF.
```

```
MOVE GWL-INFPRM-USER-DATA TO GWL-SETPRM-USER-DATA.
MOVE GWL-INFPRM-ID        TO GWL-SETPRM-ID.
MOVE GWL-INFPRM-DATA-L    TO GWL-SETPRM-DATA-L.
MOVE GWL-INFPRM-TYPE      TO GWL-SETPRM-TYPE.
```

* Get department id parameter number from known name

```
MOVE '@parm2' TO GWL-INFPRM-NAME.
MOVE 6        TO GWL-INFPRM-NAME-L.
```

```
CALL 'TDLOCPRM' USING GWL-PROC, GWL-INFPRM-ID,
                    GWL-INFPRM-NAME, GWL-INFPRM-NAME-L.
```

* Get department parameter information

```
PERFORM GET-PARM-INFO.

IF GWL-INFPRM-TYPE NOT = TDSVARYCHAR THEN
    PERFORM TDINFPRM-NOT-CHAR-PARM-ERROR
    GO TO END-PROGRAM
END-IF.
```

* Get department parameter data

```
CALL 'TDRCVPRM' USING GWL-PROC, GWL-RC, GWL-INFPRM-ID,
                    PARM-DEPT, GWL-INFPRM-TYPE,
                    GWL-INFPRM-MAX-DATA-L,
                    GWL-RCVPRM-DATA-L.
```

*-----
GET-PARM-INFO.

*-----
CALL 'TDINFPRM' USING 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.

Usage

- 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 *PARAM-ID* argument.

See also

Related functions

- TDINFPRM on page 129
- TDRCVPRM on page 154

Related documents

- Open Client DB-Library *Reference Manual* (dbrpcparam)

TDLSTSPT

Description

Lists transactions for which tracing is enabled.

Syntax

COPY SYGWCOD.

```
01 IHANDLE      PIC S9(9) USAGE COMP SYNC.
01 RETCODE      PIC S9(9) USAGE COMP SYNC.
01 TRACE-TABLE-LIST OCCURS 8 TIMES
                  PIC X(8).
```

```
CALL 'TDLSTSPT' USING 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 on page 150.

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 <i>IHANDLE</i> 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.

```

* -----
* Describe column containing transaction ID.
* -----
MOVE LENGTH OF WRK-TRANID          TO WRKLEN1.
MOVE LENGTH OF CN-LSTSPT-TRANID TO WRKLEN2.
ADD +1                              TO CTR-COLUMN.
CALL 'TDESCRIB' USING GWL-PROC, GWL-RC,
                        CTR-COLUMN, TDSCHAR,
                        WRKLEN1, WRK-TRANID,
TDS-ZERO, TDS-FALSE,
TDSCHAR, WRKLEN1,
CN-LSTSPT-TRANID, WRKLEN2.
* -----
* Find out whether specific tracing is on; if not, exit.
* -----
CALL 'TDINFLOG' USING 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-INFLOG-GLOBAL NOT = TDS-TRACE-SPECIFIC-RPCS THEN
GO TO TDLSTSPT-EXIT
END-IF.
* -----* Return
trace table IDs to client, one item at a time.
* -----
CALL 'TDLSTSPT' USING GWL-INIT-HANDLE, GWL-RC,
                    GWL-LSTSPT-LIST(1).

```

```

IF GWL-RC NOT = TDS-OK THEN
    MOVE 'N'                TO SEND-DONE-SW
    MOVE 'TDLSTSPT'        TO MSG-SRVLIB-FUNC
    GO TO TDLSTSPT-EXIT
END-IF.
PERFORM VARYING WRK-LSTSPT-SS FROM 1 BY 1
    UNTIL WRK-LSTSPT-SS = 8
    MOVE GWL-LSTSPT-LIST(WRK-LSTSPT-SS) TO WRK-TRANID
    CALL 'TDSNDROW' USING GWL-PROC, GWL-RC
    ADD +1 TO CTR-ROWS
END-PERFORM.

```

Usage

- 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 121
- TDINFSPT on page 136
- TDSETLOG on page 183

- TDSETSPT on page 200
- TDWRTLOG on page 238

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	<pre>COPY SYGWCOB. 01 TDPROC PIC S9(9) USAGE COMP SYNC. 01 NUMBER-OF-PARMS PIC S9(9) USAGE COMP SYNC. CALL 'TDNUMPRM' USING TDPROC, NUMBER-OF-PARMS.</pre>
Parameters	<p><i>TDPROC</i></p> <p>(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The <i>TDPROC</i> handle corresponds to the connection and command handles in Open Client Client-Library.</p> <p><i>NUMBER-OF-PARMS</i></p> <p>(O) Number of parameters accepted as part of the current RPC. This argument replaces the <i>RETCODE</i> argument for this function and is where the result of function execution is stored.</p>
Return value	This function returns the number of parameters in the <i>NUMBER-OF-PARMS</i> argument.
Examples	<p>The following code fragment illustrates a typical use of TDNUMPRM. It does the following: calls TDNUMPRM to determine how many parameters to retrieve; calls TDLOCPRM to ascertain the number of the parameter with the information it wants; calls TDINFPRM for a description of the parameter; calls TDRCVPRM to retrieve the parameter data.</p> <p>This example is taken from the sample program, SYCCSAR2, in Appendix B, “Sample RPC Application for CICS.”</p>

* Get number of parameters ... should be two

```
CALL 'TDNUMPRM' USING GWL-PROC, GWL-NUMPRM-PARMS.
```

```

IF GWL-NUMPRM-PARMS NOT = 2 THEN
    PERFORM TDNUMPRM-ERROR
    GO TO END-PROGRAM
END-IF.

*   Get return parameter information

MOVE 1 TO GWL-INFPRM-ID.
PERFORM GET-PARM-INFO.

(IF GWL-INFPRM-STATUS NOT = TDS-RETURN-VALUE AND
IF GWL-INFPRM-STATUS NOT = TDS-RETURN-VALUE-NULLABLE) THEN
    PERFORM TDINFPRM-NOT-RETURN-PARM-ERROR
    GO TO END-PROGRAM
END-IF.

MOVE GWL-INFPRM-USER-DATA TO GWL-SETPRM-USER-DATA.
MOVE GWL-INFPRM-ID        TO GWL-SETPRM-ID.
MOVE GWL-INFPRM-DATA-L    TO GWL-SETPRM-DATA-L.
MOVE GWL-INFPRM-TYPE      TO GWL-SETPRM-TYPE.

*   Get department id parameter number from known name

MOVE '@parm2' TO GWL-INFPRM-NAME.
MOVE 6        TO GWL-INFPRM-NAME-L.

CALL 'TDLOCPRM' USING GWL-PROC, GWL-INFPRM-ID,
                    GWL-INFPRM-NAME, GWL-INFPRM-NAME-L.

*   Get department parameter information

PERFORM GET-PARM-INFO.

IF GWL-INFPRM-TYPE NOT = TDSVARYCHAR THEN
    PERFORM TDINFPRM-NOT-CHAR-PARM-ERROR
    GO TO END-PROGRAM
END-IF.

*   Get department parameter data

CALL 'TDRCVPRM' USING GWL-PROC, GWL-RC, GWL-INFPRM-ID,
                    PARM-DEPT, GWL-INFPRM-TYPE,
                    GWL-INFPRM-MAX-DATA-L,
                    GWL-RCVPRM-DATA-L.

*-----
GET-PARM-INFO.

```

```
* -----
CALL 'TDINFPRM' USING 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.
```

- Usage
- 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 70
- TDRCVPRM on page 154

TDRCVPRM

Description Retrieves the data from an RPC parameter sent by a remote client.

Syntax COPY SYGWCOB.

```
01 TDPROC          PIC S9(9) USAGE COMP SYNC.
01 RETCODE         PIC S9(9) USAGE COMP SYNC.
01 PARM-ID         PIC S9(9) USAGE COMP SYNC.
01 HOST-VARIABLE   PIC X(n).
01 HOST-VARIABLE-TYPE PIC S9(9) USAGE COMP SYNC.
01 MAX-DATA-LENGTH PIC S9(9) USAGE COMP SYNC.
01 ACTUAL-DATA-LENGTH PIC S9(9) USAGE COMP SYNC.

CALL 'TDRCVPRM' USING 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 155.

PARAM-ID

(I) Number of the parameter or cursor 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.

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-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-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 <i>TDPROC</i> 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.

Return value	Meaning
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

Example 1

The following code fragment illustrates a typical use of TDRCVPRM. The transaction does the following: calls TDNUMPRM to determine how many parameters to retrieve; calls TDLOCPRM to ascertain the number of the parameter whose information it wants; calls TDINFPRM for a description of the parameter; calls TDRCVPRM to retrieve the parameter data.

This example is taken from the sample program, SYCCSAR2, in Appendix B, “Sample RPC Application for CICS.”

```
* Get number of parameters ... should be two

CALL 'TDNUMPRM' USING GWL-PROC, GWL-NUMPRM-PARMS.

IF GWL-NUMPRM-PARMS NOT = 2 THEN
    PERFORM TDNUMPRM-ERROR
    GO TO END-PROGRAM
END-IF.

* Get return parameter information

MOVE 1 TO GWL-INFPRM-ID.
PERFORM GET-PARM-INFO.

(IF GWL-INFPRM-STATUS NOT = TDS-RETURN-VALUE AND
IF GWL-INFPRM-STATUS NOT = TDS-RETURN-VALUE-NULLABLE) THEN
    PERFORM TDINFPRM-NOT-RETURN-PARM-ERROR
    GO TO END-PROGRAM
END-IF.

MOVE GWL-INFPRM-USER-DATA TO GWL-SETPRM-USER-DATA.
MOVE GWL-INFPRM-ID TO GWL-SETPRM-ID.
MOVE GWL-INFPRM-DATA-L TO GWL-SETPRM-DATA-L.
MOVE GWL-INFPRM-TYPE TO GWL-SETPRM-TYPE.

* Get department id parameter number from known name
```

```

MOVE '@parm2' TO GWL-INFPRM-NAME.
MOVE 6          TO GWL-INFPRM-NAME-L.

CALL 'TDLOCPRM' USING GWL-PROC, GWL-INFPRM-ID,
                    GWL-INFPRM-NAME, GWL-INFPRM-NAME-L.

*   Get department parameter information

PERFORM GET-PARM-INFO.

IF GWL-INFPRM-TYPE NOT = TDSVARYCHAR THEN
    PERFORM TDINFPRM-NOT-CHAR-PARM-ERROR
    GO TO END-PROGRAM
END-IF.

*   Get department parameter data

CALL 'TDRCVPRM' USING GWL-PROC, GWL-RC, GWL-INFPRM-ID,
                    PARM-DEPT, GWL-INFPRM-TYPE,
                    GWL-INFPRM-MAX-DATA-L,
                    GWL-RCVPRM-DATA-L.

*-----
GET-PARM-INFO.
*-----
CALL 'TDINFPRM' USING 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.

```

Example 2

The following code fragment illustrates the use of TDRCVPRM in a Gateway-Library program that uses the IMS TM implicit API. This example is taken from the sample program in Appendix D, “Sample RPC Application for IMS TM (Implicit).”

```

*   -----
*   establish gateway environment
*   -----
CALL 'TDINIT' USING IO-PCB, GWL-RC, GWL-INIT-HANDLE.
.
. [check return code]
.

```

```

* -----
*   set program type to MPP
* -----
CALL 'TDSETPT' USING  GWL-INIT-HANDLE, GWL-RC,
                    GWL-PROG-TYPE, GWL-SPA-PTR,
                    TDS-NULL, TDS- NULL.
    . [check return code]
    .
* -----
*   accept client request
* -----
CALL 'TDACCEPT' USING GWL-PROC, GWL-RC, GWL-INIT-HANDLE,
                    SNA-CONNECTION-NAME, SNA-SUBC.
    .
    . [check return code]
    .
* -----
READ-IN-USER-PARM.
* -----
    MOVE 'Y' TO SEND-DONE-SW.
    MOVE 'N' TO ALL-DONE-SW.
    MOVE SPACES TO CALL-ERROR.
    MOVE ZEROES TO CALL-ERROR-RC CTR-ROWS.
    MOVE 1 TO CTR-COLUMN.
    MOVE LENGTH OF PARM-DEPT TO WRKLEN1.
CALL 'TDRCVPRM' USING  GWL-PROC, GWL-RC, PARM-ID1, PARM-DEPT,
                    TDSCHAR, WRKLEN1, PARM-L.
IF GWL-RC NOT EQUAL TO ZEROES THEN
    MOVE 'TDRCVPRM' TO CALL-ERROR
    PERFORM DISPLAY-CALL-ERROR
END-IF.

```

Usage

- 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 *PARAM-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.

Datatype conversions

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*).

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	
TDSLONGVARCHAR	TDSVARYCHAR	Pads TDSCHAR fields with blanks.
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	

Source datatype: Open Client	Result datatype: Gateway-Library	Notes
TDSNUMERIC TDSNUMERIC	TDS-PACKED-DECIMAL TDSCHAR	When receiving numeric or Sybase decimal the <i>MAX-DATA-LENGTH</i> is the maximum length of the result (precision +2, or precision +3 if precision=scale).
TDS-SYBASE-DECIMAL TDS-SYBASE-DECIMAL	TDS-PACKED-DECIMAL TDSCHAR	When receiving numeric or Sybase decimal as packed decimal, use TDINFBCD to determine the host packed decimal precision and scale. <i>MAX-DATA-LENGTH</i> is the actual length of this packed decimal.
TDSCHAR TDSCHAR TDSVARYCHAR TDSVARYCHAR	TDSGRAPHIC TDSVARYGRAPHIC TDSGRAPHIC TDSVARYGRAPHIC	Used with Japanese double-byte character sets. Pads TDSGRAPHIC fields with blanks.
TDSDATETIME TDSDATETIME4	TDSCHAR 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 58 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 3-25 on page 160 lists the implicit conversions that the JCM does when retrieving data.

See also

Related functions

- TDACCEPT on page 70
- TDINFPRM on page 129
- TDLOCPRM on page 146
- TDNUMPRM on page 152
- TDRCVSQL on page 162
- TDSETPRM on page 188

Related topics

- “Character sets” on page 17
- “Datatypes” on page 36
- “Processing Japanese client requests” on page 58

TDRCVSQL

Description

Receives a language string from a remote client.

Syntax

```
COPY SYGWCOB.
01 TDPROC          PIC S9(9) USAGE COMP SYNC.
01 RETCODE         PIC S9(9) USAGE COMP SYNC.
01 HOST-VARIABLE   PIC X(n).
01 MAX-VAR-LENGTH PIC S9(9) USAGE COMP SYNC.
01 ACTUAL-STRING-LENGTH PIC S9(9) USAGE COMP SYNC.
CALL 'TDRCVSQL' USING TDPROC, RETCODE,HOST-VARIABLE,
                      MAX-VAR-LENGTH,ACTUAL-STRING-LENGTH.
```


Parameters	<p><i>TDPROC</i></p> <p>(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The <i>TDPROC</i> handle corresponds to the connection and command handles in Open Client Client-Library.</p> <p><i>RETCODE</i></p> <p>(O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-26 on page 163.</p> <p><i>HOST-VARIABLE</i></p> <p>(O) Host program variable where the text of the retrieved language string is stored.</p> <p><i>MAX-VAR-LENGTH</i></p> <p>(I) Maximum length of the string that can be stored in the named <i>HOST-VARIABLE</i>. For graphic datatypes, this is the number of double-byte characters; for other datatypes, it is the number of bytes.</p> <p><i>ACTUAL-STRING-LENGTH</i></p> <p>(O) The actual length of the incoming data, in bytes. If this length is greater than the specified <i>MAX-VAR-LENGTH</i>, the data is truncated.</p> <hr/> <p>Note If this is a Japanese character set, the length may be halved when converted to IBM Kanji by Gateway-Library.</p>
------------	--

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).

Return value	Meaning
TDS-INVALID-LENGTH (-173)	Wrong length. The length specified in the MAX-VAR-LENGTH argument is too short. The length must be greater than zero.
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 <i>TDPROC</i> 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.
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 TDSQLLEN and TDRCVSQL to receive a language request from the client. This example is taken from the sample program in Appendix C, "Sample Language Application for CICS."

- * Establish gateway environment

```
CALL 'TDINIT' USING DFHEIBLK, GWL-RC, GWL-INIT-HANDLE.
```

- * Turn on local tracing if not on globally or locally

```
CALL 'TDINFLOG' USING 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 NOT = TDS-TRACE-ALL-RPCS
AND  GWL-INFLOG-GLOBAL NOT = TDS-TRACE-SPECIFIC-RPCS THEN
    MOVE 1 TO TRACING-SET-SW
    PERFORM LOCAL-TRACING
END-IF.
```

```

*   Accept client request

CALL 'TDACCEPT' USING 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' USING 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 NOT = TDS-START-SQL
    MOVE MSG-NOT-LANG          TO MSG-TEXT
    MOVE LENGTH OF MSG-NOT-LANG TO MSG-TEXT-L
    PERFORM SEND-ERROR-MESSAGE
    GO TO END-PROGRAM
END-IF.

*   Prepare for receive

CALL 'TDRESULT' USING GWL-PROC, GWL-RC.

*   Get length of language text, ensure not too big for us
*   (this could be handled without TDSQLEN by checking
*   LANG-ACTUAL-LEN doesn't exceed LANG-MAX-L in TDRCVSQL call)

CALL 'TDSQLEN' USING GWL-PROC, GWL-SQLEN.
MOVE LENGTH OF LANG-BUFFER-TEXT TO LANG-MAX-L.

IF GWL-SQLEN > LANG-MAX-L THEN
    MOVE MSG-BAD-LEN          TO MSG-TEXT
    MOVE LENGTH OF MSG-BAD-LEN TO MSG-TEXT-L
    PERFORM SEND-ERROR-MESSAGE
    GO TO END-PROGRAM
END-IF.

*   Get language text
CALL 'TDRCVSQL' USING GWL-PROC, GWL-RC,
                        LANG-BUFFER-TEXT,
                        LANG-MAX-L,

```

```
LANG-ACTUAL-L .
MOVE LANG-ACTUAL-L TO LANG-BUFFER-LL .
```

Usage

- 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, as well as SQL text for cursors.

TDRCVSQL does not differentiate between SQL strings and other character text strings. It is up to your application to 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.
- If you are using a DBCS, be sure to use “G” in the PICTURE clause of the data definition statement. This is required by DB2. For example:

```
10 MYVAR PICTURE GG USAGE DISPLAY-!.
```

See also

Related functions

- TDGETREQ on page 98
- TDINFPGM on page 125

TDRESULT

Description	Determines whether a request is pending and identifies the type of object received.
Syntax	COPY SYGWCOB. 01 TDPROC PIC S9(9) USAGE COMP SYNC. 01 RETCODE PIC S9(9) USAGE COMP SYNC. CALL 'TDRESULT' USING TDPROC, RETCODE.
Parameters	<i>TDPROC</i> (I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The <i>TDPROC</i> handle corresponds to the connection and command handles in Open Client Client-Library. <i>RETCODE</i> (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 <i>RETCODE</i> 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.

Return value	Meaning
TDS-INVALID-TDPROC (-18)	Error in specifying a value for the <i>TDPROC</i> 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.
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

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, SYCCSAR2, in Appendix B, "Sample RPC Application for CICS."

```

*   Establish gateway environment

      CALL 'TDINIT' USING DFHEIBLK, GWL-RC, GWL-INIT-HANDLE.

*   Accept client request

      CALL 'TDACCEPT' USING GWL-PROC, GWL-RC, GWL-INIT-HANDLE,
                          SNA-CONNECTION-NAME, SNA-SUBC.

*   TDRESULT to make sure we were started via RPC request

      CALL 'TDRESULT' USING GWL-PROC, GWL-RC.

      IF GWL-RC NOT = TDS-PARM-PRESENT THEN
          PERFORM TDRESULT-ERROR
          GO TO END-PROGRAM
      END-IF.

* -----
* body of program
* -----
* -----

```

```
END-PROGRAM.
```

```
*-----
  IF SEND-DONE-OK
      MOVE TDS-DONE-COUNT TO WRK-DONE-STATUS
  ELSE
      MOVE TDS-DONE-ERROR TO WRK-DONE-STATUS
      MOVE ZERO           TO PARM-RETURN-ROWS
  END-IF.

  CALL 'TDSNDDON' USING GWL-PROC, GWL-RC, WRK-DONE-STATUS,
                    PARM-RETURN-ROWS, TDS-ZERO,
                    TDS-ENDRPC.

  CALL 'TDFREE' USING GWL-PROC, GWL-RC.

  EXEC CICS RETURN END-EXEC.
```

Usage

- 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 70
- TDRCVPRM on page 154
- TDRCVSQL on page 162

TDSETACT

Description Turns on or off system-wide accounting for Gateway-Library. Under CICS, rename the CICS accounting log.

Syntax COPY SYGWCOB.
 01 IHANDLE PIC S9(9) USAGE COMP SYNC.
 01 RETCODE PIC S9(9) USAGE COMP SYNC.
 01 ACCOUNTING-FLAG PIC S9(4) USAGE COMP SYNC.
 01 ACCOUNTING-FILENAME PIC X(8) VALUE IS SPACES.
 01 MAXNUM-ACCT-RECORDS PIC 9(9) USAGE COMP SYNC.
 CALL 'TDSETACT' USING 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 171.

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. Sybase recommends always setting this value to -1.

Under IMS TM: The IMS TM system log does not have a limit. Sybase recommends 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 <i>IHANDLE</i> specification. Error in specifying a value for the <i>IHANDLE</i> 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 In the following code fragment, the program receives a request to turn accounting on, uses *TDINFACT* to check that accounting is off, then uses *TDSETACT* to turn accounting on. This example is based on the sample program in Appendix G, “Sample Tracing and Accounting Program.” which runs under CICS.

```

*   Accept client request
    CALL 'TDACCEPT' ...
*   -----
    GET-PARM.
*   -----
    CALL 'TDRCVPRM' USING GWL-PROC, GWL-RC,
                        GWL-RCVPRM-ID, PARM-REQUEST, TDSCHAR,
                        GWL-RCVPRM-MAX-DATA-L,
                        GWL-RCVPRM-DATA-L.
    IF PARM-REQUEST-INFACIT THEN
        PERFORM TDINFACT THRU TDINFACT-EXIT
    ELSE IF PARM-REQUEST-SETACT-ON THEN
        PERFORM TDSETACT-ON THRU TDSETACT-ON-EXIT

```

```

*   Request was to set accounting on.
*   -----
      TDSETACT-ON.
*   -----
      CALL 'TDINFACT' USING GWL-INIT-HANDLE, GWL-RC,
                          GWL-INFACT-STATUS,
                          GWL-INFACT-FILENAME,
                          GWL-INFACT-RECORDS.

      IF GWL-RC NOT = TDS-OK THEN
          MOVE 'N' TO SEND-DONE-SW
          MOVE 'TDINFACT' TO MSG-SRVLIB-FUNC
          GO TO TDSETACT-ON-EXIT
      END-IF.
*   Turn on mainframe accounting.
      CALL 'TDSETACT' USING GWL-INIT-HANDLE, GWL-RC,
                          TDS-TRUE, GWL-INFACT-FILENAME,
                          GWL-INFACT-RECORDS.

      IF GWL-RC NOT = TDS-OK THEN
          MOVE 'N' TO SEND-DONE-SW
          MOVE 'TDSETACT' TO MSG-SRVLIB-FUNC
          GO TO TDSETACT-ON-EXIT
      END-IF.
*   -----
      TDSETACT-ON-EXIT.
*   -----
      EXIT.

```

Usage

- 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 70
- TDFREE on page 95
- TDINFACT on page 112

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

```
COPY SYGWCOB.
01 TDPROC      PIC S9(9) USAGE COMP SYNC.
01 RETCODE     PIC S9(9) USAGE COMP SYNC.
01 OBJECT-TYPE PIC S9(9) USAGE COMP SYNC.
01 OBJECT-NUMBER PIC S9(9) USAGE COMP SYNC.
01 BCD-LENGTH  PIC S9(9) USAGE COMP SYNC.
01 BCD-NUMBER-DECIMAL-PLACES
    PIC S9(9) USAGE COMP SYNC.

CALL 'TDSETBCD' USING 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 175.

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: TDSETBCD 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 <i>BCD-LENGTH</i> 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 <i>TDPROC</i> argument.

Examples**Example 1**

The following code fragment shows how to set the column maximum length to 35.

```

MOVE +1                                TO COLUMN-NUMBER.
* need to set the Host Max Length to actual Length *
MOVE LENGTH OF WS-OUTPUT-DECIMAL      TO HOST-LEN
* need to set the Column Max Length to 35 (max len of dec) *
MOVE 35                                TO COLUMN-LEN.
MOVE LENGTH OF WS-OUTPUT-COL-NAME     TO COLUMN-NAME-LEN.

CALL 'TDESCRIB' USING GWL-PROC,
                        GWL-RC,
                        COLUMN-NUMBER,
```

```
TDS-PACKED-DECIMAL,
HOST-LEN,
WS-OUTPUT-DECIMAL,
TDS-ZERO,
TDS-FALSE,
TDS-SYBASE-DECIMAL,
COLUMN-LEN,
WS-OUTPUT-COL-NAME,
COLUMN-NAME-LEN.
```

```
IF GWL-RC NOT = TDS-OK THEN
  MOVE GWL-RC          TO WS-MSG-RC
  MOVE 'TDESCRIB'     TO WS-MSG-FUNC
  PERFORM 920-SEND-MESSAGE THRU 920-EXIT
  PERFORM 910-ERR-PROCESS THRU 910-EXIT
END-IF.
```

```
* move the total number of digits to Percision *
  MOVE 11          TO WS-PERCISION.
* move the total number of digits to right of dec point *
  MOVE 03          TO WS-SCALER.
  CALL 'TDSETBCD' USING GWL-PROC,
                        GWL-RC,
                        TDS-OBJECT-COL,
                        COLUMN-NUMBER,
                        WS-PERCISION,
                        WS-SCALER.
```

```
IF GWL-RC NOT = TDS-OK THEN
  MOVE GWL-RC          TO WS-MSG-RC
  MOVE 'TDSETBCD'     TO WS-MSG-FUNC
  PERFORM 920-SEND-MESSAGE THRU 920-EXIT
  PERFORM 910-ERR-PROCESS THRU 910-EXIT
END-IF.
  PERFORM 310-SEND-ROW      THRU 310-EXIT.
```

Example 2

The following code fragment shows two methods of converting datatypes. One uses TDESCRIB to convert data from the DB2 datatype DECIMAL (TDSDECIMAL) to TDSFLT8. The other uses TDCONVRT to convert data from the DB2 datatype DECIMAL (TDSDECIMAL) to the DB-Library datatype DBMONEY (TDSMONEY).

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, SYCCSAR2, in Appendix B, “Sample RPC Application for CICS.”

```

* Here we let TDESCRIB convert from TDSDECIMAL to TDSFLT8.

CALL 'SYGETAD' USING DB-DESCRIBE-HV-PTR, EMPLOYEE-JC.
CALL 'SYGETAD' USING DB-COLUMN-NAME-HV-PTR, CN-JC.
MOVE LENGTH OF EMPLOYEE-JC TO WRKLEN1.
MOVE LENGTH OF CN-JC      TO WRKLEN2.
MOVE TDSDECIMAL          TO DB-HOST-TYPE.
MOVE TDSFLT8             TO DB-CLIENT-TYPE.
PERFORM DESCRIBE-COLUMN.

* We must inform the Server Library how many decimal places
* are in the EMPLOYEE-JC column.

CALL 'TDSETBCD' USING GWL-PROC, GWL-RC, TDS-OBJECT-COL,
                  CTR-COLUMN, TDS-DEFAULT-LENGTH,
                  GWL-SETBCD-SCALE.

* Demonstrate getting decimal column information.

CALL 'TDINFBCD' USING 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.

CALL 'SYGETAD' USING DB-DESCRIBE-HV-PTR, WRK-EMPLOYEE-SAL.
CALL 'SYGETAD' USING DB-COLUMN-NAME-HV-PTR, CN-SAL.
MOVE LENGTH OF WRK-EMPLOYEE-SAL TO WRKLEN1.
MOVE LENGTH OF CN-SAL          TO WRKLEN2.
MOVE TDSMONEY                  TO DB-HOST-TYPE.
MOVE TDSMONEY                  TO DB-CLIENT-TYPE.
PERFORM DESCRIBE-COLUMN.

PERFORM FETCH-AND-SEND-ROWS
      UNTIL ALL-DONE.

*-----
  FETCH-AND-SEND-ROWS.
*-----

```

```

EXEC SQL FETCH ECURSOR INTO :EMPLOYEE-FIELDS END-EXEC.

IF SQLCODE = 0 THEN

*       Convert from DB2 decimal (TDSDECIMAL) to dblib MONEY.

        MOVE LENGTH OF EMPLOYEE-SAL      TO WRKLEN1
        MOVE LENGTH OF WRK-EMPLOYEE-SAL TO WRKLEN2

        CALL 'TDCONVRT' USING GWL-PROC, GWL-RC,
                           GWL-CONVRT-SCALE, TDSDECIMAL,
                           WRKLEN1, EMPLOYEE-SAL, TDSMONEY,
                           WRKLEN2, WRK-EMPLOYEE-SAL

*       send a row to the client

        CALL 'TDSNDROW' USING GWL-PROC, GWL-RC
        ADD 1 TO PARM-RETURN-ROWS

        IF GWL-RC = TDS-CANCEL-RECEIVED THEN
            MOVE 'Y' TO ALL-DONE-SW
        END-IF

    ELSE IF SQLCODE = +100 THEN
        MOVE 'Y' TO ALL-DONE-SW

    ELSE
        MOVE 'Y' TO ALL-DONE-SW
        PERFORM FETCH-ERROR
    END-IF.

*-----
GET-PARM-INFO.
*-----
        CALL 'TDINFPRM' USING 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.

*-----
DESCRIBE-COLUMN.
*-----
        SET ADDRESS OF LK-DESCRIBE-HV      TO DB-DESCRIBE-HV-PTR.
        SET ADDRESS OF LK-COLUMN-NAME-HV TO DB-COLUMN-NAME-HV-PTR.
        ADD 1                               TO CTR-COLUMN.

```



```
CALL 'TDESCRIB' USING GWL-PROC, GWL-RC, CTR-COLUMN,
                    DB-HOST-TYPE, WRKLEN1, LK-DESCRIBE-HV,
                    DB-NULL-INDICATOR, TDS-FALSE,
                    DB-CLIENT-TYPE, WRKLEN1,
                    LK-COLUMN-NAME-HV, WRKLEN2.
```

Usage

- Packed decimal data is supported in COBOL, 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 COBOL packed decimal data and vice versa.

Note Although the name of this function implies BCD data, in COBOL 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 87

- TDINFBCD on page 116

TDSETLEN

Description Sets the column length for a variable-length field before sending it to a client.

Syntax COPY SYGWCOB.

```
01 TDPROC          PIC S9(9) USAGE COMP SYNC.
01 RETCODE         PIC S9(9) USAGE COMP SYNC.
01 COLUMN-NUMBER  PIC S9(9) USAGE COMP SYNC.
01 NEW-COLUMN-LENGTH PIC S9(9) USAGE COMP SYNC.
```

CALL 'TDSETLEN' USING 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 180.

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.

Return value	Meaning
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 <i>NEW-COLUMN-LENGTH</i> 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 <i>TDPROC</i> argument.

Examples The following code fragment illustrates a typical use of TDSETLEN.

```

* -----
  FETCH-AND-SEND-ROWS.
* -----
      EXEC SQL FETCH ECURSOR INTO :EMPLOYEE-FIELDS
      END-EXEC.
      IF SQLCODE = 0 THEN
* -----
* Convert from DB2 decimal type (TDS-PACKED-DECIMAL) to
* DB-Library MONEY.
* -----
      MOVE LENGTH OF EMPLOYEE-SAL      TO WRKLEN1
      MOVE LENGTH OF WRK-EMPLOYEE-SAL TO WRKLEN2
      CALL 'TDCONVRT' USING   GWL-PROC, GWL-RC,
      GWL-CONVRT-SCALE,
      TDS-PACKED-DECIMAL,
      WRKLEN1, EMPLOYEE-SAL, TDSMONEY,
      WRKLEN2, WRK-EMPLOYEE-SAL
* -----
* Do not send trailing blanks of EMPLOYEE-LNM.
* -----
      MOVE LENGTH OF EMPLOYEE-LNM TO WRKLEN1
      MOVE 2                       TO CTR-COLUMN
      PERFORM VARYING WRK-BLANKS-SS FROM 1 BY 1
      UNTIL WRK-BLANKS-SS > WRKLEN1
      OR EMPLOYEE-LNM-CHARS(WRK-BLANKS-SS) <= SPACE
      END-PERFORM
      IF WRK-BLANKS-SS < WRKLEN1 THEN
          SUBTRACT 1 FROM WRK-BLANKS-SS
          CALL 'TDSETLEN' USING GWL-PROC, GWL-RC, CTR-COLUMN,

```

WRK-BLANKS-SS

```

END-IF
*
* -----
*   Send a row to the client.
* -----
      CALL 'TDSNDROW' USING GWL-PROC, GWL-RC
      ADD 1 TO PARM-RETURN-ROWS
      IF GWL-RC = TDS-CANCEL-RECEIVED THEN
          MOVE 'Y' TO ALL-DONE-SW
      END-IF
ELSE IF SQLCODE = +100 THEN
      MOVE 'Y' TO ALL-DONE-SW
ELSE IF SQLCODE < 0 THEN
      MOVE 'Y' TO ALL-DONE-SW
      PERFORM FETCH-ERROR
END-IF.

```

Usage

- 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 87
- TDSNDROW on page 220

TDSETLOG

Description Sets system-wide tracing for the mainframe server and rename the CICS trace log.

Syntax

```
COPY SYGWCORB.
01 IHANDLE          PIC S9(9) USAGE COMP SYNC.
01 RETCODE          PIC S9(9) USAGE COMP SYNC.
01 GLOBAL-TRACE-FLAG PIC S9(9) USAGE COMP SYNC.
01 API-TRACE-FLAG   PIC S9(9) USAGE COMP SYNC.
01 TDS-HEADER-TRACE-FLAG PIC S9(9) USAGE COMP SYNC.
01 TDS-DATA-TRACE-FLAG PIC S9(9) USAGE COMP SYNC.
01 TRACE-ID         PIC S9(9) USAGE COMP SYNC.
01 TRACE-FILENAME   PIC X(8) VALUE IS SPACES.
01 MAXNUM-TRACE-RECORDS PIC S9(9) USAGE COMP SYNC.

CALL 'TDSETLOG' USING 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 185.

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.
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. Sybase recommends always using -1.

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

Table 3-31: TDSETLOG return values

Return value	Meaning
TDS-OK (0)	Function completed successfully.
TDS-INVALID-IHANDLE (-19)	Invalid <i>IHANDLE</i> specification. Error in specifying a value for the <i>IHANDLE</i> 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 illustrates the use of TDSETLOG to enable TDS header and data tracing. This example is taken from the sample program SYICSAL2, which runs under IMS TM. This book does not contain a listing for SYICSAL2. It is, however, shipped on the product tape.

```
*      turn on local tracing if not on globally or locally
CALL 'TDINFLOG' USING 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.

.
. [check return code]
.
IF GWL-INFLOG-GLOBAL NOT = TDS-TRACE-ALL-RPCS THEN
    PERFORM LOCAL-TRACING
ENDIF.
.
```

```

*-----
LOCAL-TRACING.
*-----
*   turn on specific tracing for SYL2
  MOVE TDS-TRACE-SPECIFIC-RPCS to GWL-INFLOG-GLOBAL.
    MOVE TDS-TRUE TO GWL-INFLOG-TDS-HEADER,
          GWL-INFLOG-TDS-DATA.
MOVE 99 TO GWL-INFLOG-TRACE-ID.
MOVE 'IMSLOG' TO GWL-INFLOG-FILENAME.
MOVE -1 TO GWL-INFLOG-TOTAL-RECS.
CALL 'TDSETLOG' USING 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..
. [check return code]
.
CALL 'TDSETSPT' USING GWL-INIT-HANDLE, GWL-RC,
                    TRACING-SET-SW,
                    GWL-SETSPT-TRACE-LEVEL,
                    GWL-SETSPT-RPC-NAME,
                    GWL-SETSPT-RPC-NAME-L.

```

Usage

- 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 CICS:* API tracing uses the CICS auxiliary trace facility.
 - Under IMS TM:* 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.
- The trace log is also the error log.

- 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, then it 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.
- The specified types of tracing (API, TDS header, and/or TDS data) 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-TRACE-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.
- The log used for tracing depends upon the transaction processing system in use and the type of tracing being 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.
- 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 the Mainframe Connect Server Option *Installation and Administration Guide* for a general discussion of the Gateway-Library tracing facility, instructions for using it, and the layout of the trace log.

See also

Related functions

- TDACCEPT on page 70
- TDFREE on page 95
- TDINFLOG on page 121
- TDINFSPT on page 136
- TDSETSPT on page 200
- TDWRTLOG on page 238

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	<pre> COPY SYGWC0B. 01 TDPROC PIC S9(9) USAGE COMP SYNC. 01 RETCODE PIC S9(9) USAGE COMP SYNC. 01 PARM-ID PIC S9(9) USAGE COMP SYNC. 01 HOST-VARIABLE-TYPE PIC S9(9) USAGE COMP SYNC. 01 HOST-VARIABLE-LENGTH PIC S9(9) USAGE COMP SYNC. 01 HOST-VARIABLE PIC X(n). 01 USER-DATATYPE PIC S9(9) USAGE COMP SYNC. CALL 'TDSETPRM' USING TDPROC, RETCODE, PARM-ID, HOST-VARIABLE-TYPE, HOST-VARIABLE-LENGTH, HOST-VARIABLE, USER-DATATYPE. </pre>
Parameters	<p><i>TDPROC</i> (I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The <i>TDPROC</i> handle corresponds to the connection and command handles in Open Client Client-Library.</p> <p><i>RETCODE</i> (O) Variable where the result of function execution is returned. Its value is one of the codes listed in Table 3-32 on page 190.</p> <p><i>PARM-ID</i> (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.</p> <p><i>HOST-VARIABLE-TYPE</i> (I) Datatype of the <i>HOST-VARIABLE</i>.</p> <p><i>HOST-VARIABLE-LENGTH</i> (I) Length of the <i>HOST-VARIABLE</i>.</p> <p>If <i>HOST-VARIABLE-TYPE</i> 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).</p> <p><i>HOST-VARIABLE</i> (I) Name of the host program variable that contains the return data.</p> <p><i>USER-DATATYPE</i> (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.</p>
Return value	The <i>RETCODE</i> 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 (TSDATETIME4) 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 <i>HOST-VARIABLE-LENGTH</i> 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 <i>TDPROC</i> 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.
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, SYCCSAR2, in Appendix B, “Sample RPC Application for CICS.”

```
PERFORM DESCRIBE-COLUMN.
PERFORM FETCH-AND-SEND-ROWS UNTIL ALL-DONE
*   Update returned parameter with number of rows fetched
    CALL 'TDSETPRM' USING   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

- A server application uses this function to tell TDSNDDON where to find the data for a return parameter and 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 *PARAM-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 *PARAM-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 *PARAM-STATUS* and a *PARAM-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 that 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 a char or packed decimal datatype to a client numeric or Sybase decimal datatype, use TDSETPBCD 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 192 shows what conversions are possible.

Table 3-33: Datatype conversions performed by TDSETPRM

Source datatype: Gateway-Library	Result datatype: Open Client	Notes
TDSCHAR TDSCHAR TDSVARYCHAR TDSVARYCHAR TDSLONGVARCHAR TDSLONGVARCHAR TDSMONEY TDSMONEY	TDSVARYCHAR TDSLONGVARCHAR TDSCHAR TDSLONGVARCHAR TDSCHAR TDSVARYCHAR TDSCHAR TDSVARYCHAR	Does EBCDIC to ASCII conversion. For Japanese characters, converts to workstation datatype. Pads TDSCHAR fields with blanks.
TDSFLT8 TDSFLT8 TDSFLT8 TDSFLT4 TDSFLT4 TDSFLT4	TDSFLT4 TDSMONEY TDSMONEY4 TDSFLT8 TDSMONEY TDSMONEY4	Truncates low order digits.
TDSCHAR TDSVARYCHAR	TDSMONEY TDSMONEY	
TDS-PACKED-DECIMAL TDS-PACKED-DECIMAL TDS-PACKED-DECIMAL TDS-PACKED-DECIMAL	TDSCHAR TDSVARYCHAR TDSFLT8 TDSMONEY	When converting packed decimal to character values, change the length to allow for unpacking, leading or trailing zeros, the sign and the decimal point.
TDSGRAPHIC TDSGRAPHIC TDSVARYGRAPHIC TDSVARYGRAPHIC	TDSCHAR TDSVARYCHAR TDSCHAR TDSVARYCHAR	Used with Japanese double-byte character sets. Pads TDSCHAR fields with blanks.
TDSDATETIME TDSDATETIME4	TDSCHAR TDSCHAR	
TDSCHAR TDSCHAR	TDSNUMERIC TDS-SYBASE-DECIMAL	Use TDSETBBCD to set Sybase numeric or decimal precision and scale before TDSETPRM.
TDS-PACKED-DECIMAL TDS-PACKED-DECIMAL	TDSNUMERIC TDS-SYBASE-DECIMAL	Use TDSETBBCD to set Sybase numeric or decimal precision and scale before TDSETPRM.

See also

Related functions

- TDINFPRM on page 129
- TDRCVPRM on page 154
- TDSNDDON on page 206

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. MVS programs do not ignore this call.

Syntax

```
COPY SYGWCOB.
01 IHANDLE PIC S9(9) USAGE COMP SYNC.
01 RETCODE PIC S9(9) USAGE COMP SYNC.
01 PROG-TYPE PIC X(4).
01 SPA PIC X(n).
01 RESERVED1 PIC S9(9) USAGE COMP SYNC.
01 RESERVED1 PIC S9(9) USAGE COMP SYNC.
CALL 'TDSETPT' USING 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 194.

PROG-TYPE

(I) Type of IMS TM program being called. This is a 4-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 on page 194.

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 <i>IHANDLE</i> 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.

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

```

*      establish gateway environment
      CALL 'TDINIT' USING IO-PCB, GWL-RC, GWL-INIT-HANDLE.
          . [check return code]
      .
*      set program type to MPP
      CALL 'TDSETPT' USING GWL-INIT-HANDLE, GWL-RC,
                          GWL-PROG-TYPE, GWL-SPA-PTR,
                          TDS-NULL, TDS- NULL.
          . [check return code]
      .
*      accept client request
      CALL 'TDACCEPT' USING GWL-PROC, GWL-RC, GWL-INIT-HANDLE,
                          SNA-CONNECTION-NAME,
                          SNA-SUBC.
*-----
      READ-IN-USER-PARM.
*-----

```

Usage

- TDSETPT 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.
- See “Long-running transactions” on page 54, for a discussion of long-running transactions under both CICS and IMS TM.

- For more information, refer to your IMS TM product documentation.

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 70
- TDGETREQ on page 98
- TDINIT on page 142
- TDTERM on page 232

Related topics

- “Long-running transactions” on page 54

TDSETSOI

Description

Set 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

COPY SYGWCOB.

```
01 TDPROC      PIC S9(9) USAGE COMP SYNC.
01 RETCODE    PIC S9(9) USAGE COMP SYNC.
01 OBJECT-TYPE PIC S9(9) USAGE COMP SYNC.
01 OBJECT-NUMBER PIC S9(9) USAGE COMP SYNC.
01 STRIP-SOSI PIC S9(9) USAGE COMP SYNC.
```

```
CALL 'TDSETSOI' USING 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 197.

OBJECT-TYPE

(I) Indicator for the 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 host 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 listed in Table 3-35 on page 197.

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.

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 <i>TDPROC</i> argument.

Examples The following code fragment uses TDSETSOI to replace SO/SI codes with blanks before retrieving parameters and again before returning data to the client. This example is not included on the Open ServerConnect API tape, but is available to Japanese customers on the Japanese Conversion Module tape.

```

*****
PROCEDURE DIVISION.
*****
*
*   CALL 'TDINIT'    USING DFHEIBLK, GWL-RC, GWL-INIT-HANDLE.
*
*   CALL 'TDACCEPT' USING GWL-PROC, GWL-RC, GWL-INIT-HANDLE,
*                       SNA-CONNECTION-NAME,
*                       SNA-SUBC.
*
*   CALL 'TDINFRPC' USING GWL-PROC, GWL-RC, GWL-REQ-TYPE,
*                       GWL-RPC-NAME, GWL-COMM-STATE.
*
*   get the information of so-so
MOVE TDS-OBJECT-PARM  TO PRM-01-OBJ-TYPE.
MOVE PRM-01-ID       TO PRM-01-OBJ-ID.
CALL 'TDGETSOI' USING GWL-PROC, GWL-RC,
                    PRM-01-OBJ-TYPE,
                    PRM-01-OBJ-ID,
                    PRM-01-STRIP-SOSI.
*
*   IF PRM-01-STRIP = TDS-STRIP-SOSI
*   THEN
* specify the embedded blanks to the parameter
MOVE TDS-BLANK-SOSI  TO PRM-01-STRIP-SOSI
CALL 'TDSETSOI' USING GWL-PROC, GWL-RC,
                    PRM-01-OBJ-TYPE,
                    PRM-01-OBJ-ID,
                    PRM-01-STRIP-SOSI
*
*   END-IF
*
*   MOVE TDSCHAR      TO PRM-01-HOST-TYPE.
*

```

```

MOVE LENGTH OF PRM-01-DATA TO PRM-01-MAX-LEN.
*
CALL 'TDRCVPRM' USING GWL-PROC, GWL-RC,
                    PRM-01-ID,
                    PRM-01-AREA,
                    PRM-01-HOST-TYPE,
                    PRM-01-MAX-LEN,
                    PRM-01-ACT-LEN.
*
CALL 'TDESCRIB' USING GWL-PROC, GWL-RC,
                    COL-01-NUM,
                    COL-01-HOST-TYPE,
                    COL-01-HOST-LEN,
                    COL-01-AREA,
                    COL-01-NULL-INDICATOR,
                    TDS-FALSE,
                    COL-01-CLIENT-TYPE,
                    COL-01-CLIENT-LEN,
                    COL-01-NAME,
                    COL-01-NAME-LEN.
*
get the information of sosi
MOVE TDS-OBJECT-COL TO COL-01-OBJ-TYPE.
MOVE COL-01-NUM TO COL-01-OBJ-ID.
CALL 'TDGETSOI' USING GWL-PROC, GWL-RC,
                    COL-01-OBJ-TYPE,
                    COL-01-OBJ-ID,
                    COL-01-STRIP-SOSI.
*
IF COL-01-STRIP-SOSI = TDS-STRIP-SOSI
THEN
*
*
specify the embedded blanks to the column
MOVE TDS-BLANK-SOSI TO COL-01-STRIP-SOSI
CALL 'TDSETSOI' USING GWL-PROC, GWL-RC,
                    COL-01-OBJ-TYPE,
                    COL-01-OBJ-ID,
                    COL-01-STRIP-SOSI
END-IF
PERFORM FETCH-AND-SEND-ROWS UNTIL ALL-DONE.

```

Usage

- Use TDSETSOI 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 58.

See also

Related functions

- TDGETSOI on page 104

Related topics

- “Character sets” on page 17
- “Processing Japanese client requests” on page 58

TDSETSPT

Description

Sets tracing on or off for a specified transaction.

Syntax

COPY SYGWCOB.

```
01 IHANDLE          PIC S9(9) USAGE COMP SYNC.
01 RETCODE          PIC S9(9) USAGE COMP SYNC.
01 TRACE-STATUS    PIC S9(9) USAGE COMP SYNC.
01 TRACE-OPTIONS   PIC S9(9) USAGE COMP SYNC.
01 TRANSACTION-ID  PIC X(n).
01 TRANSACTION-ID-LENGTH PIC S9(9) USAGE COMP SYNC.

CALL 'TDSETSPT' USING 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 202.

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 <i>IHANDLE</i> 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 <i>STATUS</i> field is invalid.
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 shows how to use TDINFLOG at the beginning of a program to determine which types of tracing are currently enabled and TDSETSPT at the end of a program. This example is taken from the sample program in Appendix C, “Sample Language Application for CICS.”

* Establish gateway environment

```
CALL 'TDINIT' USING DFHEIBLK, GWL-RC, GWL-INIT-HANDLE.
```



```

*      Turn on local tracing if not on globally or locally
      CALL 'TDINFLOG' USING 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 NOT = TDS-TRACE-ALL-RPCS
      AND GWL-INFLOG-GLOBAL NOT = TDS-TRACE-SPECIFIC-RPCS THEN
      MOVE 1 TO TRACING-SET-SW
      PERFORM LOCAL-TRACING
      END-IF.

*      Accept client request

      CALL 'TDACCEPT' USING GWL-PROC, GWL-RC, GWL-INIT-HANDLE,
                          SNA-CONNECTION-NAME,
                          SNA-SUBC.

*-----
LOCAL-TRACING.
*-----
      CALL 'TDSETSPT' USING GWL-INIT-HANDLE, GWL-RC,
                          TRACING-SET-SW,
                          GWL-SETSPT-TRACE-LEVEL,
                          GWL-SETSPT-RPC-NAME,
                          GWL-SETSPT-RPC-NAME-L.

```

Usage

- TDSETSPT 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 on 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.

- Because 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 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 121
- TDINFSPT on page 136
- TDLSTSPT on page 149
- TDSETLOG on page 183
- TDWRTLOG on page 238

TDSETUDT

Description

Sets the user-defined datatype for the specified column.

Syntax

COPY SYGWCOB.

```
01 TDPROC      PIC S9(9) USAGE COMP SYNC.  
01 RETCODE     PIC S9(9) USAGE COMP SYNC.  
01 COLUMN-NUMBER PIC S9(9) USAGE COMP SYNC.  
01 USER-DATATYPE PIC S9(9) USAGE COMP SYNC.
```

```
CALL 'TDSETUDT' USING 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 <i>TDPROC</i> argument.

Examples The following code fragment illustrates a typical use of TDSETUDT. This example is taken from the sample program, SYCCSAR2, in Appendix B, “Sample RPC Application for CICS.”

- * Here we let TDESCRIB convert from DB2 varchar (TDSVARYCHAR)
- * to DBCHAR.

```
CALL 'SYGETAD' USING DB-DESCRIBE-HV-PTR, EMPLOYEE-ED.
CALL 'SYGETAD' USING DB-COLUMN-NAME-HV-PTR, CN-ED.
MOVE LENGTH OF EMPLOYEE-ED TO WRKLEN1.
MOVE LENGTH OF CN-ED      TO WRKLEN2.
MOVE TDSINT2              TO DB-HOST-TYPE.
MOVE TDSINT2              TO DB-CLIENT-TYPE.
PERFORM DESCRIBE-COLUMN.
```

- * Get the user defined datatype of EMPLOYEE-ED column.

```
CALL 'TDINFUDT' USING GWL-PROC, GWL-RC, CTR-COLUMN,
                  GWL-INFUDT-USER-TYPE.
```

- * Set the user defined datatype of EMPLOYEE-ED column.

```
CALL 'TDSETUDT' USING GWL-PROC, GWL-RC, CTR-COLUMN,
                    GWL-INFUDT-USER-TYPE.
```

```
*-----
DESCRIBE-COLUMN.
```

```
*-----
SET ADDRESS OF LK-DESCRIBE-HV      TO DB-DESCRIBE-HV-PTR.
SET ADDRESS OF LK-COLUMN-NAME-HV  TO DB-COLUMN-NAME-HV-PTR.
ADD 1                               TO CTR-COLUMN.
```

```
CALL 'TDESCRIB' USING GWL-PROC, GWL-RC, CTR-COLUMN,
                    DB-HOST-TYPE, WRKLEN1, LK-DESCRIBE-HV,
                    DB-NULL-INDICATOR, TDS-FALSE,
                    DB-CLIENT-TYPE, WRKLEN1,
                    LK-COLUMN-NAME-HV, WRKLEN2.
```

- Usage
- Use TDSETUDT 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 129
- TDSETPRM on page 188
- TDSETUDT on page 204

TDSNDDON

Description Sends a results completion indication to the client.

Syntax COPY SYGWCOB.

```
01 TDPROC          PIC S9(9) USAGE COMP SYNC.
01 RETCODE         PIC S9(9) USAGE COMP SYNC.
01 STATUS          PIC S9(9) USAGE COMP SYNC.
01 ROW-COUNT       PIC S9(9) USAGE COMP SYNC.
```

01 RETURN-STATUS-NUMBER PIC S9(9) USAGE COMP SYNC.
 01 CONN-OPTIONS PIC S9(9) USAGE COMP SYNC.
 CALL 'TDSNDDON' USING 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 209.

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 <i>STATUS</i> is TDS-DONE-FINAL, <i>CONN-OPTIONS</i> must be TDS-ENDREPLY or TDS-ENDRPC. <i>Note:</i> 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 <i>STATUS</i> is TDS-DONE-CONTINUE, <i>CONN-OPTIONS</i> must be TDS-FLUSH.
TDS-DONE-ERROR (0x0002)	The last request received from the client resulted in an error.
TDS-DONE-COUNT (0x0010)	The <i>ROW-COUNT</i> 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's 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 208. 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.

Table 3-38: List of Sybase return status values

Value	Meaning
0	Procedure executed without error.
-1	Missing object.
-2	Datatype error.
-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.

Note This value cannot be NULL.

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)	<p>Indicates that the reply data stream ended. The communication state is changed from SEND to RECEIVE, and the transaction awaits the next request.</p> <p>When you use this value, <i>STATUS</i> must be TDS- DONE-FINAL. For IMS TM transactions, the TDSETP <i>PROG-TYPE</i> parameter must be EXPL.</p> <hr/> <p>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.</p>
TDS-ENDRPC (3)	<p>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.</p> <p>When you use this value, <i>STATUS</i> must be TDS-DONE-FINAL.</p>
TDS-FLUSH (7)	<p>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.</p> <p>If <i>CONN-OPTIONS</i> is TDS-FLUSH, <i>STATUS</i> must be TDS-DONE-CONTINUE.</p>
Return value	<p>The <i>RETCODE</i> argument can contain any of the return values listed in Table 3-39.</p>

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).

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-STATUS (-174)	Invalid status value. The value entered in the <i>STATUS</i> field is invalid.
TDS-INVALID-TDPROC (-18)	Error in specifying a value for the <i>TDPROC</i> 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, SYCCSAR2, in Appendix B, "Sample RPC Application for CICS."

```
*   Establish gateway environment

      CALL 'TDINIT' USING DFHEIBLK, GWL-RC, GWL-INIT-HANDLE.

*   Accept client request

      CALL 'TDACCEPT' USING GWL-PROC, GWL-RC, GWL-INIT-HANDLE,
                          SNA-CONNECTION-NAME, SNA-SUBC.

*   TDRESULT to make sure we were started via RPC request

      CALL 'TDRESULT' USING GWL-PROC, GWL-RC.

      IF GWL-RC NOT = TDS-PARM-PRESENT THEN
          PERFORM TDRESULT-ERROR
          GO TO END-PROGRAM
      END-IF.

* -----
*   body of program
* -----
* -----
END-PROGRAM.
* -----
      IF SEND-DONE-OK
          MOVE TDS-DONE-COUNT TO WRK-DONE-STATUS
      ELSE
```



```

        MOVE TDS-DONE-ERROR TO WRK-DONE-STATUS
        MOVE ZERO           TO PARM-RETURN-ROWS
END-IF.

CALL 'TDSNDDON' USING GWL-PROC, GWL-RC, WRK-DONE-STATUS,
                    PARM-RETURN-ROWS, TDS-ZERO,
                    TDS-ENDRPC.

CALL 'TDFREE' USING GWL-PROC, GWL-RC.

EXEC CICS RETURN END-EXEC.

```

Example 2

The following code fragment illustrates the use of TDSNDDON and TDGETREQ in a Gateway-Library transaction using the IMS TM implicit API. This example is taken from the sample program in Appendix D, “Sample RPC Application for IMS TM (Implicit).”

```

*-----
SEND-ROWS.
*-----
        PERFORM FETCH-AND-SEND-ROWS
                UNTIL ALL-DONE.
        FINISH-REPLY.

        CALL 'TDSNDDON' USING GWL-PROC, GWL-RC,
                WRK-DONE-STATUS,
                CTR-ROWS,
                TDS-ZERO,
                TDS-ENDRPC.
        . [check return code]

*
Get next client request
MOVE TDS-TRUE TO GWL-WAIT-OPTION.
MOVE ZEROES TO GWL-REQ-TYPE.
MOVE SPACES TO GWL-RPC-NAME.
CALL 'TDGETREQ' USING GWL-PROC, GWL-RC, GWL-WAIT-OPTION,
                GWL-REQ-TYPE, GWL-RPC-NAME.
EVALUATE GWL-RC
        WHEN ZEROES
                GO TO READ-IN-USER-PARM
        WHEN TDS-RESULTS-COMPLETE
                PERFORM FREE-ALL-STORAGE
        WHEN TDS-CONNECTION-TERMINATED
                PERFORM FREE-ALL-STORAGE
        WHEN OTHER
                MOVE 'TDGETREQ' TO CALL-ERROR

```

PERFORM DISPLAY-CALL-ERROR
END-EVALUATE.
GOBACK.

Usage

- 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.

For Long-Running Transactions

Note IMS TM Users: Long-running transactions are only supported for the explicit API (the TDSETP *PROG-TYPE* parameter is set to EXPL).

- With short transactions, a transaction ends after it sends results to the client; in long-running transactions, it stays active and processes new requests as they are sent.
- 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.

Therefore, 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 Open ServerConnect transaction, Open ServerConnect may forward results to the client before it is aware that the client canceled the request.

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 70
- TDRESULT on page 167
- TDSETPRM on page 188

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

COPY SYGWCOB.

```
01 TDPROC          PIC S9(9) USAGE COMP SYNC.
01 RETCODE         PIC S9(9) USAGE COMP SYNC.
01 MESSAGE-TYPE    PIC S9(9) USAGE COMP SYNC.
01 MESSAGE-NUMBER PIC S9(9) USAGE COMP SYNC.
01 SEVERITY        PIC S9(9) USAGE COMP SYNC.
01 ERROR-STATE     PIC S9(9) USAGE COMP SYNC.
01 LINE-ID         PIC S9(9) USAGE COMP SYNC.
01 TRANSACTION-ID  PIC X(n).
01 TRANSACTION-ID-LENGTH PIC S9(9) USAGE COMP SYNC.
01 MESSAGE-TEXT    PIC X(n).
01 MESSAGE-LENGTH  PIC S9(9) USAGE COMP SYNC.
```

CALL 'TDSNDMSG' USING 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 216.

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 *SMMSG-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 *SMMSG-LINE* field of the Open Client CS-SERVERMSG structure.

TRANSACTION-ID

(I) Identifier of the transaction that is currently executing. This value identifies the transaction that is issuing the error message.

Under CICS: This is the *TRANSID* from the CICS 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 *SMMSG-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 *SMMSG-TEXT-LEN* field of the 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).
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 <i>MESSAGE-LENGTH</i> 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 <i>STATUS</i> field is invalid.
TDS-INVALID-TDPROC (-18)	Error in specifying a value for the <i>TDPROC</i> argument.

Return value	Meaning
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

Example 1

The following code fragment shows how a program uses TDSNDMSG to send an error message to a client. This example is taken from the sample program, SYCCSAR2, in Appendix B, “Sample RPC Application for CICS.”

```

* -----
SEND-SQL-ERROR.
* -----

MOVE SQLCODE TO MSG-SQL-ERROR-C.
MOVE SQLERRMC TO MSG-SQL-ERROR-K.

* -----
* ensure possible non-printables translated to spaces
* -----
PERFORM VARYING MSG-SQL-ERROR-SS FROM 1 BY 1
UNTIL MSG-SQL-ERROR-SS > SQLERRML

IF MSG-SQL-ERROR-K-CHARS(MSG-SQL-ERROR-SS) < SPACE
OR MSG-SQL-ERROR-K-CHARS(MSG-SQL-ERROR-SS) > '9' THEN
MOVE SPACE TO MSG-SQL-ERROR-K-CHARS(MSG-SQL-ERROR-SS)
END-IF

END-PERFORM.

MOVE MSG-SQL-ERROR TO MSG-TEXT.
MOVE LENGTH OF MSG-SQL-ERROR TO MSG-TEXT-L.
PERFORM SEND-ERROR-MESSAGE.

* -----
SEND-ERROR-MESSAGE.
* -----

MOVE 'N' TO SEND-DONE-SW.
MOVE MSG-SEVERITY-ERROR TO MSG-SEVERITY.
MOVE MSG-NR-ERROR TO MSG-NR.
MOVE TDS-ERROR-MSG TO MSG-TYPE.
PERFORM SEND-MESSAGE.

```

```
*-----  
SEND-MESSAGE.  
*-----  
MOVE LENGTH OF MSG-RPC TO MSG-RPC-L.  
  
*   Ensure we're in right state to send a message  
  
CALL 'TDSTATUS' USING 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 AND  
    GWL-STATUS-COMM = TDS-RECEIVE) THEN  
  
    CALL 'TDSNDMSG' USING 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-IF.
```

Example 2

This code fragment illustrates the use of TDSTATUS and TDSNDMSG in a Gateway-Library transaction using the IMS TM implicit API. This example is taken from the sample program in Appendix D, "Sample RPC Application for IMS TM (Implicit)."

```
*-----  
SEND-ERROR-MESSAGE.  
*-----  
MOVE 'N'                TO SEND-DONE-SW.  
MOVE TDS-ERROR-MSG      TO MSG-TYPE.  
MOVE LENGTH OF MSG-RPC TO MSG-RPC-L.  
  
*   Ensure we're in right state to send a message  
  
CALL 'TDSTATUS' USING 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 AND
    GWL-STATUS-COMM = TDS-RECEIVE) THEN

    CALL 'TDSNDMSG' USING 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-IF.

```

Usage

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.

- A server application uses this function to send an error or informational message to a remote client.
- 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 a TDSNDDON call is issued.

For Japanese users

- If the 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

COPY SYGWCORB.

01 TDPROC PIC S9(9) USAGE COMP SYNC.

01 RETCODE PIC S9(9) USAGE COMP SYNC.

CALL 'TDSNDROW' USING 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.

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-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 <i>TDPROC</i> 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.
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 in a paragraph that converts packed decimal data to the client money datatype before sending the row to the client. This example is taken from the sample program, SYCCSAR2, in Appendix B, “Sample RPC Application for CICS.”

```

* -----
  FETCH-AND-SEND-ROWS.
* -----
      EXEC SQL FETCH ECURSOR INTO :EMPLOYEE-FIELDS END-EXEC.

      IF SQLCODE = 0 THEN

*           Convert from DB2 decimal (TDSDECIMAL) to dblib MONEY.

```

```
MOVE LENGTH OF EMPLOYEE-SAL      TO WRKLEN1
MOVE LENGTH OF WRK-EMPLOYEE-SAL  TO WRKLEN2

CALL 'TDCONVRT' USING GWL-PROC, GWL-RC,
                        GWL-CONVRT-SCALE, TDSDECIMAL,
                        WRKLEN1, EMPLOYEE-SAL, TDSMONEY,
                        WRKLEN2, WRK-EMPLOYEE-SAL

*      send a row to the client

CALL 'TDSNDROW' USING GWL-PROC, GWL-RC
ADD 1 TO PARM-RETURN-ROWS

IF GWL-RC = TDS-CANCEL-RECEIVED THEN
    MOVE 'Y' TO ALL-DONE-SW
END-IF

ELSE IF SQLCODE = +100 THEN
    MOVE 'Y' TO ALL-DONE-SW

ELSE
    MOVE 'Y' TO ALL-DONE-SW
    PERFORM FETCH-ERROR
END-IF.
```

Usage

- 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, every 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	
TDS-PACKED-DECIMAL	TDSNUMERIC	Use TDESETBCD after TDESCRIB to set precision and scale for numeric or Sybase decimal columns.
TDSCHAR	TDSNUMERIC	
TDS-PACKED-DECIMAL	TDS-SYBASE-DECIMAL	Use TDESETBCD after TDESCRIB to set precision and scale for numeric or Sybase decimal columns.
TDSCHAR	TDS-SYBASE-DECIMAL	

Source datatype: Gateway-Library	Result datatype: Open Client	Notes
TDSGRAPHIC	TDSCHAR	Performed by Japanese Conversion Module.
TDSGRAPHIC	TDSVARYCHAR	
TDSVARGRAPHIC	TDSCHAR	Pads TDSCHAR fields with blanks.
TDSVARGRAPHIC	TDSVARYCHAR	
TDSDATETIME	TDSCHAR	
TDSDATETIME4	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. Calling TDRESULT puts your application in SEND state.
- If the *RETCODE* argument contains the value, TDS-CANCELRECEIVED, 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 87
- TDSNDDON on page 206
- TDSNDMSG on page 213

Related topics

- “Datatypes” on page 36

TDSQLLEN

Description

Determines the length of a language string received from a client.

Syntax

COPY SYGWCOB.

01 TDPROC PIC S9(9) USAGE COMP SYNC.

01 SQL-LENGTH PIC S9(9) USAGE COMP SYNC.

CALL 'TDSQLLEN' USING TDPROC, SQL-LENGTH.

Parameters	<p><i>TDPROC</i></p> <p>(I) Handle for this client/server connection. This must be the same value specified in the associated TDACCEPT call. The <i>TDPROC</i> handle corresponds to the connection and command handles in Open Client Client-Library.</p> <p><i>SQL-LENGTH</i></p> <p>(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.</p>
Return value	This function has no <i>RETCODE</i> argument. It returns the length of the SQL string in the <i>SQL-LENGTH</i> argument. If the value in <i>SQL-LENGTH</i> is -1, call TDRESULT, and examine its return code to determine what the problem is.
Examples	<p>The following code fragment illustrates the use of TDSQLEN and TDRCVSQL to receive a language request from the client. This example is taken from the sample program in Appendix C, "Sample Language Application for CICS."</p> <p>* Establish gateway environment</p> <pre>CALL 'TDINIT' USING DFHEIBLK, GWL-RC, GWL-INIT-HANDLE.</pre> <p>* Turn on local tracing if not on globally or locally</p> <pre>CALL 'TDINFLOG' USING 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.</pre> <pre>IF GWL-INFLOG-GLOBAL NOT = TDS-TRACE-ALL-RPCS AND GWL-INFLOG-GLOBAL NOT = TDS-TRACE-SPECIFIC-RPCS THEN MOVE 1 TO TRACING-SET-SW PERFORM LOCAL-TRACING END-IF.</pre> <p>* Accept client request</p> <pre>CALL 'TDACCEPT' USING GWL-PROC, GWL-RC, GWL-INIT-HANDLE, SNA-CONNECTION-NAME, SNA-SUBC.</pre> <p>* Ensure kicked off via language request</p> <p>* (this could be handled more reasonably by TDRESULT)</p>

```
CALL 'TDINFPGM' USING 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 NOT = TDS-START-SQL  
    MOVE MSG-NOT-LANG          TO MSG-TEXT  
    MOVE LENGTH OF MSG-NOT-LANG TO MSG-TEXT-L  
    PERFORM SEND-ERROR-MESSAGE  
    GO TO END-PROGRAM  
END-IF.
```

* Prepare for receive

```
CALL 'TDRESULT' USING GWL-PROC, GWL-RC.
```

* Get length 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' USING GWL-PROC, GWL-SQLLEN.  
MOVE LENGTH OF LANG-BUFFER-TEXT TO LANG-MAX-L.
```

```
IF GWL-SQLLEN > LANG-MAX-L THEN  
    MOVE MSG-BAD-LEN          TO MSG-TEXT  
    MOVE LENGTH OF MSG-BAD-LEN TO MSG-TEXT-L  
    PERFORM SEND-ERROR-MESSAGE  
    GO TO END-PROGRAM  
END-IF.
```

* Get language text

```
CALL 'TDRCVSQL' USING GWL-PROC, GWL-RC,  
                    LANG-BUFFER-TEXT,  
                    LANG-MAX-L,  
                    LANG-ACTUAL-L.
```

```
MOVE LANG-ACTUAL-L TO LANG-BUFFER-LL.
```

Usage

- A server application uses this function to determine the actual length of an incoming string.

- Typically, an application calls TDSQLEN after TDACCEPT and before TDRCVSQL to determine how large a storage area to allocate for the incoming string.
- You can use TDSQLEN to store incoming text in more than one variable. Read part of the text into one variable, using TDRCVSQL, then call TDSQLEN 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 TDSQLEN, 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.

See also

Related functions

- TDRCVSQL on page 162
- TDRESULT on page 167

TDSTATUS

Description Retrieves the last status information received from a remote procedure call (RPC) or SQL command string.

Syntax

```
COPY SYGWC0B.
01 TDPROC          PIC S9(9)  USAGE COMP SYNC.
01 RETCODE         PIC S9(9)  USAGE COMP SYNC.
01 RETURN-STATUS-NUMBER PIC S9(9)  USAGE COMP SYNC.
01 DONE-STATUS     PIC S9(9)  USAGE COMP SYNC.
01 DONE-COUNT      PIC S9(9)  USAGE COMP SYNC.
01 COMM-STATE      PIC S9(9)  USAGE COMP SYNC.
01 COMM-RETCODE    PIC S9(9)  USAGE COMP SYNC.
01 COMM-ERROR-SUBCODE PIC S9(9)  USAGE COMP SYNC.

CALL 'TDSTATUS' USING 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 229.

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 206.

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 <i>ROW-COUNT</i> 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.

A list of these codes is in the 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 3-43.

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.
TDS-INVALID-TDPROC (-18)	Error in specifying a value for the <i>TDPROC</i> 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

Example 1

The following code fragment shows how a program uses TDSTATUS to determine the communication state before sending an error message to a client. This example is taken from the sample program, SYCCSAR2, in Appendix B, “Sample RPC Application for CICS.”

```

*-----
SEND- SQL-ERROR .
*-----
      MOVE SQLCODE  TO MSG-SQL-ERROR-C .
      MOVE SQLERRMC TO MSG-SQL-ERROR-K .
*-----

```

```

*   ensure possible non-printables translated to spaces
*   -----
      PERFORM VARYING MSG-SQL-ERROR-SS FROM 1 BY 1
          UNTIL MSG-SQL-ERROR-SS > SQLERRML

          IF MSG-SQL-ERROR-K-CHARS(MSG-SQL-ERROR-SS) < SPACE
            OR MSG-SQL-ERROR-K-CHARS(MSG-SQL-ERROR-SS) > '9' THEN
              MOVE SPACE TO MSG-SQL-ERROR-K-CHARS(MSG-SQL-ERROR-SS)
            END-IF
          END-PERFORM.
      MOVE MSG-SQL-ERROR          TO MSG-TEXT.
      MOVE LENGTH OF MSG-SQL-ERROR TO MSG-TEXT-L.
      PERFORM SEND-ERROR-MESSAGE.
*   -----
      SEND-ERROR-MESSAGE.
*   -----
      MOVE 'N'          TO SEND-DONE-SW.
      MOVE MSG-SEVERITY-ERROR TO MSG-SEVERITY.
      MOVE MSG-NR-ERROR      TO MSG-NR.
      MOVE TDS-ERROR-MSG     TO MSG-TYPE.
      PERFORM SEND-MESSAGE.
*   -----
      SEND-MESSAGE.
*   -----
      MOVE LENGTH OF MSG-RPC TO MSG-RPC-L.
*   Ensure we're in right state to send a message

      CALL 'TDSTATUS' USING 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 AND
          GWL-STATUS-COMM = TDS-RECEIVE) THEN

          CALL 'TDSNDMSG' USING 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-IF.

```

Example 2

The following code fragment illustrates the use of TDSTATUS and TDSNDMSG in a Gateway-Library transaction using the IMS TM implicit API.

This example is taken from the sample program in Appendix D, “Sample RPC Application for IMS TM (Implicit).”

```

*-----
SEND-ERROR-MESSAGE.
*-----
MOVE 'N'                TO SEND-DONE-SW.
MOVE TDS-ERROR-MSG      TO MSG-TYPE.
MOVE LENGTH OF MSG-RPC TO MSG-RPC-L.

*   Ensure we're in right state to send a message

CALL 'TDSTATUS' USING 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 AND
    GWL-STATUS-COMM = TDS-RECEIVE) THEN

    CALL 'TDSNDMSG' USING 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-IF.

```

Usage

- TDSTATUS 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.

See also *Related functions*

- TDRESULT on page 167
- TDSNDDON on page 206

Related topics

- “Long-running transactions” on page 54

Related documents

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

TDTERM

Description	Frees up all MVS storage.
Syntax	COPY SYGWCOB. 01 IHANDLE PIC S9(9) USAGE COMP SYNC. 01 RETCODE PIC S9(9) USAGE COMP SYNC. CALL 'TDTERM' USING IHANDLE, RETCODE.
Parameters	<i>IHANDLE</i> (I) Pointer to a transaction-wide structure that contains information used to set up the Gateway-Library environment. This must be the same <i>IHANDLE</i> specified in the program's initial TDINIT call. It corresponds to the context structure in Open Client Client-Library. <i>RETCODE</i> (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 <i>RETCODE</i> argument can contain any of the return values listed in Table 3-44.

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 <i>IHANDLE</i> 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 in a Gateway-Library transaction using the IMS TM implicit API. The transaction processes multiple client requests, using TDGETREQ to call each request after the first. This example is taken from the sample program in Appendix D, “Sample RPC Application for IMS TM (Implicit).”
----------	--

```

*-----
SEND-ROWS.
*-----
    PERFORM FETCH-AND-SEND-ROWS
        UNTIL ALL-DONE.
    FINISH-REPLY.
    CALL 'TDSNDDON' USING GWL-PROC, GWL-RC,
                        WRK-DONE-STATUS, CTR-ROWS,
                        TDS-ZERO, TDS-ENDRPC.
. [check return code]
.
*   Get next client request
    MOVE TDS-TRUE TO GWL-WAIT-OPTION.
    MOVE ZEROES TO GWL-REQ-TYPE.
    MOVE SPACES TO GWL-RPC-NAME.
    CALL 'TDGETREQ' USING GWL-PROC, GWL-RC, GWL-WAIT-OPTION,
                        GWL-REQ-TYPE, GWL-RPC-NAME.
    EVALUATE GWL-RC
        WHEN ZEROES
            GO TO READ-IN-USER-PARM
        WHEN TDS-RESULTS-COMPLETE
            PERFORM FREE-ALL-STORAGE
        WHEN TDS-CONNECTION-TERMINATED
            PERFORM FREE-ALL-STORAGE
        WHEN OTHER
            MOVE 'TDGETREQ' TO CALL-ERROR
            PERFORM DISPLAY-CALL-ERROR
    END-EVALUATE.
    GOBACK.
*-----
    FREE-ALL-STORAGE.
*-----
    CALL 'TDFREE' USING GWL-PROC, GWL-RC.
. [check return code]
    CALL 'TDTERM' USING GWL-INIT-HANDLE, GWL-RC.

```

Usage

- TDTERM 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 “Long-running transactions” on page 54.
- For more information, refer to your IMS TM product documentation.

See also

Related functions

- TDACCEPT on page 70
- TDGETREQ
- TDINIT on page 142
- TDSETPT on page 193

Related topics

- “Long-running transactions” on page 54

Related documents

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

TDYNAMIC

Description

Reads or responds to a client dynamic SQL command.

Syntax

Copy SYGWC0B.

```
01 TDPROC          PIC S9(9) USAGE COMP SYNC.
01 RETCODE         PIC S9(9) USAGE COMP SYNC.
01 CMD             PIC S9(9) USAGE COMP SYNC.
01 ITEM           PIC S9(9) USAGE COMP SYNC.
01 HOST-VARIABLE  PIC X(n).
01 HOST-VAR-LENGTH PIC S9(9) USAGE COMP SYNC.
01 ACTUAL-DATA-LENGTH PIC S9(9) USAGE COMP SYNC.
CALL 'TDYNAMIC' USING 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 3-47.

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. *ITEM* can be one of the following values listed in Table 3-45.

Table 3-45: *ITEM* argument values

Value	Notes
TDS-DYN-TYPE	Legal values for TDS-DYN-TYPE when <i>CMD</i> is TDS-GET: TDS-PREPARE TDS-DESCRIBE-INPUT TDS-DESCRIBE-OUTPUT TDS-EXECUTE TDS-EXEC-IMMEDIATE TDS-DEALLOC When <i>CMD</i> is TDS-SET, TDS-DYN-ACK is the only valid value.
TDS-DYN-IDLEN	The length of the dynamic statement ID.
TDS-DYN-ID	The dynamic statement ID.
TDS-DYN-STMTLEN	The length of the dynamic statement.
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 listed in Table 3-46 on page 235.

Table 3-46: *HOST-VAR-LEN* values

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 3-47.

Table 3-47: 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.
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 <i>OPTION</i> 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 <i>OPTION</i> 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-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.
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 <i>OPTION</i> field of the <i>CURSOR-DESC</i> 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 <i>OPTION</i> field of the <i>CURSOR-DESC</i> 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.

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-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 <i>HOST-VAR-LEN</i> 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.
TDS-INVALID-STATUS (-174)	Invalid status value. The value entered in the <i>STATUS</i> 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.

See also

Related topics

- “Dynamic SQL support” on page 45

TDWRTLOG

Description Writes a user-created message or a system entry to the trace and error log.

Syntax COPY SYGWCOB.

 01 TDPROC PIC S9(9) USAGE COMP SYNC.

 01 RETCODE PIC S9(9) USAGE COMP SYNC.

 01 DATETIME-FLAG PIC S9(9) USAGE COMP SYNC.

```

01 MESSAGE      PIC X(x).
01 MESSAGE-LENGTH PIC S9(9) USAGE COMP SYNC.
CALL 'TDWRTLOG' USING 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 3-48 on page 239.

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 3-48.

Table 3-48: 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 <i>TDPROC</i> argument.

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 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.

```

* -----
* Determine whether tracing is on or off.
* -----
PERFORM GET-TRACE-STATUS THRU GET-TRACE-STATUS-EXIT.

* -----
* Write a log entry only if logging is enabled.
* -----
IF TRACING-ON THEN
    CALL 'TDWRTLOG' USING GWL-PROC,
                        GWL-RC,
                        TDS-TRUE,
                        GWL-WRTLOG-MSG,
                        GWL-WRTLOG-MSG-L

    IF GWL-RC NOT = TDS-OK THEN
        MOVE 'N'          TO SEND-DONE-SW
        MOVE 'TDWRTLOG' TO MSG-SRVLIB-FUNC
        GO TO TDWRTLOG-EXIT
    END-IF
ELSE
    MOVE 'N'          TO SEND-DONE-SW
    MOVE 'LOGNOTON' TO MSG-SRVLIB-FUNC
END-IF.

```

Usage

- 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:
 - *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.

- *Under IMS TM:* Header, data, and API tracing information are all written to the IMS TM system log.
- *Under MVS:* The information is written to a sequential file.
- The log must be open for this function to execute successfully.
- *Under CICS:* TDSETLOG opens the trace and error log when it turns tracing on.
- *Under IMS TM:* The IMS TM system log is always open, but TDSETLOG does a logical OPEN by turning tracing on.
- *Under MVS:* Gateway-Library opens the log, but TDSETLOG does a logical OPEN by turning tracing on.
- This function can be used to send local messages to the trace and error log even when the connection is down.
- 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 121
- TDSETLOG on page 183

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	(<i>TDPROC</i> , <i>RETCODE</i> , <i>IHANDLE</i> , <i>ACCEPT-CONNECTION-NAME</i> , <i>ERROR-SUBCODE</i>);	
TDCONVRT	(<i>TDPROC</i> , <i>RETCODE</i> , <i>NUM-DECIMAL-PLACES</i> , <i>SOURCE-TYPE</i> , <i>SOURCE-LENGTH</i> , <i>SOURCE-VARIABLE</i> , <i>RESULT-TYPE</i> , <i>RESULT-LENGTH</i> , <i>RESULT-VARIABLE</i> , <i>OUTLEN</i>);	
	<i>Note: OUTLEN is optional.</i>	
TDCURPRO	(<i>TDPROC</i> , <i>RETCODE</i> , <i>ACTION</i> , <i>CURSOR-DESC</i>);	

Function	Arguments	Symbolic constants
TDESCRIB	(<i>TDPROC</i> , <i>RETCODE</i> , <i>COLUMN-NUMBER</i> , <i>HOST-VARIABLE-TYPE</i> , <i>HOST-VARIABLE-MAXLEN</i> , <i>HOST-VARIABLE-NAME</i> , <i>NULL-INDICATOR-VARIABLE</i> , <i>NULLS-ALLOWED</i> , <i>COLUMN-TYPE</i> , <i>COLUMN-MAXLEN</i> , <i>COLUMN-NAME</i> , <i>COLUMN-NAME-LENGTH</i>);	TDS-TRUE TDS-FALSE
TDFREE	(<i>TDPROC</i> , <i>RETCODE</i>);	
TDGETREQ	(<i>TDPROC</i> , <i>RETCODE</i> , <i>WAIT-OPTION</i> , <i>REQUEST-TYPE</i> , <i>TRAN-NAME</i>);	TDS-TRUE TDS-FALSE TDS-LANGUAGE-EVENT TDS-RPC-EVENT TDS-DYNAMIC-EVENT TDS-CURSOR-EVENT
TDGETSOI	(<i>TDPROC</i> , <i>RETCODE</i> , <i>OBJECT-TYPE</i> , <i>OBJECT-NUMBER</i> , <i>STRIP-SOSI</i>);	TDS-OBJECT-COL TDS-OBJECT-PARM TDS-STRIP-SOSI TDS-BLANK-SOSI

Function	Arguments	Symbolic constants
TDGETUSR	(<i>TDPROC</i> , <i>RETCODE</i> , <i>ACCESS-CODE</i> , <i>USER-ID</i> , <i>PASSWORD</i> , <i>SERVER-NAME</i> , <i>CLIENT-CHARSET</i> , <i>NATIONAL-LANGUAGE</i> , <i>SERVER-CHARSET</i> , <i>SERVER-DBCS</i> , <i>APPNAME-ID</i>);	
TDINFACT	(<i>IHANDLE</i> , <i>RETCODE</i> , <i>ACCOUNTING-FLAG</i> , <i>ACCOUNTING-FILENAME</i> , <i>MAXNUM-ACCT-RECORDS</i>);	TDS-TRUE TDS-FALSE
TDINFBCD	(<i>TDPROC</i> , <i>RETCODE</i> , <i>OBJECT-TYPE</i> , <i>OBJECT-NUMBER</i> , <i>BCD-LENGTH</i> , <i>BCD-NUMBER-DECIMAL-PLACES</i>);	TDS-OBJECT-COL TDS-OBJECT-PARM

Function	Arguments	Symbolic constants
TDINFLOG	<i>(IHANDLE,</i>	
	<i>RETCODE,</i>	
	<i>GLOBAL-TRACE-FLAG,</i>	TDS-NO-TRACING TDS-TRACE-ALL-RPCS TDS-TRACE-SPECIFIC-RPCS TDS-TRACE-ERRORS-ONLY
	<i>API-TRACE-FLAG,</i>	TDS-TRUE TDS-FALSE
	<i>TDS-HEADER-TRACE-FLAG,</i>	TDS-TRUE TDS-FALSE
	<i>TDS-DATA-TRACE-FLAG,</i>	TDS-TRUE TDS-FALSE
	<i>TRACE-ID,</i>	
	<i>TRACE-FILENAME,</i>	
	<i>MAXNUM-TRACE-RECORDS);</i>	
	TDINFPGM	<i>(TDPROC,</i>
<i>RETCODE,</i>		
<i>TDS-VERSION,</i>		TDS-VERSION2-3 TDS-VERSION3-4 TDS-VERSION4-0 TDS-VERSION4-2 TDS-VERSION4-6 TDS-VERSION4-8 TDS-VERSION4-9 TDS-VERSION5-0
<i>LONGVAR-TRUNC-FLAG,</i>		TDS-TRUE TDS-FALSE
<i>ROW-LIMIT,</i>		
<i>REMOTE-TRACE-FLAG,</i>		TDS-TRUE TDS-FALSE
<i>USER-CORRELATOR,</i>		
<i>DB2GW-OPTIONS,</i>		
<i>DB2GW-PID,</i>		
<i>REQUEST-TYPE);</i>		TDS-LANGUAGE-EVENT TDS-RPC-EVENT TDS-CURSOR-EVENT TDS-DYNAMIC-EVENT

Function	Arguments	Symbolic constants
TDINFPRM	<i>(TDPROC,</i>	
	<i>RETCODE,</i>	
	<i>PARAM-ID,</i>	
	<i>DATATYPE,</i>	
	<i>ACTUAL-DATA-LENGTH,</i>	
	<i>MAX-DATA-LENGTH,</i>	
	<i>PARAM-STATUS,</i>	For TDS 4.6: TDS-INPUT-VALUE TDS-RETURN-VALUE For TDS 5.0: TDS-INPUT-VALUE-NULLABLE TDS-RETURN-VALUE-NULLABLE
	<i>PARAM-NAME,</i>	
	<i>PARAM-NAME-LENGTH,</i>	
TDINFRPC	<i>USER-DATATYPE);</i>	
	<i>(TDPROC,</i>	
	<i>RETCODE,</i>	
	<i>REQUEST-TYPE,</i>	TDS-LANGUAGE-EVENT TDS-RPC-EVENT TDS-CURSOR-EVENT TDS-DYNAMIC-EVENT
	<i>REC-NAME,</i>	
	<i>COMM-STATE);</i>	TDS-RESET TDS-SEND TDS-RECEIVE
TDINFSPPT	<i>(IHANDLE,</i>	
	<i>RETCODE,</i>	
	<i>TRACE-STATUS,</i>	TDS-TRUE TDS-FALSE
	<i>TRACE-OPTION,</i>	TDS-SPT-API-TRACE TDS-SPT-ERRLOG TDS-SPT-TDS-DATA
	<i>TRANSACTION-ID,</i>	
	<i>TRANSACTION-ID-LENGTH);</i>	
TDINFUDT	<i>(TDPROC,</i>	
	<i>RETCODE,</i>	
	<i>COLUMN-NUMBER,</i>	
	<i>USER-DATATYPE);</i>	

Function	Arguments	Symbolic constants
TDINIT	<i>For CICS: (DFHEIBLK, For IMS TM: (IO-PCB, RETCODE, IHANDLE);</i>	
TDLOCPRM	<i>(TDPROC, PARAM-ID, PARAM-NAME, PARAM-NAME-LENGTH);</i>	
TDLSTSPT	<i>(IHANDLE, RETCODE, TRACE-TABLE-LIST);</i>	
TDNUMPRM	<i>(TDPROC, NUMBER-OF-PARMS);</i>	
TDRCVPRM	<i>(TDPROC, RETCODE, PARAM-ID, HOST-VARIABLE, HOST-VARIABLE-TYPE, MAX-DATA-LENGTH, ACTUAL-DATA-LENGTH);</i>	
TDRCVSQL	<i>(TDPROC, RETCODE, HOST-VARIABLE, MAX-VAR-LENGTH, ACTUAL-STRING-LENGTH);</i>	
TDRESULT	<i>(TDPROC, RETCODE);</i>	
TDSETACT	<i>(IHANDLE, RETCODE, ACCOUNTING-FLAG, ACCOUNTING-FILENAME, MAXNUM-ACCT-RECORDS);</i>	TDS-TRUE TDS-FALSE

Function	Arguments	Symbolic constants	
TDSETBCD	<i>(TDPROC,</i>		
	<i>RETCODE,</i>		
	<i>OBJECT-TYPE,</i>	TDS-OBJECT-COL TDS-OBJECT-PARM	
	<i>OBJECT-NUMBER,</i>		
	<i>BCD-LENGTH,</i>		
	<i>BCD-NUMBER-DECIMAL-PLACES);</i>		
TDSETLEN	<i>(TDPROC,</i>		
	<i>RETCODE,</i>		
	<i>COLUMN-NUMBER,</i>		
	<i>NEW-COLUMN-LENGTH);</i>		
TDSETLOG	<i>(HANDLE,</i>		
	<i>RETCODE,</i>		
	<i>GLOBAL-TRACE-FLAG,</i>	TDS-NO-TRACING TDS-TRACE-ALL-RPCS TDS-TRACE-SPECIFIC-RPCS TDS-TRACE-ERRORS-ONLY	
	<i>API-TRACE-FLAG,</i>	TDS-TRUE TDS-FALSE	
	<i>TDS-HEADER-TRACE-FLAG,</i>	TDS-TRUE TDS-FALSE	
	<i>TDS-DATA-TRACE-FLAG,</i>	TDS-TRUE TDS-FALSE	
	<i>TRACE-ID</i>		
	<i>TRACE-FILENAME</i>		
	<i>MAXNUM-TRACE-RECORDS);</i>		
	TDSETPRM	<i>(TDPROC,</i>	
		<i>RETCODE,</i>	
<i>PARAM-ID,</i>			
<i>HOST-VARIABLE-TYPE,</i>			
<i>HOST-VARIABLE-LENGTH,</i>			
<i>HOST-VARIABLE,</i>			
<i>USER-DATATYPE);</i>			

Function	Arguments	Symbolic constants
TDSETPT	<i>(IHANDLE,</i>	
	<i>RETCODE,</i>	
	<i>PROG-TYPE,</i>	MPP BMP CONV EXPL
	<i>SPA,</i>	
	<i>RESERVED1,</i>	
TDSETSOI	<i>RESERVED2);</i>	
	<i>(TDPROC,</i>	
	<i>RETCODE,</i>	
	<i>OBJECT-TYPE,</i>	TDS-OBJECT-COLUMN TDS-OBJECT-PARAMETER
	<i>OBJECT-NUMBER,</i>	
TDSETSPT	<i>STRIP-SOSI);</i>	TDS-STRIP-SOSI TDS-BLANK-SOSI
	<i>(IHANDLE,</i>	
	<i>RETCODE,</i>	
	<i>TRACE-STATUS,</i>	TDS-TRUE TDS-FALSE
	<i>TRACE-OPTIONS,</i>	TDS-SPT-API-TRACE TDS-SPT-ERRLOG TDS-SPT-TDS-DATA
TDSETUDT	<i>TRANSACTION-ID,</i>	
	<i>TRANSACTION-ID-LENGTH);</i>	
	<i>(TDPROC,</i>	
	<i>RETCODE,</i>	
	<i>COLUMN-NUMBER,</i>	
	<i>USER-DATATYPE);</i>	

Function	Arguments	Symbolic constants
TDSNDDON	<i>(TDPROC,</i>	
	<i>RETCODE,</i>	
	<i>STATUS,</i>	TDS-DONE-FINAL TDS-DONE-CONTINUE TDS-DONE-ERROR TDS-DONE-COUNT
	<i>ROW-COUNT,</i>	
	<i>RETURN-STATUS-NUMBER,</i>	
	<i>CONN-OPTIONS);</i>	TDS-ENDREPLY TDS-ENDRPC TDS-FLUSH
TDSNDMSG	<i>(TDPROC,</i>	
	<i>RETCODE,</i>	
	<i>MESSAGE-TYPE,</i>	TDS-INFO-MSG TDS-ERROR-MSG
	<i>MESSAGE-NUMBER,</i>	
	<i>SEVERITY,</i>	TDS-INFO-SEV TDS-ERROR-SEV
	<i>ERROR-STATE,</i>	
	<i>LINE-ID,</i>	
	<i>TRANSACTION-ID,</i>	
	<i>TRANSACTION-ID-LENGTH,</i>	
	<i>MESSAGE-TEXT,</i>	
<i>MESSAGE-LENGTH);</i>		
TDSNDROW	<i>(TDPROC,</i> <i>RETCODE);</i>	
TDSQLEN	<i>(TDPROC,</i> <i>SQL-LENGTH);</i>	

Function	Arguments	Symbolic constants
TDSTATUS	<i>(TDPROC,</i> <i>RETCODE,</i> <i>RETURN-STATUS-NUMBER,</i> <i>DONE-STATUS,</i> <i>DONE-COUNT,</i> <i>COMM-STATE,</i> <i>COMM-RETCODE,</i> <i>COMM-ERROR-SUBCODE);</i>	TDS-DONE-FINAL TDS-DONE-CONTINUE TDS-DONE-ERROR TDS-DONE-COUNT TDS-RESET TDS-SEND TDS-RECEIVE
TDTERM	<i>(IHANDLE,</i> <i>RETCODE);</i>	
TDYNAMIC	<i>TDPROC,</i> <i>RETCODE,</i> <i>CMD,</i> <i>ITEM,</i> <i>HOST-VARIABLE,</i> <i>HOST-VAR-LENGTH,</i> <i>ACTUAL-DATA-LENGTH</i>	TDS-DYN-TYPE TDS-DYN-IDLEN TDS-DYN-ID TDS-DYN-STMTLEN TDS-DYN-STMT
TDWRTLOG	<i>(TDPROC,</i> <i>RETCODE,</i> <i>DATETIME-FLAG,</i> <i>MESSAGE,</i> <i>MESSAGE-LENGTH);</i>	TDS-TRUE TDS-FALSE

Sample RPC Application for CICS

This appendix contains five Open ServerConnect application programs:

- “Sample program SYCCSAR2” on page 254
A sample application that processes a LAN-side RPC from the Open Client DB-Library
- “Sample program SYCCSAU2” on page 268
A sample cursor application
- “Sample program SYCCSAW2” on page 277
A sample application that receives parameters up to 55 bytes in length and echoes them back in 55 byte rows
- “Sample program SYCCSAY2” on page 287
A sample application that receives one of two keywords, @ERRORMSG or @WARNMSG and other keywords, and then replies with the keywords and data
- “Sample program SYCCSAZ2” on page 298
A sample application that receives a text input string (10,000 bytes) and returns it in a 50-byte column, one row at a time

The purpose of these sample programs 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 programs do limited error checking.

Sample program SYCCSAR2

This sample program, SYCCSAR2, processes a LAN-side client RPC, syr2.c, from the Open Client DB-Library program. syr2.c is included on the TRS CD or tape.

The SYCCSAR2 sample program is provided as part of the Open ServerConnect package. It references a table, SYBASE.SAMPLETB, which you create from the file SYOSCREA provided with Open ServerConnect in the CTRL library.

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. One of the return parameters indicates how many rows are affected.

After each row is sent, this program examines the TDSNDROW return code. If a cancel request is received, it stops sending rows.

If the program completes successfully, it sends a confirmation message to the client; otherwise, it sends an error message.

This program demonstrates the use of the following Gateway-Library functions:

Table B-1: List of functions used in SYCCSAR2

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 <i>TDPROC</i> structure for the connection.
TDINFB CD	Get packed decimal information for a described column.
TDINFPRM	Get information about one RPC parameter.
TDINFUDT	Get a column's user-defined datatype.
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.
TDSETBCD	Set scaling for a described column.
TDSETPRM	Set one return parameter.
TDSETUDT	Set a column's user datatype.
TDSDDON	Send results-completion to client.

Name	Action
TDSNDMSG	Send message to client.
TDSNDROW	Send row to client.
TDSTATUS	Get status information.

```

*@(#) syccsar2.cobol 1.1 3/17/98          */
  IDENTIFICATION DIVISION.
  PROGRAM-ID. SYCCSAR2.
***** SYCCSAR2 - RPC REQUEST APPLICATION - COBOL2 - CICS *****
*
*   TRANID:          SYR2
*   PROGRAM:         SYCCSAR2
*   PLAN NAME:       SYR2PLAN
*   FILES:           n/a
*   TABLES:         SYBASE.SAMPLETB
*
*   This program is executed via a client RPC request from sample
*   dblib program 'SYR2'.  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
*   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
*   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:
*   4/90      - Created, MPM
*   10/93     - Added SAMPLETB DCLGEN, some restructuring, TC
*
*-----*

ENVIRONMENT DIVISION.
DATA DIVISION.
*****
WORKING-STORAGE SECTION.
*****

*-----*
*   DB2 SQLCA
*-----*
*   EXEC SQL INCLUDE SQLCA END-EXEC.
*-----*
*   SYBASE.SAMPLETB Table Declaration
*-----*
*   EXEC SQL INCLUDE SYCCSMPT END-EXEC.
*-----*
*   SERVER LIBRARY COBOL COPY BOOK
*-----*
*   COPY SYGWCOP.
*-----*
*   WORK AREAS
*-----*
01  GW-LIB-MISC-FIELDS.
    05  GWL-PROC                POINTER.
    05  GWL-INIT-HANDLE        POINTER.
    05  GWL-RC                 PIC S9(9) COMP.
    05  GWL-INFPRM-ID          PIC S9(9) COMP.
    05  GWL-INFPRM-TYPE        PIC S9(9) COMP.
    05  GWL-INFPRM-DATA-L      PIC S9(9) COMP.
    05  GWL-INFPRM-MAX-DATA-L  PIC S9(9) COMP.

```

```

05  GWL-INFPRM-STATUS      PIC S9(9) COMP.
05  GWL-INFPRM-NAME       PIC X(30) .
05  GWL-INFPRM-NAME-L     PIC S9(9) COMP.
05  GWL-INFPRM-USER-DATA  PIC S9(9) COMP.
05  GWL-INFUDD-USER-TYPE  PIC S9(9) COMP.
05  GWL-STATUS-NR        PIC S9(9) COMP.
05  GWL-STATUS-DONE      PIC S9(9) COMP.
05  GWL-STATUS-COUNT     PIC S9(9) COMP.
05  GWL-STATUS-COMM      PIC S9(9) COMP.
05  GWL-STATUS-RETURN-CODE PIC S9(9) COMP.
05  GWL-STATUS-SUBCODE   PIC S9(9) COMP.
05  GWL-NUMPRM-PARMS     PIC S9(9) COMP.
05  GWL-RCVPRM-DATA-L    PIC S9(9) COMP.
05  GWL-SETPRM-ID       PIC S9(9) COMP.
05  GWL-SETPRM-TYPE     PIC S9(9) COMP.
05  GWL-SETPRM-DATA-L   PIC S9(9) COMP.
05  GWL-SETPRM-USER-DATA PIC S9(9) COMP.
05  GWL-CONVRT-SCALE     PIC S9(9) COMP VALUE 2.
05  GWL-SETBCD-SCALE     PIC S9(9) COMP VALUE 0.
05  GWL-INFBCD-LENGTH   PIC S9(9) COMP.
05  GWL-INFBCD-SCALE    PIC S9(9) COMP.

01  PARM-FIELDS.
05  PARM-DEPT.
    49  PARM-DEPT-LEN     PIC S9(4) COMP.
    49  PARM-DEPT-TEXT   PIC X(3) .
05  PARM-RETURN-ROWS    PIC S9(9) COMP VALUE 0.

01  SNA-FIELDS.
05  SNA-SUBC            PIC S9(9) COMP.
05  SNA-CONNECTION-NAME PIC X(8)  VALUE SPACES.

01  EMPLOYEE-FIELDS.
05  EMPLOYEE-FNM.
    49  EMPLOYEE-FNM-LEN  PIC S9(4) COMP.
    49  EMPLOYEE-FNM-TEXT PIC X(12) .
05  EMPLOYEE-LNM.
    49  EMPLOYEE-LNM-LEN  PIC S9(4) COMP.
    49  EMPLOYEE-LNM-TEXT PIC X(15) .
05  EMPLOYEE-ED         PIC S9(4) COMP.
05  EMPLOYEE-JC        PIC S9(3)  COMP-3.
05  EMPLOYEE-SAL       PIC S9(6)V9(2) COMP-3.

01  EMPLOYEE-FIELDS-CHAR REDEFINES EMPLOYEE-FIELDS.
05  FILLER              PIC X(16) .
05  EMPLOYEE-LNM-CHARS OCCURS 15 TIMES

```

```

                                PIC X.
05  FILLER                       PIC X(9) .

01  COLUMN-NAME-FIELDS .
05  CN-FNM                       PIC X(10) VALUE 'FIRST_NAME' .
05  CN-LNM                       PIC X(9)  VALUE 'LAST_NAME' .
05  CN-ED                        PIC X(9)  VALUE 'EDUCATION' .
05  CN-JC                        PIC X(7)  VALUE 'JOBCODE' .
05  CN-SAL                       PIC X(6)  VALUE 'SALARY' .

01  DESCRIBE-BIND-FIELDS .
05  DB-HOST-TYPE                 PIC S9(9) COMP .
05  DB-CLIENT-TYPE              PIC S9(9) COMP .
05  DB-DESCRIBE-HV-PTR          POINTER .
05  DB-COLUMN-NAME-HV-PTR       POINTER .
05  DB-NULL-INDICATOR           PIC S9(4) COMP VALUE 0 .

01  TDGETUSR-FIELDS .
05  GU-ACCESS-CODE              PIC X(32) .
05  GU-USER-ID                  PIC X(32) .
05  GU-PASSWORD                  PIC X(32) .
05  GU-SERVER-NAME              PIC X(32) .
05  GU-CLIENT-CHARSET           PIC X(32) .
05  GU-NATIONAL-LANG            PIC X(32) .
05  GU-SERVER-CHARSET           PIC X(32) .
05  GU-SERVER-DBCS              PIC X(32) .
05  GU-APP-ID                   PIC X(32) .

01  COUNTER-FIELDS .
05  CTR-COLUMN                  PIC S9(9) COMP VALUE 0 .

01  WORK-FIELDS .
05  WRKLEN1                     PIC S9(9) COMP .
05  WRKLEN2                     PIC S9(9) COMP .
05  WRK-BLANKS-SS               PIC S9(9) COMP .
05  WRK-DONE-STATUS             PIC S9(9) COMP .
05  WRK-EMPLOYEE-SAL            PIC X(8) .

01  MESSAGE-FIELDS .
05  MSG-TYPE                    PIC S9(9) COMP .
05  MSG-SEVERITY                PIC S9(9) COMP .
05  MSG-SEVERITY-OK             PIC S9(9) COMP VALUE 9 .
05  MSG-SEVERITY-ERROR         PIC S9(9) COMP VALUE 11 .
05  MSG-NR                      PIC S9(9) COMP .
05  MSG-NR-OK                  PIC S9(9) COMP VALUE 1 .
05  MSG-NR-ERROR               PIC S9(9) COMP VALUE 2 .

```



```

05 MSG-RPC                PIC X(4)          VALUE 'SYR2'.
05 MSG-RPC-L              PIC S9(9) COMP.
05 MSG-TEXT               PIC X(100).
05 MSG-TEXT-L             PIC S9(9) COMP.
05 MSG-NOT-RPC            PIC X(30)
    VALUE 'SYR2 not begun via rpc request'.
05 MSG-NOT-AUTH           PIC X(19)
    VALUE 'User not authorized'.
05 MSG-WRONG-NR-PARMS     PIC X(30)
    VALUE 'Number of parameters was not 2'.
05 MSG-NOT-RETURN-PARM    PIC X(42)
    VALUE 'First parameter must be a RETURN parameter'.
05 MSG-NOT-CHAR-PARM      PIC X(41)
    VALUE 'Second parameter must be a CHARACTER type'.
05 MSG-BAD-CURSOR         PIC X(27)
    VALUE 'ERROR - can not open cursor'.
05 MSG-BAD-FETCH          PIC X(24)
    VALUE 'ERROR - fetch row failed'.
05 MSG-SQL-ERROR.
    10 FILLER              PIC X(10) VALUE 'Sqlcode = '.
    10 MSG-SQL-ERROR-C     PIC -9(3) DISPLAY.
    10 FILLER              PIC X(16)
        VALUE ', Error Tokens: '.
    10 MSG-SQL-ERROR-K     PIC X(70).
    10 MSG-SQL-ERROR-K-CHARS
        REDEFINES MSG-SQL-ERROR-K
        OCCURS 70 TIMES
        PIC X.
05 MSG-SQL-ERROR-SS      PIC S9(4) COMP.

01 CICS-FIELDS.
05 CICS-RESPONSE         PIC S9(9) COMP.

01 SWITCHES.
05 ALL-DONE-SW           PIC X          VALUE 'N'.
    88 NOT-ALL-DONE      VALUE 'N'.
    88 ALL-DONE          VALUE 'Y'.
05 SEND-DONE-SW          PIC X          VALUE 'Y'.
    88 SEND-DONE-ERROR   VALUE 'N'.
    88 SEND-DONE-OK      VALUE 'Y'.

* -----
*   DECLARE CURSOR
* -----
EXEC SQL
    DECLARE ECURSOR CURSOR
        FOR SELECT FIRSTNME, LASTNAME,

```

```

                EDUCLVL, JOBCODE, SALARY
FROM SYBASE.SAMPLETB
WHERE WORKDEPT = :PARM-DEPT
END-EXEC.
*****
LINKAGE SECTION.
*****
01 LK-DESCRIBE-HV          PIC X(255).
01 LK-COLUMN-NAME-HV      PIC X(30).
*****
PROCEDURE DIVISION.
*****

*   Reset DB2 error handlers

EXEC SQL WHENEVER SQLWARNING CONTINUE END-EXEC.
EXEC SQL WHENEVER SQLERROR   CONTINUE END-EXEC.
EXEC SQL WHENEVER NOT FOUND  CONTINUE END-EXEC.

*   Establish gateway environment

CALL 'TDINIT' USING DFHEIBLK, GWL-RC, GWL-INIT-HANDLE.

*   Accept client request

CALL 'TDACCEPT' USING GWL-PROC, GWL-RC, GWL-INIT-HANDLE,
                    SNA-CONNECTION-NAME, SNA-SUBC.

*   TDRESULT to make sure we were started via RPC request

CALL 'TDRESULT' USING GWL-PROC, GWL-RC.

IF GWL-RC NOT = TDS-PARM-PRESENT THEN
    PERFORM TDRESULT-ERROR
    GO TO END-PROGRAM
END-IF.

*   Verify user login information

MOVE 'TOP SECRET' TO GU-ACCESS-CODE.

CALL 'TDGETUSR' USING 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 NOT = TDS-OK THEN
    PERFORM TDGETUSR-ERROR
    GO TO END-PROGRAM
END-IF.

*   Get number of parameters ... should be two

CALL 'TDNUMPRM' USING GWL-PROC, GWL-NUMPRM-PARMS.

IF GWL-NUMPRM-PARMS NOT = 2 THEN
    PERFORM TDNUMPRM-ERROR
    GO TO END-PROGRAM
END-IF.

*   Get return parameter information

MOVE 1 TO GWL-INFPRM-ID.
PERFORM GET-PARM-INFO.

(IF GWL-INFPRM-STATUS NOT = TDS-RETURN-VALUE AND
IF GWL-INFPRM-STATUS NOT = TDS-RETURN-VALUE-NULLABLE) THEN
    PERFORM TDINFPRM-NOT-RETURN-PARM-ERROR
    GO TO END-PROGRAM
END-IF.

MOVE GWL-INFPRM-USER-DATA TO GWL-SETPRM-USER-DATA.
MOVE GWL-INFPRM-ID        TO GWL-SETPRM-ID.
MOVE GWL-INFPRM-DATA-L    TO GWL-SETPRM-DATA-L.
MOVE GWL-INFPRM-TYPE      TO GWL-SETPRM-TYPE.

*   Get department id parameter number from known name

MOVE '@parm2' TO GWL-INFPRM-NAME.
MOVE 6        TO GWL-INFPRM-NAME-L.

CALL 'TDLOCPRM' USING GWL-PROC, GWL-INFPRM-ID,
                    GWL-INFPRM-NAME, GWL-INFPRM-NAME-L.

*   Get department parameter information

PERFORM GET-PARM-INFO.

IF GWL-INFPRM-TYPE NOT = TDSVARYCHAR THEN
    PERFORM TDINFPRM-NOT-CHAR-PARM-ERROR
    GO TO END-PROGRAM
END-IF.

```

* Get department parameter data

```
CALL 'TDRCVPRM' USING GWL-PROC, GWL-RC, GWL-INFPRM-ID,
                    PARM-DEPT, GWL-INFPRM-TYPE,
                    GWL-INFPRM-MAX-DATA-L,
                    GWL-RCVPRM-DATA-L.
```

* Open the DB2 cursor for fetch

```
EXEC SQL OPEN ECURSOR END-EXEC.
```

```
IF SQLCODE NOT = 0
    PERFORM OPEN-ERROR
    GO TO END-PROGRAM
END-IF.
```

* The SYGETAD assembler subroutine returns the address of any
* data item in parameter two into parameter 1. It's a way to
* get around the limitations of the COBOL 2 SET verb.

```
CALL 'SYGETAD' USING DB-DESCRIBE-HV-PTR, EMPLOYEE-FNM.
```

* During 'DESCRIBE-COLUMN', LK-DESCRIBE-HV will be based on
* DB-DESCRIBE-HV-PTR, which addresses EMPLOYEE-FNM. This
* allows us to call a 'generic' TDESCRIB, using LK-DESCRIBE-HV
* as a constant in the call, even though it actually varies
* depending on the SYGETAD and SET sequence preceding it.

* The same technique will be used for other data items which
* must be passed by address; for example, the name of the
* columns.

```
CALL 'SYGETAD' USING DB-COLUMN-NAME-HV-PTR, CN-FNM.
MOVE LENGTH OF EMPLOYEE-FNM-TEXT TO WRKLEN1.
MOVE LENGTH OF CN-FNM           TO WRKLEN2.
MOVE TDSVARYCHAR                TO DB-HOST-TYPE.
MOVE TDSVARYCHAR                TO DB-CLIENT-TYPE.
PERFORM DESCRIBE-COLUMN.
```

* Here we let TDESCRIB convert from DB2 varchar (TDSVARYCHAR)
* to DBCHAR.

```
CALL 'SYGETAD' USING DB-DESCRIBE-HV-PTR, EMPLOYEE-LNM.
CALL 'SYGETAD' USING DB-COLUMN-NAME-HV-PTR, CN-LNM.
MOVE LENGTH OF EMPLOYEE-LNM-TEXT TO WRKLEN1.
```

```

MOVE LENGTH OF CN-LNM          TO WRKLEN2.
MOVE TDSVARYCHAR              TO DB-HOST-TYPE.
MOVE TDSCHAR                  TO DB-CLIENT-TYPE.
PERFORM DESCRIBE-COLUMN.

CALL 'SYGETAD' USING DB-DESCRIBE-HV-PTR, EMPLOYEE-ED.
CALL 'SYGETAD' USING DB-COLUMN-NAME-HV-PTR, CN-ED.
MOVE LENGTH OF EMPLOYEE-ED TO WRKLEN1.
MOVE LENGTH OF CN-ED       TO WRKLEN2.
MOVE TDSINT2               TO DB-HOST-TYPE.
MOVE TDSINT2               TO DB-CLIENT-TYPE.
PERFORM DESCRIBE-COLUMN.

*   Get the user defined datatype of EMPLOYEE-ED column.

CALL 'TDINFUDT' USING GWL-PROC, GWL-RC, CTR-COLUMN,
                    GWL-INFUDT-USER-TYPE.

*   Set the user defined datatype of EMPLOYEE-ED column.

CALL 'TDSETUDT' USING GWL-PROC, GWL-RC, CTR-COLUMN,
                    GWL-INFUDT-USER-TYPE.

*   Here we let TDESCRIB convert from TDSDECIMAL to TDSFLT8.

CALL 'SYGETAD' USING DB-DESCRIBE-HV-PTR, EMPLOYEE-JC.
CALL 'SYGETAD' USING DB-COLUMN-NAME-HV-PTR, CN-JC.
MOVE LENGTH OF EMPLOYEE-JC TO WRKLEN1.
MOVE LENGTH OF CN-JC       TO WRKLEN2.
MOVE TDSDECIMAL            TO DB-HOST-TYPE.
MOVE TDSFLT8               TO DB-CLIENT-TYPE.
PERFORM DESCRIBE-COLUMN.

*   We must inform the Server Library how many decimal places
*   are in the EMPLOYEE-JC column.

CALL 'TDSETBCD' USING GWL-PROC, GWL-RC, TDS-OBJECT-COL,
                    CTR-COLUMN, TDS-DEFAULT-LENGTH,
                    GWL-SETBCD-SCALE.

*   Demonstrate getting decimal column information.

CALL 'TDINFBCD' USING 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.

```
CALL 'SYGETAD' USING DB-DESCRIBE-HV-PTR, WRK-EMPLOYEE-SAL.  
CALL 'SYGETAD' USING DB-COLUMN-NAME-HV-PTR, CN-SAL.  
MOVE LENGTH OF WRK-EMPLOYEE-SAL TO WRKLEN1.  
MOVE LENGTH OF CN-SAL           TO WRKLEN2.  
MOVE TDSMONEY                   TO DB-HOST-TYPE.  
MOVE TDSMONEY                   TO DB-CLIENT-TYPE.  
PERFORM DESCRIBE-COLUMN.
```

```
PERFORM FETCH-AND-SEND-ROWS  
      UNTIL ALL-DONE.
```

* Close cursor

```
EXEC SQL CLOSE ECURSOR END-EXEC.
```

* Update returned parameter with number of rows fetched

```
CALL 'TDSETPRM' USING 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.

*-----
EXEC SQL FETCH ECURSOR INTO :EMPLOYEE-FIELDS END-EXEC.

```
IF SQLCODE = 0 THEN
```

* Convert from DB2 decimal (TDSDECIMAL) to dblib MONEY.

```
MOVE LENGTH OF EMPLOYEE-SAL      TO WRKLEN1  
MOVE LENGTH OF WRK-EMPLOYEE-SAL TO WRKLEN2
```

```
CALL 'TDCONVRT' USING GWL-PROC, GWL-RC,  
                  GWL-CONVRT-SCALE, TDSDECIMAL,  
                  WRKLEN1, EMPLOYEE-SAL, TDSMONEY,  
                  WRKLEN2, WRK-EMPLOYEE-SAL
```

* send a row to the client

```

CALL 'TDSNDROW' USING GWL-PROC, GWL-RC
ADD 1 TO PARM-RETURN-ROWS

IF GWL-RC = TDS-CANCEL-RECEIVED THEN
    MOVE 'Y' TO ALL-DONE-SW
END-IF

ELSE IF SQLCODE = +100 THEN
    MOVE 'Y' TO ALL-DONE-SW

ELSE
    MOVE 'Y' TO ALL-DONE-SW
    PERFORM FETCH-ERROR
END-IF.

*-----
GET-PARM-INFO.
*-----

CALL 'TDINFPRM' USING 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.

*-----
DESCRIBE-COLUMN.
*-----

SET ADDRESS OF LK-DESCRIBE-HV      TO DB-DESCRIBE-HV-PTR.
SET ADDRESS OF LK-COLUMN-NAME-HV  TO DB-COLUMN-NAME-HV-PTR.
ADD 1                             TO CTR-COLUMN.

CALL 'TDESCRIB' USING GWL-PROC, GWL-RC, CTR-COLUMN,
                    DB-HOST-TYPE, WRKLEN1, LK-DESCRIBE-HV,
                    DB-NUL-INDICATOR, TDS-FALSE,
                    DB-CLIENT-TYPE, WRKLEN1,
                    LK-COLUMN-NAME-HV, WRKLEN2.

*-----
TDGETUSR-ERROR.
*-----

MOVE MSG-NOT-AUTH TO MSG-TEXT.
MOVE LENGTH OF MSG-NOT-AUTH TO MSG-TEXT-L.
PERFORM SEND-ERROR-MESSAGE.

*-----
TDRESULT-ERROR.
*-----

MOVE MSG-NOT-RPC                TO MSG-TEXT.

```

```
MOVE LENGTH OF MSG-NOT-RPC TO MSG-TEXT-L.
PERFORM SEND-ERROR-MESSAGE.
*-----
TDNUMPRM-ERROR.
*-----
MOVE MSG-WRONG-NR-PARMS          TO MSG-TEXT.
MOVE LENGTH OF MSG-WRONG-NR-PARMS TO MSG-TEXT-L.
PERFORM SEND-ERROR-MESSAGE.

*-----
TDINFPRM-NOT-RETURN-PARM-ERROR.
*-----
MOVE MSG-NOT-RETURN-PARM          TO MSG-TEXT.
MOVE LENGTH OF MSG-NOT-RETURN-PARM TO MSG-TEXT-L.
PERFORM SEND-ERROR-MESSAGE.

*-----
TDINFPRM-NOT-CHAR-PARM-ERROR.
*-----
MOVE MSG-NOT-CHAR-PARM           TO MSG-TEXT.
MOVE LENGTH OF MSG-NOT-CHAR-PARM TO MSG-TEXT-L.
PERFORM SEND-ERROR-MESSAGE.

*-----
OPEN-ERROR.
*-----
MOVE MSG-BAD-CURSOR              TO MSG-TEXT.
MOVE LENGTH OF MSG-BAD-CURSOR TO MSG-TEXT-L.
PERFORM SEND-ERROR-MESSAGE.
PERFORM SEND-SQL-ERROR.

*-----
FETCH-ERROR.
*-----
MOVE MSG-BAD-FETCH               TO MSG-TEXT.
MOVE LENGTH OF MSG-BAD-FETCH TO MSG-TEXT-L.
PERFORM SEND-ERROR-MESSAGE.
PERFORM SEND-SQL-ERROR.

*-----
SEND-SQL-ERROR.
*-----
MOVE SQLCODE TO MSG-SQL-ERROR-C.
MOVE SQLERRMC TO MSG-SQL-ERROR-K.
*-----
* ensure possible non-printables translated to spaces
*-----
PERFORM VARYING MSG-SQL-ERROR-SS FROM 1 BY 1
      UNTIL MSG-SQL-ERROR-SS > SQLERRML
```



```

        IF MSG-SQL-ERROR-K-CHARS (MSG-SQL-ERROR-SS) < SPACE
        OR MSG-SQL-ERROR-K-CHARS (MSG-SQL-ERROR-SS) > '9' THEN
            MOVE SPACE TO MSG-SQL-ERROR-K-CHARS (MSG-SQL-ERROR-SS)
        END-IF

    END-PERFORM.

    MOVE MSG-SQL-ERROR           TO MSG-TEXT.
    MOVE LENGTH OF MSG-SQL-ERROR TO MSG-TEXT-L.
    PERFORM SEND-ERROR-MESSAGE.

* -----
SEND-ERROR-MESSAGE.
* -----
    MOVE 'N'                     TO SEND-DONE-SW.
    MOVE MSG-SEVERITY-ERROR TO MSG-SEVERITY.
    MOVE MSG-NR-ERROR        TO MSG-NR.
    MOVE TDS-ERROR-MSG       TO MSG-TYPE.
    PERFORM SEND-MESSAGE.

* -----
SEND-MESSAGE.
* -----
    MOVE LENGTH OF MSG-RPC TO MSG-RPC-L.

*   Ensure we're in right state to send a message

    CALL 'TDSTATUS' USING 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 AND
        GWL-STATUS-COMM = TDS-RECEIVE) THEN

        CALL 'TDSNDMSG' USING 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-IF.

* -----
END-PROGRAM.
* -----
    IF SEND-DONE-OK
        MOVE TDS-DONE-COUNT TO WRK-DONE-STATUS
    ELSE
        MOVE TDS-DONE-ERROR TO WRK-DONE-STATUS

```

```
        MOVE ZERO          TO PARM-RETURN-ROWS
END-IF.

CALL 'TDSNDDON' USING GWL-PROC, GWL-RC, WRK-DONE-STATUS,
        PARM-RETURN-ROWS, TDS-ZERO,
        TDS-ENDRPC.

CALL 'TDFREE' USING GWL-PROC, GWL-RC.

EXEC CICS RETURN END-EXEC.
```

Sample program SYCCSAU2

The following sample program, SYCCSAU2, establishes a long-running conversational transaction which returns data to the client, then waits for client requests via the TDGETREQ interface. The purpose of this sample is to demonstrate the handling of cursor commands. This sample processes an Embedded SQL™/C Open Client RPC, syu2.c, which is included on the TRS tape. The SYCCSAU2 sample program is included on the Open ServerConnect API tape.

This sample program does not use any table, the data used by the cursor commands is hard-coded in the program.

```
*(#) syccsau2.cobol 1.1 4/26/96      */
    IDENTIFICATION DIVISION.
    PROGRAM-ID. SYCCSAU2.
***** SYCCSAU2 - SAMPLE LONG-RUNNING CURSOR transaction program ***
*
*   TRANID:      SYU2
*   PROGRAM:     SYCCSAU2
*   PLAN NAME:   n/a
*   FILES:       n/a
*   TABLES:    n/a
*
*   This program establishes a long-running "conversational"
*   transaction which returns data to the client then waits for
*   client requests via the TDGETREQ interface.
*   This version of the program is built to use the open server
*   cursor commands which are introduced on OS 3.1 and netgateways
*   3.0.1
*
*   The following Open Server Library calls are used:
```

```

*
* TDINIT          initializes the TDS environment
* TDACCEPT       accept a request from a client
* TDCURPRO       cursor processing command
* TDESCRIB       describe a column in a result row
* TDFREE         free the TDPROC structure
* TDGETREQ       get the next cursor request
* TDINFPRM       retrieve information about a RPC parameter
* TDINIT         initialize the TDS environment
* TDRCVPRM       retrieve the data from a RPC parameter
* TDRCVSQL       get SQL next
* TDRESULT       describe the next object from a client
* TDSNDDON       send result completion indication to client
* TDSNDROW       send a row of data to the requesting client
* TDNUMPRM       get number of cursor parameters
*
* Change Activity:
* 04/13 J.A.- code to handle cursor support select support
* 04/17 J.A.- added code to handle update and delete from cursor
*****
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
*-----
*      Work variables
*-----
77 COL-20          PIC S9(9) COMP VALUE +20.
77 COL2-LNG       PIC S9(9) COMP VALUE +4.
77 COL-COUNT      PIC S9(9) COMP VALUE +1.
77 COLUMN-NAME    PIC X(4)  VALUE 'COLS'.
77 COLUMN-NAME-LEN PIC S9(9) COMP VALUE +4.
* Gateway Library interface variables
77 GWL-INIT-HANDLE POINTER.
77 GWL-PROC        POINTER.
77 GWL-RC          PIC S9(9) COMP VALUE +0.
77 FILL-COUNT      PIC S9(9) COMP VALUE +0.
77 NULL-IND        PIC S9(4) COMP VALUE +0.
77 PARM-NAME       PIC X(20) .
77 PARM-NAME-LNG   PIC S9(9) COMP.
77 PARM-FILLCHAR   PIC X(1)  VALUE SPACES.
77 PARM-ID         PIC S9(9) COMP.
77 PARM-STATUS     PIC S9(9) COMP.
77 PARM-DATA-TYPE  PIC S9(9) COMP.
77 PARM-DATA-LNG   PIC S9(9) COMP.
77 PARM-LNG        PIC S9(9) COMP.

```

```

77  PARM-MAXLNG          PIC S9(9) COMP.
77  PARM-NUMROW          PIC S9(9) COMP VALUE +0.
77  PARM-UDT             PIC S9(9) COMP.
77  REQ-TYPE             PIC S9(9) COMP VALUE +0.
77  RETURN-STATUS       PIC S9(9) COMP VALUE +0.
01  ROW-DATA.
    05  ROW-CHAR         PIC X(1) VALUE SPACES
                          OCCURS 80 TIMES.
77  RPC-NAME             PIC X(4) VALUE 'TS02'.
77  RPC-NAME-LENGTH     PIC S9(9) COMP VALUE +4.
77  SNA-CONNECTION-NAME PIC X(8) VALUE SPACES.
77  SNA-SUBCODE         PIC S9(9) COMP.
77  WAIT-OPTION         PIC S9(9) COMP VALUE +0.
01  CMD                  PIC S9(9) COMP SYNC.
01  REMOTE-TRACE-FLAG   PIC S9(9) USAGE COMP SYNC.
01  TDS-VERSION         PIC S9(9) USAGE COMP.
01  LONGVAR-TRUNC-FLAG PIC S9(9) USAGE COMP.
01  ROW-LIMIT           PIC S9(9) USAGE COMP.
01  USER-CORRELATOR    PIC S9(9) USAGE COMP.
01  DB2GW-OPTIONS      PIC S9(9) USAGE COMP.
01  DB2GW-PID          PIC X(1).
77  ERR-MSG             PIC X(40) VALUE IS SPACES.
77  ERR-MSG-LEN        PIC S9(9) USAGE COMP VALUE IS 40.
01  NO-OF-ROWS         PIC S9(9) USAGE COMP VALUE IS 0.
01  ROWS-TOTAL         PIC S9(9) USAGE COMP VALUE IS 0.
01  SEND-STATUS        PIC S9(9) USAGE COMP SYNC.
01  STATUS-NUMBER      PIC S9(9) USAGE COMP SYNC.
01  OPEN-COUNT         PIC S9(9) USAGE COMP VALUE IS 0.
01  SAVE-CURSOR-ID     PIC S9(9) USAGE COMP SYNC.
01  SQLSTR             PIC X(300) VALUE IS SPACES.
01  MAX-SQL-LENGTH     PIC S9(9) USAGE COMP VALUE IS 300.
01  ACT-SQL-LENGTH     PIC S9(9) USAGE COMP.
01  COL1-DATA          PIC X(20) VALUE IS SPACES.
01  COL2-DATA          PIC S9(9) USAGE COMP.
01  UPDATES-THIS-CURSOR PIC S9(9) USAGE COMP.
01  DELETES-THIS-CURSOR PIC S9(9) USAGE COMP.

```

```

*-----*
*  Server library COBOL copybook
*-----*
*  COPY SYGWC0B.
*-----*
*  Procedure division.
*-----*
PROCEDURE DIVISION.
*  Initialize TDS environment
  CALL 'TDINIT' USING DFHEIBLK, GWL-RC, GWL-INIT-HANDLE.

```

```

*   Accept client request
      CALL 'TDACCEPT' USING GWL-PROC, GWL-RC, GWL-INIT-HANDLE,
                          SNA-CONNECTION-NAME, SNA-SUBCODE.
*   If no parameters set 20 rows and wait for cursor command
      CALL 'TDRESULT' USING GWL-PROC, GWL-RC.
      IF GWL-RC NOT EQUAL TDS-PARM-PRESENT THEN
          MOVE 20 TO PARM-NUMROW
          CALL 'TDSNDDON' USING GWL-PROC, GWL-RC, TDS-DONE-FINAL,
                          PARM-NUMROW, TDS-ZERO, TDS-ENDREPLY
          GO TO NEXT-STEP.
*   Read in user parameters, and process the request
*   only parameter for now is number of rows requested
*   Get info for RPC parameter 1 - number rows
      MOVE 1 TO PARM-ID.
      CALL 'TDINFPRM' USING GWL-PROC, GWL-RC, PARM-ID,
                          PARM-DATA-TYPE, PARM-LNG, PARM-MAXLNG,
                          PARM-STATUS, PARM-NAME, PARM-NAME-LNG,
                          PARM-UDT.
*   Initialize the cursor-id , 111 is not significant number
      MOVE 111 TO SAVE-CURSOR-ID.
*   Get number of row to return from RPC parameter 2
*   if parmeter is not entered then return 20 rows
      IF GWL-RC = 0
          CALL 'TDRCVPRM' USING GWL-PROC, GWL-RC, PARM-ID,
                          PARM-NUMROW, TDSINT4, PARM-MAXLNG, PARM-LNG
      ELSE
          STRING 'PARM 1 SHOULD BE INT4'
          DELIMITED BY SIZE INTO ERR-MSG
          PERFORM SEND-ERROR.
*   we are assuming client program just starts long running rpc
*   used with the cursor support we are adding in 3.1
      CALL 'TDSNDDON' USING GWL-PROC, GWL-RC, TDS-DONE-FINAL,
                          PARM-NUMROW, TDS-ZERO, TDS-ENDREPLY.
*   Wait for the next request from the client
      PERFORM NEXT-STEP.
GOT-REQ.
      CALL 'TDINFPGM' USING GWL-PROC GWL-RC TDS-VERSION
                          LONGVAR-TRUNC-FLAG ROW-LIMIT
                          REMOTE-TRACE-FLAG USER-CORRELATOR
                          DB2GW-OPTIONS DB2GW-PID REQ-TYPE.
*   make sure we are getting a cursor command from the client
      IF REQ-TYPE NOT EQUAL TDS-CURSOR-EVENT
          STRING 'REQ-TYPE NOT EQUAL TDS-CURSOR-EVENT'
          DELIMITED BY SIZE INTO ERR-MSG
          PERFORM SEND-ERROR.
*   look at the incoming request and perform necessary action

```

```
* in this simple example we just handle the client cursor requests
MOVE TDS-GET TO CMD.
CALL 'TDCURPRO' USING GWL-PROC, GWL-RC,
      CMD, CURSOR-DESC.
IF GWL-RC NOT EQUAL TDS-OK
  STRING 'TDCURPRO GET FAILED' DELIMITED BY SIZE
  INTO ERR-MSG
  PERFORM SEND-ERROR.
EVALUATE CURSOR-COMMAND
WHEN TDS-CURSOR-DECLARE
  PERFORM DECLARE-LOGIC
WHEN TDS-CURSOR-INFO
  PERFORM INFO-LOGIC
WHEN TDS-CURSOR-OPENCMD
  PERFORM OPEN-LOGIC
WHEN TDS-CURSOR-FETCH
  PERFORM FETCH-LOGIC
WHEN TDS-CURSOR-UPDATE
  PERFORM UPDATE-LOGIC
WHEN TDS-CURSOR-DELETE
  PERFORM DELETE-LOGIC
WHEN TDS-CURSOR-CLOSE
  PERFORM CLOSE-LOGIC
WHEN TDS-CURSOR-DEALLOC
* not a lot of meaning here as in server it frees structures
* we will never require the client to deallocate
* in a very large application this might be necessary
  PERFORM DEALLOC-LOGIC
  WHEN OTHER
    STRING 'TDCURPRO GOT UNEXPECTED CMD REQUEST'
    DELIMITED BY SIZE INTO ERR-MSG
    PERFORM SEND-ERROR
  END-EVALUATE.
  PERFORM NEXT-STEP.
NEXT-STEP.
  PERFORM DO-GET-REQ.
  PERFORM GOT-REQ.
DO-GET-REQ.
  CALL 'TDGETREQ' USING GWL-PROC GWL-RC WAIT-OPTION
      REQ-TYPE RPC-NAME.
  IF REQ-TYPE NOT EQUAL TDS-CURSOR-EVENT
    GO TO END-OF-REQUESTS.
  DECLARE-LOGIC.
* set CURSOR-ID and CURSOR-STATUS
```

```

* increment the cursors we have used
  ADD 1 TO SAVE-CURSOR-ID.
  MOVE SAVE-CURSOR-ID TO CURSOR-ID.
  PERFORM DECLARE-VALIDATION.
  PERFORM TDSSET-CURSOR.
  PERFORM SEND-ENDREPLY-200.
DECLARE-VALIDATION.
* the cursor must be CRSLONG, initially and then CRSRESULTS
* this is a little harsh but the
* sample is only meant to work with its counter part embedded sql
* or ctlibrary sample program
* the name implies that this is a long running transaction
* it is possible to not be a long running transaction but the
* program would have to be the default language transaction at
* the mainframe server gateway, which is not very likely
* one could do much more in this validation
  IF CURSOR-NAME IS EQUAL 'CRSLONG' OR CURSOR-NAME
    IS EQUAL 'CRSRESULTS' PERFORM COMPARE-SQL.
COMPARE-SQL.
* could look at the incoming sql w/cursor and have logic if needed
* but we don't care about the sql received at all for the sample
* in a real program one could use this to pass a where clause etc
* for the logic which is to materialize the results
  CALL 'TDRCVSQL' USING GWL-PROC  GWL-RC
                        SQLSTR  MAX-SQL-LENGTH
                        ACT-SQL-LENGTH.
  IF GWL-RC NOT EQUAL TDS-OK
    STRING 'TDRCVSQL FAILED'
    DELIMITED BY SIZE
    INTO ERR-MSG
    PERFORM SEND-ERROR.
INFO-LOGIC.
* Here our assumption is that row count is set via client program
  PERFORM TDSSET-CURSOR.
  PERFORM SEND-ENDREPLY-200.
OPEN-LOGIC.
* for this sample we are going to only return 20 rows
* if no parameter specified. With real data the actual data source
* determines the number of fetches
* initialize counters for the total number of updates and deletes
* which are performed on this cursor. In this sample we will
* communicate this back to the client after the cursor is
* closed. In real applications data would be deleted or updated
* for the results cursor only 2 rows returned
  IF CURSOR-NAME IS EQUAL 'CRSLONG'
    MOVE PARM-NUMROW TO ROWS-TOTAL

```

```

        MOVE 0 TO UPDATES-THIS-CURSOR
        MOVE 0 TO DELETES-THIS-CURSOR
    ELSE
        MOVE 2 TO ROWS-TOTAL.
* describe results
*   MOVE TDS-CURSOR-OPEN TO CURSOR-STATUS
        PERFORM TDSSET-CURSOR.
* for this problem just send two columns of data
        PERFORM SEND-TWO-COLUMN.
        PERFORM SEND-OPEN.
    FETCH-LOGIC.
        PERFORM TDSSET-CURSOR.
* send fetch-count number of rows to the client
* fetch-count set in cursor descriptor block
* the following code assumes that the fetch-count is
* an integral multiple of actual data
* when the row count is zero then there is one more fetch
* which just gets the SQLCODE 100
        IF ROWS-TOTAL NOT EQUAL ZERO
            IF ROWS-TOTAL LESS THAN FETCH-COUNT
                MOVE ROWS-TOTAL TO NO-OF-ROWS
            ELSE
                MOVE FETCH-COUNT TO NO-OF-ROWS END-IF
            PERFORM SEND-ROW
            UNTIL NO-OF-ROWS = 0 OR ROWS-TOTAL = 0
            ELSE PERFORM SEND-ENDREPLY-200 END-IF.
    SEND-ROW.
* if we are using results cursor then send different data
        IF CURSOR-NAME IS EQUAL 'CRSRESULTS'
            IF ROWS-TOTAL = 2
                MOVE 'Updates last cursor ' TO COL1-DATA
                MOVE UPDATES-THIS-CURSOR   TO COL2-DATA
            ELSE
                MOVE 'Deletes last cursor ' TO COL1-DATA
                MOVE DELETES-THIS-CURSOR   TO COL2-DATA END-IF
        END-IF
* send a row of data
        CALL 'TDSNDROW' USING GWL-PROC, GWL-RC
        IF GWL-RC NOT EQUAL TDS-OK
            STRING 'TDSNDROW FAILED'
            DELIMITED BY SIZE
            INTO ERR-MSG
            PERFORM SEND-ERROR.
        SUBTRACT 1 FROM NO-OF-ROWS.
        SUBTRACT 1 FROM ROWS-TOTAL.
        IF NO-OF-ROWS = 0 OR ROWS-TOTAL = 0

```



```
PERFORM SEND-ENDREPLY-200.
UPDATE-LOGIC.
    PERFORM TDSSET-CURSOR.
    PERFORM COMPARE-SQL.
*   at this point we would look at the update sql to decide
*   what must be done, in our case just move information to
*   the column data being returned
*   looking at the text string is more than we want to do here so
*   move a couple some new data to the colums being
*   returned to show that the update was processed and add one
*   to the update counter
*
*   this doesn't appear to the client to be very accurate unless
*   the fetch-count is 1 as the fetch here is out of sync with
*   the one the application is issuing
    IF FETCH-COUNT = 1
        MOVE 'Updated coll data ' TO COL1-DATA
        MOVE 123 TO COL2-DATA END-IF
    ADD 1 TO UPDATES-THIS-CURSOR.
    PERFORM SEND-ENDREPLY-200.
DELETE-LOGIC.
*   on a delete request we have nothing to actually delete so
*   we will just update a counter to show the activity took place
    PERFORM TDSSET-CURSOR.
    ADD 1 TO DELETES-THIS-CURSOR.
    PERFORM SEND-ENDREPLY-200.
CLOSE-LOGIC.
    PERFORM TDSSET-CURSOR.
    PERFORM SEND-ENDREPLY-200.
DEALLOC-LOGIC.
    STRING 'DEALLOC NOT IMPLEMENTED'
    DELIMITED BY SIZE
    INTO ERR-MSG
    PERFORM SEND-ERROR.
    PERFORM SEND-ENDREPLY-200.
SEND-OPEN.
    PERFORM SEND-ENDREPLY-200.
TDSSET-CURSOR.
    MOVE TDS-SET TO CMD.
    CALL 'TDCURPRO' USING GWL-PROC, GWL-RC,
        CMD, CURSOR-DESC.
    IF GWL-RC NOT EQUAL TDS-OK
        STRING 'TDCURPRO SET FAILED' DELIMITED BY SIZE
        INTO ERR-MSG
        PERFORM SEND-ERROR.
    SEND-ENDREPLY-200.
```

```

MOVE TDS-DONE-FINAL TO SEND-STATUS.
ADD TDS-DONE-COUNT TO SEND-STATUS.
MOVE 200 TO STATUS-NUMBER.
CALL 'TDSNDDON' USING GWL-PROC  GWL-RC
                SEND-STATUS
                NO-OF-ROWS  STATUS-NUMBER TDS-ENDREPLY.
IF GWL-RC NOT EQUAL TDS-OK
    STRING 'TDSNDDON FAILED'
        DELIMITED BY SIZE INTO ERR-MSG
        PERFORM SEND-ERROR.
SEND-TWO-COLUMN.
MOVE 1 TO OPEN-COUNT.
MOVE 1 TO COL-COUNT.
MOVE 'COL1' TO COLUMN-NAME.
* Describe the column host variable to the client
CALL 'TDESCRIB' USING GWL-PROC, GWL-RC, COL-COUNT, TDSCHAR,
                    COL-20, COL1-DATA, NULL-IND, TDS-FALSE,
                    TDSCHAR, COL-20, COLUMN-NAME,
                    COLUMN-NAME-LEN.
IF GWL-RC NOT EQUAL TDS-OK
    STRING 'TDESCRIB FAILED'
        DELIMITED BY SIZE INTO ERR-MSG
        PERFORM SEND-ERROR.
ADD 1 TO COL-COUNT.
MOVE 'COL2' to COLUMN-NAME.
MOVE LENGTH OF COL2-DATA TO COL2-LNG.
* Describe the column host variable to the client
CALL 'TDESCRIB' USING GWL-PROC, GWL-RC, COL-COUNT, TDSINT4,
                    COL2-LNG, COL2-DATA, NULL-IND,
                    TDS-FALSE,
                    TDSINT4, COL2-LNG, COLUMN-NAME,
                    COLUMN-NAME-LEN.
IF GWL-RC NOT EQUAL TDS-OK
    STRING 'TDESCRIB FAILED'
        DELIMITED BY SIZE INTO ERR-MSG
        PERFORM SEND-ERROR.
* Here we are just hardcoding some meaningless data into
* these columns. In a real application there must be some
* logic here to update the data columns.
MOVE 'ABCDEFGHJIJabcdefghij' TO COL1-DATA.
MOVE 999 TO COL2-DATA.
* Transaction termination routine
END-OF-REQUESTS.
* Send result completion to the client
CALL 'TDSNDDON' USING GWL-PROC, GWL-RC, TDS-DONE-FINAL,
                    TDS-ZERO, RETURN-STATUS, TDS-ENDRPC.

```

```

*   Free the session data structure and exit
    CALL 'TDFREE' USING GWL-PROC, GWL-RC.
    EXEC CICS RETURN END-EXEC.
SEND-ERROR.
    CALL 'TDSNDMSG' USING GWL-PROC  GWL-RC
                          TDS-ERROR-MSG TDS-SYBERDNR TDS-EXUSER
                          TDS-ZERO      TDS-ZERO
                          RPC-NAME      RPC-NAME-LENGTH
                          ERR-MSG       ERR-MSG-LEN.
    PERFORM END-OF-REQUESTS.

```

Sample program SYCCSAW2

The following program, SYCCSAW2, receives parameters up to 55 bytes in length and echoes them back in 55-byte rows.

Note This application replaces the sample remote stored procedure RSP3C for MDI-heritage customers. For information about RSP3C, see the Mainframe Connect Server Option *Programmer's Reference for Remote Stored Procedures*.

```

IDENTIFICATION DIVISION.
    PROGRAM-ID. SYCCSAW2.
    DATE-WRITTEN. 12/02/96.
    DATE-COMPILED.
*****
**
**           (c) 1995 by Sybase, Inc. All Rights Reserved
**
*****
*****
** PROGRAM:      SYCCSAW2
**
** THIS PROGRAM IS THE OPEN SERVER VERSION OF RSP3C.
** This program receives parms up to 55 bytes in length
** will echo it back in 55 byte rows.
** NOTE: OS app cannot recieve input pipes as an RSP can,
** this is the only method using OS to do it...
** The input data is treated a char type as RSP3c did...
** exec syw2 1234567890, 1234567890, .....
*****

```

ENVIRONMENT DIVISION.

DATA DIVISION.

WORKING-STORAGE SECTION.

```
*****
* COPY IN THE OS SERVER LIBRARYS
*****
COPY SYGWC0B.
*****
*OPEN SERVER WORK VARIABLES FOR OS CALL TO USE ...
*****
01  WS-GWL-WORK-VARIABLES.
    05  GWL-PROC                POINTER.
    05  GWL-INIT-HANDLE        POINTER.
    05  GWL-RC                  PIC S9(9) COMP.
    05  GWL-INFPRM-ID          PIC S9(9) COMP.
    05  GWL-INFPRM-TYPE        PIC S9(9) COMP.
    05  GWL-INFPRM-DATA-L      PIC S9(9) COMP.
    05  GWL-INFPRM-MAX-DATA-L  PIC S9(9) COMP.
    05  GWL-INFPRM-STATUS      PIC S9(9) COMP.
    05  GWL-INFPRM-NAME        PIC X(30) .
    05  GWL-INFPRM-NAME-L      PIC S9(9) COMP.
    05  GWL-INFPRM-USER-DATA   PIC S9(9) COMP.
    05  GWL-INFUDT-USER-TYPE   PIC S9(9) COMP.
    05  GWL-STATUS-NR          PIC S9(9) COMP.
    05  GWL-STATUS-DONE        PIC S9(9) COMP.
    05  GWL-STATUS-COUNT       PIC S9(9) COMP.
    05  GWL-STATUS-COMM        PIC S9(9) COMP.
    05  GWL-COMM-STATE         PIC S9(9) COMP.
    05  GWL-STATUS-RETURN-CODE PIC S9(9) COMP.
    05  GWL-STATUS-SUBCODE     PIC S9(9) COMP.
    05  GWL-NUMPRM-PARMS       PIC S9(9) COMP.
    05  GWL-RCVPRM-DATA-L      PIC S9(9) COMP.
    05  GWL-SETPRM-ID          PIC S9(9) COMP.
    05  GWL-SETPRM-TYPE        PIC S9(9) COMP.
    05  GWL-SETPRM-DATA-L      PIC S9(9) COMP.
    05  GWL-SETPRM-USER-DATA   PIC S9(9) COMP.
    05  GWL-CONVRT-SCALE       PIC S9(9) COMP VALUE 2.
    05  GWL-SETBCD-SCALE       PIC S9(9) COMP VALUE 0.
    05  GWL-INFBCD-LENGTH      PIC S9(9) COMP.
    05  GWL-INFBCD-SCALE       PIC S9(9) COMP.
    05  GWL-RETURN-ROWS        PIC S9(9) COMP VALUE +0.
    05  SNA-CONN-NAME          PIC X(8)  VALUE SPACES.
```

```

05 SNA-SUBC PIC S9(9) COMP.
05 WRK-DONE-STATUS PIC S9(9) COMP.
05 GWL-ACTUAL-LEN PIC S9(9) COMP.
05 GWL-TRAN-LEN PIC S9(9) COMP.
05 GWL-MSG-LEN PIC S9(9) COMP.
05 WS-NUMPRM-PARMS PIC S9(9) COMP.
05 GWL-REQUEST-TYP PIC S9(9) COMP.
05 GWL-RPC-NAME PIC X(30) VALUE SPACES.
05 GWL-COMM-STATE PIC S9(9) COMP.
05 I PIC S9(9) COMP.

01 DESCRIPTION-FIELDS.
05 COLUMN-NUMBER PIC S9(09) COMP VALUE +0.
05 HOST-TYPE PIC S9(09) COMP VALUE +0.
05 HOST-LEN PIC S9(09) COMP VALUE +0.
05 COLUMN-LEN PIC S9(09) COMP VALUE +0.
05 COLUMN-NAME-LEN PIC S9(09) COMP VALUE +0.
05 WS-ZERO PIC S9(09) COMP VALUE +0.

01 WS-MSG-WORK-VARS.
05 MSG-NR PIC S9(9) COMP VALUE +9999.

01 WS-INPUT-LEN PIC S9(9) COMP VALUE +55.
01 WS-INPUT-DATA PIC X(55) VALUE SPACES.

01 WS-OUTPUT-DATA PIC X(55) VALUE SPACES.

01 WS-OUTPUT-COL-NAME PIC X(13)
VALUE 'OUTPUT_COLUMN'.
01 WS-QUEUE-NAME.
05 WS-TRANID PIC X(4) VALUE 'SYW2'.
05 WS-TRMID PIC X(4) VALUE SPACES.
01 CICSRC PIC S9(8) COMP.
01 CICSRC-DIS PIC S9(8).

*****
* MESSAGES *
*****

01 WS-MSG.
05 FILLER PIC X(17)
VALUE 'ERROR IN OS CALL '.
05 WS-MSG-FUNC PIC X(10).
05 FILLER PIC X(04)
VALUE 'RC=' .

```

```

05 WS-MSG-RC          PIC S9(9) .
05 FILLER             PIC X(18)
   VALUE ' SUBCODE ERROR = ' .
05 MSG-SUBC          PIC 9(9) VALUE 0 .
05 WS-MSG-TEXT       PIC X(50) VALUE SPACES .

```

```

01 WORK-SRVIN-INFO .
05 WK-INFO-TBL-ID    PIC S9(8) COMP .
05 WK-INFO-TBL-NAME  PIC X(30) .
05 WK-INFO-TBL-VALUE PIC X(10) .

```

LINKAGE SECTION.

```

*****
* THE LINKAGE SECTION DEFINES MASKS FOR DATA AREAS THAT ARE
* PASSED BETWEEN THIS PROGRAM.
*****

```

```

01 DFHCOMMAREA       PIC X(1) .

```

PROCEDURE DIVISION.

000-MAIN-PROCESSING.

```

PERFORM 100-INITIALIZE          THRU 100-EXIT .
PERFORM 200-PROCESS-INPUT      THRU 200-EXIT .
PERFORM 300-PROCESS-OUTPUT     THRU 300-EXIT .
PERFORM 900-ALL-DONE           THRU 900-EXIT .

```

GOBACK.

```

000-EXIT .
EXIT .

```

100-INITIALIZE.

```

*****
* INTIALIZED THE TDS CONNECTION AND CONFIRM THAT IT
* WAS AN RPC CALL, . . . . .
*****
*==> INITIAL QUEUE NAME          <===*

```

```

MOVE EIBTRMID      TO WS-TRMID.

*==> ESTABLISH GATEWAY ENVIRONMENT <===*

CALL 'TDINIT' USING DFHEIBLK, GWL-RC, GWL-INIT-HANDLE.
IF GWL-RC NOT = TDS-OK THEN
    PERFORM 910-ERR-PROCESS THRU 910-EXIT
END-IF.

*==> ACCEPT CLIENT REQUEST <===*

CALL 'TDACCEPT' USING GWL-PROC, GWL-RC, GWL-INIT-HANDLE,
                    SNA-CONN-NAME, SNA-SUBC.
IF GWL-RC NOT = TDS-OK THEN
    PERFORM 910-ERR-PROCESS THRU 910-EXIT
END-IF.

*==> TO MAKE SURE WE WERE STARTED BY RPC REQUEST... <===*

CALL 'TDINFRPC' USING GWL-PROC,          GWL-RC,
                    GWL-REQUEST-TYP, GWL-RPC-NAME,
                    GWL-COMM-STATE.
IF GWL-RC          NOT = TDS-OK    OR
   GWL-REQUEST-TYP NOT = TDS-RPC-EVENT
THEN
    MOVE GWL-RC          TO WS-MSG-RC
    MOVE 'TDINFRPC'     TO WS-MSG-FUNC
    PERFORM 920-SEND-MESSAGE THRU 920-EXIT
    PERFORM 910-ERR-PROCESS THRU 910-EXIT
END-IF.

100-EXIT.
EXIT.

200-PROCESS-INPUT.
*****
* RECEIVE THE INPUT PARAMETER INTO HOST VARIABLE, SEND ROW DATA *
* BACK DOWN TO CLIENT                                           *
*****

*---> Find out how many parms are being passed <---*
CALL 'TDNUMPRM' USING GWL-PROC, GWL-NUMPRM-PARMS.

*---> No Parms ---> pump back a message          <---*

```

```

IF GWL-NUMPRM-PARMS < +1 THEN
  MOVE 'At least one parm is needed'
      TO WS-MSG-TEXT
  MOVE GWL-RC          TO WS-MSG-RC
  MOVE 'TDNUMPRM'     TO WS-MSG-FUNC
  PERFORM 920-SEND-MESSAGE THRU 920-EXIT
  PERFORM 910-ERR-PROCESS THRU 910-EXIT
END-IF

*---> SAVE THE NUMBER OF PARMS FOR THE LOOP <---*
MOVE GWL-NUMPRM-PARMS TO WS-NUMPRM-PARMS.

*---> LOOP THRU THE PARMS AND WRITE TO TEMP STORAGE <---*
PERFORM VARYING GWL-NUMPRM-PARMS FROM 1 BY 1
      UNTIL GWL-NUMPRM-PARMS > WS-NUMPRM-PARMS
  PERFORM 210-GET-PARM THRU 210-EXIT
  PERFORM 220-WRITE-TS THRU 220-EXIT

END-PERFORM.
200-EXIT.
EXIT.
210-GET-PARM.
*****
* *---> GET THE PARM INTO THE HOST VARIABLE <---* *
*****

CALL 'TDRCVPRM' USING GWL-PROC, GWL-RC,
                    GWL-NUMPRM-PARMS,
                    WS-INPUT-DATA,
                    TDSCHAR,
                    WS-INPUT-LEN,
                    GWL-ACTUAL-LEN

IF GWL-RC NOT = TDS-OK THEN
  MOVE GWL-RC          TO WS-MSG-RC
  MOVE 'TDRCVPRM'     TO WS-MSG-FUNC
  PERFORM 920-SEND-MESSAGE THRU 920-EXIT
  PERFORM 910-ERR-PROCESS THRU 910-EXIT
END-IF.
210-EXIT.
EXIT.
220-WRITE-TS.
*****
* *---> USING TEMP STORAGE, STORE PARMS FOR OUTPUT LATER <---*
*****

```



```

EXEC CICS
  WRITEQ TS QUEUE(WS-QUEUE-NAME)
        FROM (WS-INPUT-DATA)
        LENGTH(LENGTH OF WS-INPUT-DATA)
        RESP (CICSRC)

END-EXEC.
IF CICSRC NOT = DFHRESP(NORMAL)
  MOVE CICSRC          TO CICSRC-DIS
  MOVE CICSRC-DIS     TO WS-MSG-RC
  MOVE 'WRITEQ'       TO WS-MSG-FUNC
  PERFORM 920-SEND-MESSAGE THRU 920-EXIT
  PERFORM 910-ERR-PROCESS THRU 910-EXIT
END-IF.

220-EXIT.
EXIT.

300-PROCESS-OUTPUT.
*****
* READ TEMP STORAGE QUEUE AND SEND ROWS TO CLIENT          *
*****

PERFORM 310-DEFINE-OUTPUT THRU 310-EXIT.

PERFORM VARYING I FROM 1 BY 1 UNTIL I > WS-NUMPRM-PARMS
  PERFORM 320-READQ-TS THRU 320-EXIT
  PERFORM 330-SEND-ROW THRU 330-EXIT

END-PERFORM.

300-EXIT.
EXIT.

310-DEFINE-OUTPUT.
*****
* DEFINE THE OUTPUT COLUMN AS CHAR OF 55 BYTES          *
*****

MOVE +1                                TO COLUMN-NUMBER.
MOVE LENGTH OF WS-OUTPUT-DATA          TO HOST-LEN
                                         COLUMN-LEN.
MOVE LENGTH OF WS-OUTPUT-COL-NAME      TO COLUMN-NAME-LEN.
CALL 'TDESCRIB' USING GWL-PROC,
                    GWL-RC,
                    COLUMN-NUMBER,

```

```

                                TDSCHAR,
                                HOST-LEN,
                                WS-OUTPUT-DATA,
                                TDS-ZERO,
                                TDS-FALSE,
                                TDSCHAR,
                                COLUMN-LEN,
                                WS-OUTPUT-COL-NAME,
                                COLUMN-NAME-LEN.

IF GWL-RC NOT = TDS-OK THEN
    MOVE GWL-RC          TO WS-MSG-RC
    MOVE 'TDESCRIB'     TO WS-MSG-FUNC
    PERFORM 920-SEND-MESSAGE THRU 920-EXIT
    PERFORM 910-ERR-PROCESS THRU 910-EXIT
END-IF.

310-EXIT.
EXIT.

320-READQ-TS.
*****
* READ THE INPUT TEMP STORAGE QUEUE
*****
    EXEC CICS
        READQ TS QUEUE (WS-QUEUE-NAME)
            INTO (WS-OUTPUT-DATA)
            LENGTH (LENGTH OF WS-OUTPUT-DATA)
            NEXT
            RESP (CICSRC)
    END-EXEC.
IF CICSRC NOT = DFHRESP (NORMAL)
    MOVE CICSRC          TO CICSRC-DIS
    MOVE CICSRC-DIS     TO WS-MSG-RC
    MOVE 'READQ'        TO WS-MSG-FUNC
    PERFORM 920-SEND-MESSAGE THRU 920-EXIT
    PERFORM 910-ERR-PROCESS THRU 910-EXIT
END-IF.

320-EXIT.
EXIT.

330-SEND-ROW.
*****
* SEND ROW OF DATA TO CLIENT....

```

```
CALL 'TDSNDROW' USING GWL-PROC, GWL-RC
IF GWL-RC NOT = TDS-OK
THEN
    MOVE GWL-RC          TO WS-MSG-RC
    MOVE 'TDSNDROW'      TO WS-MSG-FUNC
    PERFORM 920-SEND-MESSAGE THRU 920-EXIT
    PERFORM 910-ERR-PROCESS THRU 910-EXIT
END-IF.
```

```
330-EXIT.
EXIT.
EJECT
900-ALL-DONE.
```

 * CLOSE CONNECTION TO CLIENT AND RETURN TO CICS... *

```
CALL 'TDSNDDON' USING GWL-PROC, GWL-RC, WRK-DONE-STATUS,
                    GWL-RETURN-ROWS, TDS-ZERO, TDS-ENDRPC.
IF GWL-RC NOT = TDS-OK THEN
    PERFORM 980-CICS-DUMP      THRU 980-EXIT
    PERFORM 990-CICS-RETURN   THRU 990-EXIT
END-IF.
```

```
CALL 'TDFREE' USING GWL-PROC, GWL-RC.
```

```
EXEC CICS
    DELETEQ TS QUEUE(WS-QUEUE-NAME)
           RESP (CICSRC)
END-EXEC.
IF CICSRC NOT = DFHRESP(NORMAL)
    MOVE CICSRC          TO CICSRC-DIS
    MOVE CICSRC-DIS     TO WS-MSG-RC
    MOVE 'DELETEQ'      TO WS-MSG-FUNC
    PERFORM 920-SEND-MESSAGE THRU 920-EXIT
    PERFORM 910-ERR-PROCESS THRU 910-EXIT
END-IF.
```

```
PERFORM 990-CICS-RETURN      THRU 990-EXIT.
```

```
900-EXIT.
EXIT.
```

```

910-ERR-PROCESS.
*****
* PERFORM ALL-DONE IN A ERROR STATE *
*****

MOVE ZERO          TO GWL-RETURN-ROWS.
MOVE TDS-DONE-ERROR TO WRK-DONE-STATUS.
PERFORM 900-ALL-DONE THRU 900-EXIT.
910-EXIT.
EXIT.
920-SEND-MESSAGE.
*-----
SEND-ERROR-MESSAGE.
*-----

MOVE 'N'          TO SEND-DONE-SW.
MOVE TDS-ERROR-MSG TO MSG-TYPE.
MOVE LENGTH OF MSG-RPC TO MSG-RPC-L.
*   Ensure we're in right state to send a message
CALL 'TDSTATUS' USING 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 AND
    GWL-STATUS-COMM = TDS-RECEIVE) THEN
    CALL 'TDSNDMSG' USING 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-IF.
920-EXIT.
EXIT.
980-CICS-DUMP.
*****
* CAUSE A CICS TRANSACTION DUMP USUALLY BECAUSE SOMETHING IS BAD *
*****
EXEC CICS
    DUMP DUMPCODE('SYW2') NOHANDLE
END-EXEC.

```

```

980-EXIT.
    EXIT.

990-CICS-RETURN.
*****
* RETURN TO CICS... *
*****

EXEC CICS
    RETURN
END-EXEC.

990-EXIT.
    EXIT.

```

Sample program SYCCSAY2

The following program receives one of two keywords, @ERRORMSG or @WARNMSG and other keywords, and then replies with the keywords and data.

Note This application replaces the sample remote stored procedure RSP4C for MDI-heritage customers. For information about RSP4C, see the Mainframe Connect Server Option *Programmer's Reference for Remote Stored Procedures*.

```

IDENTIFICATION DIVISION.
    PROGRAM-ID. SYCCSAY2.
    DATE-WRITTEN. 12/17/96.
    DATE-COMPILED.
*****
**
**          (c) 1995 by Sybase, Inc. All Rights Reserved
*****

*****
** PROGRAM:    SYCCSAY2
**
** THIS PROGRAM IS A THE OPEN SERVER VERSION OF RSP4C.
** It will receive one of 2 Keywords @ERRORMSG or @WARNMSG and
** Other Keywords. Will reply with the keywords and data.

```

```

** If @ERRORMSG AND/OR @WARNMSG are 'Y' that type of message
** will be returned..
** exec sy2 @WARNMSG=Y,@ERRORMSG=Y.....
*****

```

```

ENVIRONMENT DIVISION.
DATA DIVISION.

```

```

WORKING-STORAGE SECTION.

```

```

*****
* COPY IN THE OS SERVER LIBRARYS
*****
COPY SYGWC0B.
*****
*OPEN SERVER WORK VARIABLES FOR OS CALL TO USE ...
*****

```

```

01  WS-GWL-WORK-VARIABLES.
    05  GWL-PROC                POINTER.
    05  GWL-INIT-HANDLE         POINTER.
    05  GWL-RC                  PIC S9(9)  COMP.
    05  GWL-INFPRM-ID           PIC S9(9)  COMP.
    05  GWL-INFPRM-TYPE         PIC S9(9)  COMP.
    05  GWL-INFPRM-DATA-L       PIC S9(9)  COMP.
    05  GWL-INFPRM-MAX-DATA-L   PIC S9(9)  COMP.
    05  GWL-INFPRM-STATUS       PIC S9(9)  COMP.
    05  GWL-INFPRM-NAME         PIC X(30) .
    05  GWL-INFPRM-NAME-L       PIC S9(9)  COMP.
    05  GWL-INFPRM-USER-DATA    PIC S9(9)  COMP.
    05  GWL-INFUDT-USER-TYPE    PIC S9(9)  COMP.
    05  GWL-STATUS-NR           PIC S9(9)  COMP.
    05  GWL-STATUS-DONE         PIC S9(9)  COMP.
    05  GWL-STATUS-COUNT        PIC S9(9)  COMP.
    05  GWL-STATUS-COMM         PIC S9(9)  COMP.
    05  GWL-COMM-STATE          PIC S9(9)  COMP.
    05  GWL-STATUS-RETURN-CODE  PIC S9(9)  COMP.
    05  GWL-STATUS-SUBCODE      PIC S9(9)  COMP.
    05  GWL-NUMPRM-PARMS        PIC S9(9)  COMP.
    05  GWL-RCVPRM-DATA-L       PIC S9(9)  COMP.
    05  GWL-SETPRM-ID           PIC S9(9)  COMP.
    05  GWL-SETPRM-TYPE         PIC S9(9)  COMP.
    05  GWL-SETPRM-DATA-L       PIC S9(9)  COMP.
    05  GWL-SETPRM-USER-DATA    PIC S9(9)  COMP.
    05  GWL-CONVRT-SCALE        PIC S9(9)  COMP VALUE 2.
    05  GWL-SETBCD-SCALE        PIC S9(9)  COMP VALUE 0.
    05  GWL-INFBCD-LENGTH       PIC S9(9)  COMP.

```

```

05  GWL-INFBCD-SCALE          PIC S9(9) COMP.
05  GWL-RETURN-ROWS          PIC S9(9) COMP VALUE +0.
05  SNA-CONN-NAME           PIC X(8)  VALUE SPACES.
05  SNA-SUBC                 PIC S9(9) COMP.
05  WRK-DONE-STATUS         PIC S9(9) COMP.
05  GWL-ACTUAL-LEN          PIC S9(9) COMP.
05  GWL-TRAN-LEN           PIC S9(9) COMP.
05  GWL-MSG-LEN            PIC S9(9) COMP.
05  WS-NUMPRM-PARMS        PIC S9(9) COMP.
05  GWL-REQUEST-TYP        PIC S9(9) COMP.
05  GWL-RPC-NAME           PIC X(30) VALUE SPACES.
05  GWL-COMM-STATE         PIC S9(9) COMP.
05  I                       PIC S9(9) COMP.
05  WS-ERROR-MSG           PIC S9(9) COMP VALUE ZERO.
05  WS-ERROR-SEV           PIC S9(9) COMP VALUE ZERO.
01  DESCRIPTION-FIELDS.
05  COLUMN-NUMBER          PIC S9(09) COMP VALUE +0.
05  HOST-TYPE              PIC S9(09) COMP VALUE +0.
05  HOST-LEN              PIC S9(09) COMP VALUE +0.
05  COLUMN-LEN            PIC S9(09) COMP VALUE +0.
05  COLUMN-NAME-LEN       PIC S9(09) COMP VALUE +0.
05  WS-ZERO               PIC S9(09) COMP VALUE +0.

01  WS-MSG-WORK-VARS.
05  MSG-NR                 PIC S9(9) COMP VALUE +9999.

01  WS-INPUT-LEN          PIC S9(9) COMP VALUE +55.
01  WS-INPUT-DATA         PIC X(55)  VALUE SPACES.

01  WS-LENGTH            PIC S9(9) COMP VALUE ZERO.
01  WS-WARNMSG           PIC X(8)   VALUE '@WARNMSG'.
01  WS-WARNMSG-ID        PIC S9(9) COMP VALUE ZERO.
01  WS-WARNMSG-88        PIC X(1)   VALUE 'N'.
08  WARNING-MSG          VALUE 'Y'.

01  WS-ERRORMSG          PIC X(9)   VALUE '@ERRORMSG'.
01  WS-ERRORMSG-ID       PIC S9(9) COMP VALUE ZERO.
01  WS-ERRORMSG-88       PIC X(1)   VALUE 'N'.
08  ERROR-MSG           VALUE 'Y'.
01  WS-OUTPUT-DATA       PIC X(55)  VALUE SPACES.

01  WS-OUTPUT-COL-NAME    PIC X(13)
    VALUE 'OUTPUT_COLUMN'.

01  WS-QUEUE-NAME.
05  WS-TRANID            PIC X(4)  VALUE 'SYY2'.

```

```

05 WS-TRMID          PIC X(4) VALUE SPACES.
01 CICSRC            PIC S9(8) COMP.
01 CICSRC-DIS        PIC S9(8) .

```

```

*****
* MESSAGES                                                    *
*****

```

```

01 WS-MSG.
05 FILLER              PIC X(17)
   VALUE 'ERROR IN OS CALL ' .
05 WS-MSG-FUNC         PIC X(10) .
05 FILLER              PIC X(04)
   VALUE 'RC=' .
05 WS-MSG-RC           PIC S9(9) .
05 FILLER              PIC X(18)
   VALUE ' SUBCODE ERROR = ' .
05 MSG-SUBC            PIC 9(9) VALUE 0.
05 WS-MSG-TEXT         PIC X(50) VALUE SPACES.

01 WS-HOLD-MSG         PIC X(107) VALUE SPACES.
01 WS-WARN-MSG         PIC X(107) VALUE
   'THIS IS A WARNING MESSAGE.....' .
01 WS-ERR-MSG         PIC X(107) VALUE
   'THIS IS A ERROR MESSAGE.....' .

01 WORK-SRVIN-INFO.
05 WK-INFO-TBL-ID     PIC S9(8) COMP.
05 WK-INFO-TBL-NAME   PIC X(30) .
05 WK-INFO-TBL-VALUE  PIC X(10) .

```

LINKAGE SECTION.

```

*****
* THE LINKAGE SECTION DEFINES MASKS FOR DATA AREAS THAT ARE
* PASSED BETWEEN THIS PROGRAM.
*****

```

```

01 DFHCOMMAREA        PIC X(1) .
PROCEDURE DIVISION.
000-MAIN-PROCESSING.

    PERFORM 100-INITIALIZE          THRU 100-EXIT.
    PERFORM 200-PROCESS-INPUT       THRU 200-EXIT.

    PERFORM 300-PROCESS-OUTPUT      THRU 300-EXIT.

```



```

PERFORM 900-ALL-DONE                THRU 900-EXIT.

GOBACK.

000-EXIT.
EXIT.

100-INITIALIZE.

*****
* INTIALIZED THE TDS CONNECTION AND CONFIRM THAT IT
* WAS AN RPC CALL,      . . . . .
*****
*==> INITIAL QUEUE NAME                <===*
      MOVE EIBTRMID      TO WS-TRMID.

*==> ESTABLISH GATEWAY ENVIRONMENT <===*

      CALL 'TDINIT' USING DFHEIBLK, GWL-RC, GWL-INIT-HANDLE.
      IF GWL-RC NOT = TDS-OK THEN
          PERFORM 910-ERR-PROCESS THRU 910-EXIT
      END-IF.

*==> ACCEPT CLIENT REQUEST <===*

      CALL 'TDACCEPT' USING GWL-PROC, GWL-RC, GWL-INIT-HANDLE,
                          SNA-CONN-NAME, SNA-SUBC.
      IF GWL-RC NOT = TDS-OK THEN
          PERFORM 910-ERR-PROCESS THRU 910-EXIT
      END-IF.

*==> TO MAKE SURE WE WERE STARTED BY RPC REQUEST... <===*
      CALL 'TDINFRPC' USING GWL-PROC,          GWL-RC,
                          GWL-REQUEST-TYP, GWL-RPC-NAME,
                          GWL-COMM-STATE.
      IF GWL-RC          NOT = TDS-OK    OR
         GWL-REQUEST-TYP NOT = TDS-RPC-EVENT
      THEN
          MOVE GWL-RC          TO WS-MSG-RC
          MOVE 'TDINFRPC'      TO WS-MSG-FUNC
          PERFORM 920-SEND-MESSAGE THRU 920-EXIT
          PERFORM 910-ERR-PROCESS THRU 910-EXIT
      END-IF.

100-EXIT.

```

```

EXIT.

200-PROCESS-INPUT.
*****
* RECEIVE THE INPUT PARAMETER INTO HOST VARIABLE, SEND ROW DATA *
* BACK DOWN TO CLIENT *
*****

*---> Find out how many parms are being passed <---*
      CALL 'TDNUMPRM' USING GWL-PROC, GWL-NUMPRM-PARMS.

*---> NO PARMS, pump back a message          <---*

      IF GWL-NUMPRM-PARMS < +1 THEN
          MOVE 'At least one parm is needed'
              TO WS-MSG-TEXT
          MOVE GWL-RC          TO WS-MSG-RC
          MOVE 'TDNUMPRM'     TO WS-MSG-FUNC
          MOVE WS-MSG         TO WS-HOLD-MSG
          MOVE TDS-ERROR-MSG TO WS-ERROR-MSG
          MOVE TDS-ERROR-SEV TO WS-ERROR-SEV
          PERFORM 920-SEND-MESSAGE THRU 920-EXIT
          PERFORM 910-ERR-PROCESS THRU 910-EXIT
      END-IF.

*---> TEST TO SEE IF THE KEYWORDS "WARNMSG" AND <---*
*---> OR ERRORMSG WHERE SENT...          <---*
      MOVE LENGTH OF WS-WARNMSG TO WS-LENGTH.
      CALL 'TDLOCPRM' USING GWL-PROC, WS-WARNMSG-ID,
          WS-WARNMSG, WS-LENGTH.

      MOVE LENGTH OF WS-ERRORMSG TO WS-LENGTH.
      CALL 'TDLOCPRM' USING GWL-PROC, WS-ERRORMSG-ID,
          WS-ERRORMSG, WS-LENGTH.

*---> SAVE THE NUMBER OF PARMS FOR THE LOOP <---*
      MOVE GWL-NUMPRM-PARMS TO WS-NUMPRM-PARMS.

*---> LOOP THRU THE PARMS AND WRITE TO TEMP STORAGE <---*
      PERFORM VARYING GWL-NUMPRM-PARMS FROM 1 BY 1
          UNTIL GWL-NUMPRM-PARMS > WS-NUMPRM-PARMS
          PERFORM 210-GET-PARM THRU 210-EXIT
          PERFORM 220-WRITE-TS THRU 220-EXIT

      END-PERFORM.
200-EXIT.

```

```

EXIT.
210-GET-PARM.
*****
* *--> Get that parm info into the host variable <--* *
*****

CALL 'TDRCVPRM' USING GWL-PROC, GWL-RC,
                        GWL-NUMPRM-PARMS,
                        WS-INPUT-DATA,
                        TDSCHAR,
                        WS-INPUT-LEN,
                        GWL-ACTUAL-LEN

IF GWL-RC NOT = TDS-OK THEN
    MOVE GWL-RC          TO WS-MSG-RC
    MOVE 'TDRCVPRM'     TO WS-MSG-FUNC
    MOVE WS-MSG         TO WS-HOLD-MSG
    MOVE TDS-ERROR-MSG TO WS-ERROR-MSG
    MOVE TDS-ERROR-SEV TO WS-ERROR-SEV
    PERFORM 920-SEND-MESSAGE THRU 920-EXIT
    PERFORM 910-ERR-PROCESS THRU 910-EXIT
END-IF.

210-EXIT.
EXIT.
220-WRITE-TS.
*****
* *--> WRITE PARMS TO TEMP STORAGE, LATER RETURN PARMS <--* *
* *--> BACK DOWN TO CLIENT AS OUTPUT <--* *
*****

EXEC CICS
    WRITEQ TS QUEUE(WS-QUEUE-NAME)
           FROM (WS-INPUT-DATA)
           LENGTH(LENGTH OF WS-INPUT-DATA)
           RESP (CICSRC)

END-EXEC.
IF CICSRC NOT = DFHRESP(NORMAL)
    MOVE CICSRC          TO CICSRC-DIS
    MOVE CICSRC-DIS     TO WS-MSG-RC
    MOVE 'WRITEQ'       TO WS-MSG-FUNC
    MOVE WS-MSG         TO WS-HOLD-MSG
    MOVE TDS-ERROR-MSG TO WS-ERROR-MSG
    MOVE TDS-ERROR-SEV TO WS-ERROR-SEV
    PERFORM 920-SEND-MESSAGE THRU 920-EXIT
    PERFORM 910-ERR-PROCESS THRU 910-EXIT
END-IF.

```

```

220-EXIT.
EXIT.
300-PROCESS-OUTPUT.
*****
* READ TEMP STORAGE QUEUE AND SEND ROWS TO CLIENT *
*****

PERFORM 310-DEFINE-OUTPUT THRU 310-EXIT.

PERFORM VARYING I FROM 1 BY 1 UNTIL I > WS-NUMPRM-PARMS
PERFORM 320-READQ-TS THRU 320-EXIT
PERFORM 330-SEND-ROW THRU 330-EXIT
END-PERFORM.

*---> PROCESS WARNMSG AND/OR ERRORMSG AFTER SENDING ROWS. <---*
IF WARNING-MSG
THEN
MOVE TDS-INFO-MSG TO WS-ERROR-MSG
MOVE TDS-INFO-SEV TO WS-ERROR-SEV
MOVE WS-WARN-MSG TO WS-HOLD-MSG
PERFORM 920-SEND-MESSAGE THRU 920-EXIT
END-IF.
IF ERROR-MSG
THEN
MOVE TDS-ERROR-MSG TO WS-ERROR-MSG
MOVE TDS-ERROR-SEV TO WS-ERROR-SEV
MOVE WS-ERR-MSG TO WS-HOLD-MSG
PERFORM 920-SEND-MESSAGE THRU 920-EXIT
END-IF.
300-EXIT.
EXIT.

310-DEFINE-OUTPUT.
*****
* DEFINE THE OUTPUT COLUM AS CHAR OF 55 BYTES *
*****

MOVE +1 TO COLUMN-NUMBER.
MOVE LENGTH OF WS-OUTPUT-DATA TO HOST-LEN
COLUMN-LEN.
MOVE LENGTH OF WS-OUTPUT-COL-NAME TO COLUMN-NAME-LEN.
CALL 'TDESCRIB' USING GWL-PROC,
GWL-RC,
COLUMN-NUMBER,

```

```

                                TDSCHAR,
                                HOST-LEN,
                                WS-OUTPUT-DATA,
                                TDS-ZERO,
                                TDS-FALSE,
                                TDSCHAR,
                                COLUMN-LEN,
                                WS-OUTPUT-COL-NAME,
                                COLUMN-NAME-LEN.

IF GWL-RC NOT = TDS-OK THEN
    MOVE GWL-RC          TO WS-MSG-RC
    MOVE 'TDESCRIB'     TO WS-MSG-FUNC
    MOVE WS-MSG         TO WS-HOLD-MSG
    MOVE TDS-ERROR-MSG TO WS-ERROR-MSG
    MOVE TDS-ERROR-SEV TO WS-ERROR-SEV
    PERFORM 920-SEND-MESSAGE THRU 920-EXIT
    PERFORM 910-ERR-PROCESS THRU 910-EXIT
END-IF.

310-EXIT.
EXIT.

320-READQ-TS.
*****
* READ THE INPUT TEMP STORAGE QUEUE
*****
EXEC CICS
    READQ TS QUEUE(WS-QUEUE-NAME)
           INTO (WS-OUTPUT-DATA)
           LENGTH(LENGTH OF WS-OUTPUT-DATA)
           NEXT
           RESP (CICSRC)
END-EXEC.
IF CICSRC NOT = DFHRESP(NORMAL)
    MOVE CICSRC          TO CICSRC-DIS
    MOVE CICSRC-DIS     TO WS-MSG-RC
    MOVE 'READQ'       TO WS-MSG-FUNC
    MOVE WS-MSG         TO WS-HOLD-MSG
    MOVE TDS-ERROR-MSG TO WS-ERROR-MSG
    PERFORM 920-SEND-MESSAGE THRU 920-EXIT
    PERFORM 910-ERR-PROCESS THRU 910-EXIT
END-IF.
*---> PROCESS WARNMSG AND/OR ERRORMSG PARMS IF YES... <---*
IF WS-WARNMSG-ID = I AND WS-OUTPUT-DATA = 'Y'
    MOVE 'Y'          TO WS-WARNMSG-88.
IF WS-ERRORMSG-ID = I AND WS-OUTPUT-DATA = 'Y'

```

```

        MOVE 'Y'                TO WS-ERRORMSG-88.
320-EXIT.
        EXIT.

330-SEND-ROW.
*****
* SEND ROW OF DATA TO CLIENT...
*****

        CALL 'TDSNDROW' USING GWL-PROC, GWL-RC
        IF GWL-RC NOT = 'TDS-OK'
        THEN
            MOVE GWL-RC          TO WS-MSG-RC
            MOVE 'TDSNDROW'     TO WS-MSG-FUNC
            MOVE WS-MSG          TO WS-HOLD-MSG
            MOVE TDS-ERROR-MSG  TO WS-ERROR-MSG
            MOVE TDS-ERROR-SEV  TO WS-ERROR-SEV
            PERFORM 920-SEND-MESSAGE THRU 920-EXIT
            PERFORM 910-ERR-PROCESS THRU 910-EXIT
        END-IF.

330-EXIT.
        EXIT.
        EJECT
900-ALL-DONE.
*****
* CLOSE CONNECTION TO CLIENT AND RETURN TO CICS... *
*****
        CALL 'TDSNDDON' USING GWL-PROC, GWL-RC, WRK-DONE-STATUS,
                        GWL-RETURN-ROWS, TDS-ZERO, TDS-ENDRPC.
        IF GWL-RC NOT = 'TDS-OK' THEN
            PERFORM 980-CICS-DUMP      THRU 980-EXIT
            PERFORM 990-CICS-RETURN  THRU 990-EXIT
        END-IF.
        CALL 'TDFREE' USING GWL-PROC, GWL-RC.

EXEC CICS
        DELETEQ TS QUEUE(WS-QUEUE-NAME)
                RESP (CICSRC)
END-EXEC.
IF CICSRC NOT = DFHRESP(NORMAL)
        MOVE CICSRC          TO CICSRC-DIS
        MOVE CICSRC-DIS     TO WS-MSG-RC
        MOVE 'DELETEQ'      TO WS-MSG-FUNC
        MOVE WS-MSG         TO WS-HOLD-MSG
        MOVE TDS-ERROR-MSG  TO WS-ERROR-MSG

```

```

MOVE TDS-ERROR-SEV TO WS-ERROR-SEV
PERFORM 920-SEND-MESSAGE THRU 920-EXIT
PERFORM 910-ERR-PROCESS THRU 910-EXIT
END-IF.

```

```

PERFORM 990-CICS-RETURN THRU 990-EXIT.

```

```

900-EXIT.
EXIT.

```

```

910-ERR-PROCESS.

```

```

*****
* PERFORM ALL-DONE IN A ERROR STATE *
*****

```

```

MOVE ZERO TO GWL-RETURN-ROWS.
MOVE TDS-DONE-ERROR TO WRK-DONE-STATUS.
PERFORM 900-ALL-DONE THRU 900-EXIT.

```

```

910-EXIT.
EXIT.

```

```

920-SEND-MESSAGE.

```

```

*-----
SEND-ERROR-MESSAGE.
*-----

```

```

MOVE 'N' TO SEND-DONE-SW.
MOVE TDS-ERROR-MSG TO MSG-TYPE.
MOVE LENGTH OF MSG-RPC TO MSG-RPC-L.
* Ensure we're in right state to send a message
CALL 'TDSTATUS' USING 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 AND
GWL-STATUS-COMM = TDS-RECEIVE) THEN
CALL 'TDSNDMSG' USING 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-IF.
920-EXIT.
EXIT.

980-CICS-DUMP.
*****
* CAUSE A CICS TRANSACTION DUMP USUALLY BECAUSE SOMETHING IS BAD *
*****
EXEC CICS
  DUMP DUMPCODE('SYZ2') NOHANDLE
END-EXEC.

980-EXIT.
EXIT.

990-CICS-RETURN.
*****
* RETURN TO CICS... *
*****

EXEC CICS
  RETURN
END-EXEC.

990-EXIT.
EXIT.
```

Sample program SYCCSAZ2

The following program receives a text input string (10,000 bytes) and returns it in a 50-byte column one row at a time.

Note This application replaces the sample remote stored procedure RSP8C for MDI-heritage customers. For information about RSP8C, see the Mainframe Connect Server Option *Programmer's Reference for Remote Stored Procedures*.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SYCCSAZ2.
DATE-WRITTEN. 09/17/96.
DATE-COMPILED.
```



```

*****
**
**      (c) 1995 by Sybase, Inc. All Rights Reserved
**
*****

*****
** PROGRAM:      SYCCSAZ2  TRAN:SYZ2...
**
** THIS PROGRAM IS A THE OPEN SERVER VERSION OF RSP8C.  RECEIVES
** A TEXT INPUT STRING(10,000 BYTES) AND RETURNS IT IN A 50 BYTE
** COLUMN ONE ROW AT A TIME...
** Example: exec syz2 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
*****

ENVIRONMENT DIVISION.

DATA DIVISION.

WORKING-STORAGE SECTION.

*****
* COPY IN THE OS SERVER LIBRARYS
*****
COPY SYGWC0B.
*****
*OPEN SERVER WORK VARIABLES FOR OS CALL TO USE ...
*****
01  WS-GWL-WORK-VARIABLES.
    05  GWL-PROC                POINTER.
    05  GWL-INIT-HANDLE        POINTER.
    05  GWL-RC                  PIC S9(9) COMP.
    05  GWL-INFPRM-ID          PIC S9(9) COMP.
    05  GWL-INFPRM-TYPE        PIC S9(9) COMP.
    05  GWL-INFPRM-DATA-L      PIC S9(9) COMP.
    05  GWL-INFPRM-MAX-DATA-L  PIC S9(9) COMP.
    05  GWL-INFPRM-STATUS      PIC S9(9) COMP.
    05  GWL-INFPRM-NAME        PIC X(30) .
    05  GWL-INFPRM-NAME-L      PIC S9(9) COMP.
    05  GWL-INFPRM-USER-DATA   PIC S9(9) COMP.
    05  GWL-INFUDT-USER-TYPE   PIC S9(9) COMP.
    05  GWL-STATUS-NR          PIC S9(9) COMP.
    05  GWL-STATUS-DONE        PIC S9(9) COMP.
    05  GWL-STATUS-COUNT       PIC S9(9) COMP.
    05  GWL-STATUS-COMM        PIC S9(9) COMP.
    05  GWL-COMM-STATE         PIC S9(9) COMP.

```

```

05  GWL-STATUS-RETURN-CODE  PIC S9(9)  COMP.
05  GWL-STATUS-SUBCODE      PIC S9(9)  COMP.
05  GWL-NUMPRM-PARMS        PIC S9(9)  COMP.
05  GWL-RCVPRM-DATA-L       PIC S9(9)  COMP.
05  GWL-SETPRM-ID           PIC S9(9)  COMP.
05  GWL-SETPRM-TYPE         PIC S9(9)  COMP.
05  GWL-SETPRM-DATA-L       PIC S9(9)  COMP.
05  GWL-SETPRM-USER-DATA    PIC S9(9)  COMP.
05  GWL-CONVRT-SCALE        PIC S9(9)  COMP VALUE 2.
05  GWL-SETBCD-SCALE        PIC S9(9)  COMP VALUE 0.
05  GWL-INFBCD-LENGTH       PIC S9(9)  COMP.
05  GWL-INFBCD-SCALE        PIC S9(9)  COMP.
05  GWL-RETURN-ROWS         PIC S9(9)  COMP VALUE +0.
05  SNA-CONN-NAME           PIC X(8)   VALUE SPACES.
05  SNA-SUBC                 PIC S9(9)  COMP.
05  WRK-DONE-STATUS         PIC S9(9)  COMP.
05  GWL-ACTUAL-LEN          PIC S9(9)  COMP.
05  GWL-TRAN-LEN            PIC S9(9)  COMP.
05  GWL-MSG-LEN             PIC S9(9)  COMP.
05  GWL-REQUEST-TYP        PIC S9(9)  COMP.
05  GWL-RPC-NAME            PIC X(30)  VALUE SPACES.
05  GWL-COMM-STATE          PIC S9(9)  COMP.
05  I                        PIC S9(9)  COMP VALUE +0.
05  J                        PIC S9(4)  COMP VALUE +0.

01  DESCRIPTION-FIELDS.
05  COLUMN-NUMBER           PIC S9(09)  COMP VALUE +0.
05  HOST-TYPE               PIC S9(09)  COMP VALUE +0.
05  HOST-LEN                PIC S9(09)  COMP VALUE +0.
05  COLUMN-LEN              PIC S9(09)  COMP VALUE +0.
05  COLUMN-NAME-LEN         PIC S9(09)  COMP VALUE +0.

01  WS-MSG-WORK-VARS.
05  MSG-NR                  PIC S9(9)  COMP VALUE +9999.

01  WS-INPUT-LEN            PIC S9(9)  COMP VALUE +10000.
01  WS-INPUT-DATA-HDR.
03  WS-INPUT-DATA           PIC X(10000)  VALUE SPACES.
03  WS-INPUT-REDEFINE REDEFINES WS-INPUT-DATA.
05  WS-INPUT-TABLE OCCURS 10000 TIMES.
10  WS-INPUT-CHAR           PIC X.

01  WS-OUTPUT-DATA-HDR.
03  WS-OUTPUT-DATA          PIC X(50)   VALUE SPACES.
03  WS-OUTPUT-REDEFINE REDEFINES WS-OUTPUT-DATA.
05  WS-OUTPUT-TABLE OCCURS 50 TIMES.
10  WS-OUTPUT-CHAR         PIC X.

```

```
01 WS-OUTPUT-COL-NAME          PIC X(13)
   VALUE 'OUTPUT_COLUMN'.
```

```
*****
* MESSAGES                                                                *
*****
```

```
01 WS-MSG.
   05 FILLER                    PIC X(17)
      VALUE 'ERROR IN OS CALL '.
   05 WS-MSG-FUNC               PIC X(10).
   05 FILLER                    PIC X(04)
      VALUE 'RC=' .
   05 WS-MSG-RC                 PIC 9(9).
   05 FILLER                    PIC X(18)
      VALUE ' SUBCODE ERROR = '.
   05 MSG-SUBC                  PIC 9(9) VALUE 0.
   05 WS-MSG-TEXT               PIC X(50) VALUE SPACES.
```

```
01 WORK-SRVIN-INFO.
   05 WK-INFO-TBL-ID           PIC S9(8) COMP.
   05 WK-INFO-TBL-NAME        PIC X(30).
   05 WK-INFO-TBL-VALUE       PIC X(10).
```

LINKAGE SECTION.

```
*****
* THE LINKAGE SECTION DEFINES MASKS FOR DATA AREAS THAT ARE
* PASSED BETWEEN THIS PROGRAM.
*****
```

```
01 DFHCOMMAREA                 PIC X(1).
```

PROCEDURE DIVISION.

000-MAIN-PROCESSING.

```
PERFORM 100-INITIALIZE          THRU 100-EXIT.
PERFORM 200-PROCESS-INPUT      THRU 200-EXIT.
PERFORM 300-PROCESS-OUTPUT     THRU 300-EXIT.
```

Sample program SYCCSAZ2

```
PERFORM 900-ALL-DONE                THRU 900-EXIT.

GOBACK.

000-EXIT.
EXIT.

100-INITIALIZE.
*****
* INITIALIZE THE TDS CONNECTION AND RECEIVE THE
* RPC PARM.....
*****

*==> ESTABLISH GATEWAY ENVIRONMENT <===*

CALL 'TDINIT' USING DFHEIBLK, GWL-RC, GWL-INIT-HANDLE.
IF GWL-RC NOT = TDS-OK THEN
    PERFORM 910-ERR-PROCESS THRU 910-EXIT
END-IF.

*==> ACCEPT CLIENT REQUEST <===*

CALL 'TDACCEPT' USING GWL-PROC, GWL-RC, GWL-INIT-HANDLE,
                    SNA-CONN-NAME, SNA-SUBC.
IF GWL-RC NOT = TDS-OK THEN
    PERFORM 910-ERR-PROCESS THRU 910-EXIT
END-IF.

*==> VERIFY PROGRAM INITIATED BY AN RPC REQUEST... <===*

CALL 'TDINFRPC' USING GWL-PROC,          GWL-RC,
                    GWL-REQUEST-TYP, GWL-RPC-NAME,
                    GWL-COMM-STATE.
IF GWL-RC          NOT = TDS-OK   OR
   GWL-REQUEST-TYP NOT = TDS-RPC-EVENT
THEN
    MOVE GWL-RC          TO WS-MSG-RC
    MOVE 'TDINFRPC'     TO WS-MSG-FUNC
    PERFORM 920-SEND-MESSAGE THRU 920-EXIT
    PERFORM 910-ERR-PROCESS THRU 910-EXIT
END-IF.

100-EXIT.
EXIT.
```

```

200-PROCESS-INPUT.
*****
* RECEIVE THE INPUT PARAMETER INTO HOST VARIABLE
*****

*---> Find out how many parms are being passed <---*

      CALL 'TDNUMPRM' USING GWL-PROC, GWL-NUMPRM-PARMS.

*---> More than one, pump back a message      <---*

      IF GWL-NUMPRM-PARMS not = +1 THEN
          MOVE 'Invalid Number of Parameters'
              TO WS-MSG-TEXT
          MOVE GWL-RC          TO WS-MSG-RC
          MOVE 'TDNUMPRM'     TO WS-MSG-FUNC
          PERFORM 920-SEND-MESSAGE THRU 920-EXIT
          PERFORM 910-ERR-PROCESS THRU 910-EXIT
      END-IF

*---> Get that parm info into the host variable <---*

      IF GWL-NUMPRM-PARMS = +1 THEN
          CALL 'TDRCVPRM' USING GWL-PROC, GWL-RC,
                               GWL-NUMPRM-PARMS,
                               WS-INPUT-DATA,
                               TDSLONGVARCHAR,
                               WS-INPUT-LEN,
                               GWL-ACTUAL-LEN
          IF GWL-RC NOT = TDS-OK THEN
              MOVE GWL-RC          TO WS-MSG-RC
              MOVE 'TDRCVPRM'     TO WS-MSG-FUNC
              PERFORM 920-SEND-MESSAGE THRU 920-EXIT
              PERFORM 910-ERR-PROCESS THRU 910-EXIT
          END-IF
      END-IF.

200-EXIT.
EXIT.

300-PROCESS-OUTPUT.
*****
* BREAK UP THE 10K INPUT FIELDS INTO A 50 BYTE COLUMN AND SEND
*****

      MOVE +1                                TO COLUMN-NUMBER.
      MOVE LENGTH OF WS-OUTPUT-DATA          TO HOST-LEN

```

```

                                COLUMN-LEN.
MOVE LENGTH OF WS-OUTPUT-COL-NAME      TO COLUMN-NAME-LEN.
CALL 'TDESCRIB' USING GWL-PROC,
                                GWL-RC,
                                COLUMN-NUMBER,
                                TDSCHAR,
                                HOST-LEN,
                                WS-OUTPUT-DATA,
                                TDS-ZERO,
                                TDS-FALSE,
                                TDSCHAR,
                                COLUMN-LEN,
                                WS-OUTPUT-COL-NAME,
                                COLUMN-NAME-LEN.

IF GWL-RC NOT = TDS-OK THEN
    MOVE GWL-RC          TO WS-MSG-RC
    MOVE 'TDESCRIB'      TO WS-MSG-FUNC
    PERFORM 920-SEND-MESSAGE THRU 920-EXIT
    PERFORM 910-ERR-PROCESS THRU 910-EXIT
END-IF.

PERFORM VARYING I FROM 1 BY 1 UNTIL I > GWL-ACTUAL-LEN
    COMPUTE J = J + 1
    MOVE WS-INPUT-CHAR(I)      TO WS-OUTPUT-CHAR(J)
    IF J = 50
        THEN
            PERFORM 310-SEND-ROW      THRU 310-EXIT
            MOVE ZERO                  TO J
            MOVE SPACES                TO WS-OUTPUT-DATA
        END-IF
    END-PERFORM.
    IF J > ZERO
        THEN PERFORM 310-SEND-ROW      THRU 310-EXIT.

300-EXIT.
EXIT.
310-SEND-ROW.
*****
* SEND ROW OF DATA TO CLIENT...
*****

CALL 'TDSNDROW' USING GWL-PROC, GWL-RC
IF GWL-RC NOT = TDS-OK
THEN

```

```

        MOVE GWL-RC          TO WS-MSG-RC
        MOVE 'TDSNDROW'     TO WS-MSG-FUNC
        PERFORM 920-SEND-MESSAGE THRU 920-EXIT
        PERFORM 910-ERR-PROCESS THRU 910-EXIT
    END-IF.
310-EXIT.
    EXIT.
    EJECT
900-ALL-DONE.
*****
* CLOSE CONNECTION TO CLIENT AND RETURN TO CICS...          *
*****

        CALL 'TDSNDDON' USING GWL-PROC, GWL-RC, WRK-DONE-STATUS,
                                GWL-RETURN-ROWS, TDS-ZERO, TDS-ENDRPC.
    IF GWL-RC NOT = TDS-OK THEN
        PERFORM 980-CICS-DUMP THRU 980-EXIT
        PERFORM 990-CICS-RETURN THRU 990-EXIT
    END-IF.

        CALL 'TDFREE' USING GWL-PROC, GWL-RC.
        PERFORM 990-CICS-RETURN THRU 990-EXIT.

900-EXIT.
    EXIT.

910-ERR-PROCESS.
*****
* PERFORM ALL-DONE IN A ERROR STATE                          *
*****

        MOVE ZERO          TO GWL-RETURN-ROWS.
        MOVE TDS-DONE-ERROR TO WRK-DONE-STATUS.
        PERFORM 900-ALL-DONE THRU 900-EXIT.

910-EXIT.
    EXIT.

920-SEND-MESSAGE.
*-----
SEND-ERROR-MESSAGE.
*-----
        MOVE 'N'          TO SEND-DONE-SW.
        MOVE TDS-ERROR-MSG TO MSG-TYPE.
        MOVE LENGTH OF MSG-RPC TO MSG-RPC-L.
*   Ensure we're in right state to send a message

```

```
CALL 'TDSTATUS' USING 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 AND
    GWL-STATUS-COMM = TDS-RECEIVE) THEN
    CALL 'TDSNDMSG' USING 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-IF.

920-EXIT.
EXIT.

980-CICS-DUMP.
*****
* CAUSE A CICS TRANSACTION DUMP USUALLY BECAUSE SOMETHING IS BAD *
*****
EXEC CICS
    DUMP DUMPCODE('SYZ2') NOHANDLE
END-EXEC.

980-EXIT.
EXIT.

990-CICS-RETURN.
*****
* RETURN TO CICS... *
*****

EXEC CICS
    RETURN
END-EXEC.

990-EXIT.
EXIT.
```


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 VS COBOL II, DB2, and Gateway-Library.

The client language request can be entered on line, using ISQL or another Sybase or third party front end product, or it can be coded in a DB-Library program. A corresponding DB-Library program, syl2.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 syl2.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 SYL2 rather than AMD2 as the mainframe transaction 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 SYCCSAL2

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 SYCCSAL2

This 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.

```
*@(#) syccsal2.cobol 1.1 3/17/98          */
IDENTIFICATION DIVISION.
PROGRAM-ID. SYCCSAL2.

***** SYCCSAL2 - LANGUAGE REQUEST APPLICATION - COBOL2 - CICS **
*
*  TRANID:          SYL2
*  PROGRAM:         SYCCSAL2
*  PLAN NAME:      SYL2PLAN
*  FILES:           n/a
*  TABLES:        adhoc
*
*  This program is executed via a client language request
*  from sample dblib program 'SYL2', or by SYBASE's ISQL if
*  installed. The client program must login to a transaction
*  group with SYL2 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:

* 4/90 - Created, MPM
 * 10/93 - Some restructuring, TC

ENVIRONMENT DIVISION.
 DATA DIVISION.

WORKING-STORAGE SECTION.

* DB2 SQLCA

EXEC SQL INCLUDE SQLCA END-EXEC.

* DB2 MINIMUM SQLDA FOR COBOL II

01 SQLDA.

```

02  SQLDAID                PIC X(8)          VALUE 'SQLDA' .
02  SQLDABC                PIC S9(8) COMP VALUE 60 .
02  SQLN                  PIC S9(4) COMP VALUE 1 .
02  SQLD                  PIC S9(4) COMP VALUE 0 .
02  SQLVAR .
    03  SQLTYPE            PIC S9(4) COMP .
    03  SQLLEN            PIC S9(4) COMP .
    03  SQLDATA           POINTER .
    03  SQLIND            POINTER .
    03  SQLNAME .
        49  SQLNAMEL      PIC S9(4) COMP .
        49  SQLNAMEC     PIC X(30) .

```

```

*-----
*  SERVER LIBRARY COBOL COPY BOOK
*-----
COPY SYGWCOB .

```

```

*-----
*  WORK AREAS
*-----

```

```

01  GW-LIB-MISC-FIELDS .
    05  GWL-PROC          POINTER .
    05  GWL-INIT-HANDLE  POINTER .
    05  GWL-RC           PIC S9(9) COMP .
    05  GWL-SQLLEN      PIC S9(9) COMP .
    05  GWL-STATUS-NR   PIC S9(9) COMP .
    05  GWL-STATUS-DONE PIC S9(9) COMP .
    05  GWL-STATUS-COUNT PIC S9(9) COMP .
    05  GWL-STATUS-COMM PIC S9(9) COMP .
    05  GWL-STATUS-RETURN-CODE PIC S9(9) COMP .
    05  GWL-STATUS-SUBCODE PIC S9(9) COMP .
    05  GWL-INFPGM-TDS-VERSION PIC S9(9) COMP .
    05  GWL-INFPGM-LONGVAR PIC S9(9) COMP .
    05  GWL-INFPGM-ROW-LIMIT PIC S9(9) COMP .
    05  GWL-INFPGM-REMOTE-TRACE PIC S9(9) COMP .
    05  GWL-INFPGM-CORRELATOR PIC S9(9) COMP .
    05  GWL-INFPGM-DB2GW-OPTION PIC S9(9) COMP .
    05  GWL-INFPGM-DB2GW-PID PIC X(8) .
    05  GWL-INFPGM-TYPE-RPC PIC S9(9) COMP .
    05  GWL-INFLOG-GLOBAL PIC S9(9) COMP .
    05  GWL-INFLOG-API   PIC S9(9) COMP .
    05  GWL-INFLOG-TDS-HEADER PIC S9(9) COMP .
    05  GWL-INFLOG-TDS-DATA PIC S9(9) COMP .
    05  GWL-INFLOG-TRACE-ID PIC S9(9) COMP .
    05  GWL-INFLOG-FILENAME PIC X(8) .

```

```

05  GWL-INFLOG-TOTAL-RECS  PIC S9(9) COMP.
05  GWL-SETSPT-TRACE-LEVEL PIC S9(9) COMP VALUE 4.
05  GWL-SETSPT-RPC-NAME   PIC X(4)      VALUE 'SYL2'.
05  GWL-SETSPT-RPC-NAME-L PIC S9(9) COMP VALUE 4.

01  LANGUAGE-FIELDS.
    05  LANG-MAX-L          PIC S9(9) COMP.
    05  LANG-ACTUAL-L      PIC S9(9) COMP.
    05  LANG-TEXT-SS       PIC S9(4) COMP.

01  LANG-BUFFER.
    49  LANG-BUFFER-LL     PIC S9(4) COMP.
    49  LANG-BUFFER-TEXT   PIC X(1024).

01  PARSESQL-BUFFER       REDEFINES LANG-BUFFER.
    05  PARSESQL-TEXT.
        10  PARSESQL-TEXT-LL PIC S9(4) COMP.
        10  PARSESQL-TEXT-CHARS OCCURS 1024 TIMES
            PIC X.
    05  PARSESQL-TEXT-DUMMY-LVL PIC X.

01  SNA-FIELDS.
    05  SNA-SUBC           PIC S9(9) COMP.
    05  SNA-CONNECTION-NAME PIC X(8)  VALUE SPACES.

01  PARSE-FIELDS.
    05  PARSE-PTR          PIC S9(4) COMP VALUE 0.
    05  PARSE-TOKEN        PIC X(18) VALUE SPACES.
    05  PARSE-FROM         PIC X(04).
    05  PARSE-TABLE        PIC X(46).
    05  PARSE-CORRELATION  PIC X(18) VALUE SPACES.
    05  PARSE-WHERE        PIC X(05) VALUE SPACES.

01  WORK-FIELDS.
    05  WRK-DONE-STATUS    PIC S9(9) COMP.

01  MESSAGE-FIELDS.
    05  MSG-TYPE           PIC S9(9) COMP.
    05  MSG-SEVERITY       PIC S9(9) COMP.
    05  MSG-SEVERITY-OK    PIC S9(9) COMP VALUE 9.
    05  MSG-SEVERITY-ERROR PIC S9(9) COMP VALUE 11.
    05  MSG-NR             PIC S9(9) COMP.
    05  MSG-NR-OK         PIC S9(9) COMP VALUE 1.
    05  MSG-NR-ERROR      PIC S9(9) COMP VALUE 2.
    05  MSG-RPC           PIC X(4)      VALUE 'SYL2'.
    05  MSG-RPC-L         PIC S9(9) COMP.

```

```

05 MSG-TEXT                PIC X(50) .
05 MSG-TEXT-L              PIC S9(9) COMP.
05 MSG-SQL-ERROR.
    10 MSG-SQL-ERROR-T      PIC X(31)
        VALUE 'Invalid sql request, sqlcode = '.
    10 MSG-SQL-ERROR-C      PIC -9(3) DISPLAY.
05 MSG-SELECT              PIC X(24)
    VALUE 'SQL select not supported'.
05 MSG-NOT-LANG            PIC X(35)
    VALUE 'SYL2 not begun via language request'.
05 MSG-BAD-LEN             PIC X(31)
    VALUE 'Request has too many characters'.
05 MSG-NO-WHERE           PIC X(26)
    VALUE 'Delete has no where clause'.
05 MSG-OK                  PIC X(22)
    VALUE 'Execute was successful'.
05 MSG-NOT-OK.
    10 FILLER                PIC X(26)
        VALUE 'Execute failed, sqlcode = '.
    10 MSG-NOT-OK-C         PIC -9(3) DISPLAY.
    10 FILLER                PIC X(18)
        VALUE ', ROLLBACK issued.'.

01 CICS-FIELDS.
05 CICS-RESPONSE          PIC S9(9) COMP.

01 SWITCHES.
05 TRACING-SET-SW         PIC S9(9) COMP VALUE 0.
    88 TRACING-RESET              VALUE 0.
    88 TRACING-SET                VALUE 1.
05 SEND-DONE-SW          PIC X          VALUE 'Y'.
    88 SEND-DONE-ERROR            VALUE 'N'.
    88 SEND-DONE-OK              VALUE 'Y'.

*-----
*   DECLARE STATEMENT AND CURSOR
*-----
EXEC SQL DECLARE S1 STATEMENT END-EXEC.
EXEC SQL DECLARE C1 CURSOR FOR S1 END-EXEC.

*****
PROCEDURE DIVISION.
*****

*   Reset db2 error handlers

```

```

EXEC SQL WHENEVER SQLWARNING CONTINUE END-EXEC.
EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.

*   Establish gateway environment

CALL 'TDINIT' USING DFHEIBLK, GWL-RC, GWL-INIT-HANDLE.

*   Turn on local tracing if not on globally or locally

CALL 'TDINFLOG' USING 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 NOT = TDS-TRACE-ALL-RPCS
AND GWL-INFLOG-GLOBAL NOT = TDS-TRACE-SPECIFIC-RPCS THEN
    MOVE 1 TO TRACING-SET-SW
    PERFORM LOCAL-TRACING
END-IF.

*   Accept client request

CALL 'TDACCEPT' USING 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' USING 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 NOT = TDS-START-SQL
    MOVE MSG-NOT-LANG TO MSG-TEXT

```

```
        MOVE LENGTH OF MSG-NOT-LANG TO MSG-TEXT-L
        PERFORM SEND-ERROR-MESSAGE
        GO TO END-PROGRAM
    END-IF.

*   Prepare for receive

    CALL 'TDRESULT' USING GWL-PROC, GWL-RC.

*   Get lenth of language text, ensure not too big for us
*   (this could be handled without TDSQLEN by checking
*   LANG-ACTUAL-LEN doesn't exceed LANG-MAX-L in TDRCVSQL call)

    CALL 'TDSQLEN' USING GWL-PROC, GWL-SQLEN.
    MOVE LENGTH OF LANG-BUFFER-TEXT TO LANG-MAX-L.

    IF GWL-SQLEN > LANG-MAX-L THEN
        MOVE MSG-BAD-LEN          TO MSG-TEXT
        MOVE LENGTH OF MSG-BAD-LEN TO MSG-TEXT-L
        PERFORM SEND-ERROR-MESSAGE
        GO TO END-PROGRAM
    END-IF.

*   Get language text

    CALL 'TDRCVSQL' USING GWL-PROC, GWL-RC,
                        LANG-BUFFER-TEXT,
                        LANG-MAX-L,
                        LANG-ACTUAL-L.

    MOVE LANG-ACTUAL-L TO LANG-BUFFER-LL.

*   Ensure line feeds, low-values, etc. translated to blanks

    PERFORM VARYING LANG-TEXT-SS FROM 1 BY 1
        UNTIL LANG-TEXT-SS > PARSESQL-TEXT-LL

        IF PARSESQL-TEXT-CHARS(LANG-TEXT-SS) < SPACE THEN
            MOVE SPACE TO PARSESQL-TEXT-CHARS(LANG-TEXT-SS)
        END-IF

*   Save position of first non-blank

    IF PARSE-PTR = 0 AND
        PARSESQL-TEXT-CHARS(LANG-TEXT-SS) > SPACE THEN
        MOVE LANG-TEXT-SS TO PARSE-PTR
```



```

        END-IF

    END-PERFORM.

*   Let DB2 edit and tell us if SELECT

    EXEC SQL PREPARE S1 INTO SQLDA FROM :LANG-BUFFER END-EXEC.

    IF SQLD NOT = 0 THEN
        MOVE MSG-SELECT           TO MSG-TEXT
        MOVE LENGTH OF MSG-SELECT TO MSG-TEXT-L
        PERFORM SEND-ERROR-MESSAGE
        GO TO END-PROGRAM
    END-IF.

    IF SQLCODE < 0 THEN
        MOVE SQLCODE              TO MSG-SQL-ERROR-C
        MOVE MSG-SQL-ERROR        TO MSG-TEXT
        MOVE LENGTH OF MSG-SQL-ERROR TO MSG-TEXT-L
        PERFORM SEND-ERROR-MESSAGE
        GO TO END-PROGRAM
    END-IF.

*   Parse and handle special case of DELETE without WHERE clause

    UNSTRING LANG-BUFFER-TEXT DELIMITED BY ALL ' '
        INTO PARSE-TOKEN
            PARSE-FROM
            PARSE-TABLE
            PARSE-CORRELATION
            PARSE-WHERE
        POINTER PARSE-PTR.

    PERFORM XLATE-TOKEN-UPPERCASE.

    IF PARSE-TOKEN = 'DELETE' THEN
        MOVE PARSE-CORRELATION TO PARSE-TOKEN
        PERFORM XLATE-TOKEN-UPPERCASE
        MOVE PARSE-TOKEN TO PARSE-CORRELATION

        MOVE PARSE-WHERE TO PARSE-TOKEN
        PERFORM XLATE-TOKEN-UPPERCASE

        IF PARSE-CORRELATION NOT = 'WHERE ' AND
           PARSE-TOKEN          NOT = 'WHERE ' THEN
            MOVE MSG-NO-WHERE           TO MSG-TEXT

```

```

                MOVE LENGTH OF MSG-NO-WHERE TO MSG-TEXT-L
                PERFORM SEND-ERROR-MESSAGE
                GO TO END-PROGRAM
            END-IF
        END-IF.

*   Execute the SQL statement

        EXEC SQL EXECUTE S1 END-EXEC.

        IF SQLCODE < 0 THEN
            PERFORM CICS-ROLLBACK
            MOVE SQLCODE                TO MSG-NOT-OK-C
            MOVE MSG-NOT-OK              TO MSG-TEXT
            MOVE LENGTH OF MSG-NOT-OK TO MSG-TEXT-L
            PERFORM SEND-ERROR-MESSAGE
            GO TO END-PROGRAM
        END-IF.

        MOVE MSG-OK                    TO MSG-TEXT.
        MOVE LENGTH OF MSG-OK TO MSG-TEXT-L.
        PERFORM SEND-CONFIRM-MESSAGE.
        GO TO END-PROGRAM.

*-----
        XLATE-TOKEN-UPPERCASE.
*-----

*   All we care about is DELETE and WHERE

        INSPECT PARSE-TOKEN REPLACING ALL 'd' BY 'D'
                                           'e' BY 'E'
                                           'h' BY 'H'
                                           'l' BY 'L'
                                           'r' BY 'R'
                                           't' BY 'T'
                                           'w' BY 'W'.

*-----
        SEND-CONFIRM-MESSAGE.
*-----

        MOVE MSG-SEVERITY-OK TO MSG-SEVERITY.
        MOVE MSG-NR-OK       TO MSG-NR.
        MOVE TDS-INFO-MSG   TO MSG-TYPE.
        PERFORM SEND-MESSAGE.
    
```

```

*-----
SEND-ERROR-MESSAGE.
*-----
      MOVE 'N'                TO SEND-DONE-SW.
MOVE MSG-SEVERITY-ERROR TO MSG-SEVERITY.
MOVE MSG-NR-ERROR      TO MSG-NR.
MOVE TDS-ERROR-MSG     TO MSG-TYPE.
PERFORM SEND-MESSAGE.

*-----
SEND-MESSAGE.
*-----
      MOVE LENGTH OF MSG-RPC TO MSG-RPC-L.

*
* -----
* ensure we're in right state to send a message
* -----
      CALL 'TDSTATUS' USING 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 AND
          GWL-STATUS-COMM = TDS-RECEIVE) THEN

          CALL 'TDSNDMSG' USING 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-IF.

*-----
LOCAL-TRACING.
*-----
      CALL 'TDSETSPT' USING GWL-INIT-HANDLE, GWL-RC,
                          TRACING-SET-SW,
                          GWL-SETSPT-TRACE-LEVEL,
                          GWL-SETSPT-RPC-NAME,
                          GWL-SETSPT-RPC-NAME-L.

```

```
*-----
CICS-ROLLBACK.
*-----
      EXEC CICS SYNCPOINT
              ROLLBACK
              RESP(CICS-RESPONSE)
      END-EXEC.

*-----
END-PROGRAM.
*-----

      IF TRACING-SET
          MOVE 0 TO TRACING-SET-SW
          PERFORM LOCAL-TRACING
      END-IF.

      IF SEND-DONE-OK
          MOVE TDS-DONE-COUNT TO WRK-DONE-STATUS
      ELSE
          MOVE TDS-DONE-ERROR TO WRK-DONE-STATUS
          MOVE ZERO           TO SQLERRD(3)
      END-IF.

      CALL 'TDSNDDON' USING GWL-PROC, GWL-RC,
                          WRK-DONE-STATUS,
                          SQLERRD(3),
                          TDS-ZERO,
                          TDS-ENDRPC.

      CALL 'TDFREE' USING GWL-PROC, GWL-RC.
      EXEC CICS RETURN END-EXEC.
```

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 SYD2. The COBOL 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. It uses DB2, VS COBOL II and Gateway-Library. It references a DB2 sample table, SYBASE.SAMPLETB, which is provided with the product.

Sample program SYICSAD2

```

IDENTIFICATION DIVISION.
*-----
PROGRAM-ID. SYICSAD2.

***** SYICSAD2 - RPC REQUEST APPLICATION - COBOL2 - IMS *****
*
* TRANID: SYD2
* PROGRAM: SYICSAD2
* PLAN NAME: SYICSAD2
* FILES: n/a
* TABLES: SYBASE.SAMPLETB
*
* This program is executed via a client RPC request from sample

```

```
* dblib program 'SYD2' or from isql. The program expects one sample
* character parm which is equal to a department number in the DB2
* table SYBASE.SAMPLETB. The program then selects and returns all
* rows with that department number.
```

```
*
* To execute from isql type:
```

```
* >isql -Usa -Sservername
```

```
* >exec SYD2 'D11'
```

```
* >go
```

```
* NOTE: Add SYD2 using isql as follows:
```

```
* exec sgw_addrpc SYD2,SYD2,IMSLU62,none
```

```
* where IMSLU62 is the APPC name of your IMS region.
```

```
* Server Library calls:
```

```
* TDACCEPT      accept request from client
* TDESCRIB     describe a column
* TDFREE       free TDPROC structure
* TDGETREQ     get next set of parms
* TDINIT       establish environment
* TDRCVPRM    retrieve rpc parameter from client
* TDSNDDON    send results-completion to client
* TDSNDMSG    send message to client
* TDSNDROW    send row to client
* TDSTATUS    get status information
* TDSETPT     pass type of program to gwlib
* TDTERM      clean up control blocks
```

```
*-----*
```

```
ENVIRONMENT DIVISION.
DATA DIVISION.
```

```
*****
```

```
WORKING-STORAGE SECTION.
```

```
*****
```

```
*-----*
```

```
* DB2 SQLCA
```

```
*-----*
```

```
EXEC SQL INCLUDE SQLCA END-EXEC.
```

```

* -----
* SERVER LIBRARY COBOL COPY BOOK
* -----
COPY SYGWCOB.

* -----
* WORK AREAS
* -----
01 GW-LIB-MISC-FIELDS.
    05 GWL-SPA-PTR          POINTER.
    05 GWL-PROC            POINTER.
    05 GWL-INIT-HANDLE     POINTER.
    05 GWL-RC              PIC S9(9) COMP.
    05 GWL-REQ-TYPE        PIC S9(9) COMP VALUE +0.
    05 GWL-WAIT-OPTION     PIC S9(9) COMP.
    05 GWL-STATUS-NR       PIC S9(9) COMP.
    05 GWL-STATUS-DONE     PIC S9(9) COMP.
    05 GWL-STATUS-COUNT    PIC S9(9) COMP.
    05 GWL-STATUS-COMM     PIC S9(9) COMP.
    05 GWL-STATUS-RETURN-CODE PIC S9(9) COMP.
    05 GWL-STATUS-SUBCODE  PIC S9(9) COMP.
    05 GWL-PROG-TYPE       PIC X(04) VALUE 'MPP '.
    05 GWL-RPC-NAME        PIC X(30) VALUE SPACES.

01 PARM-FIELDS.
    05 PARM-L              PIC S9(9) COMP.
    05 PARM-ID1            PIC S9(9) COMP VALUE 1.
    05 PARM-DEPT          PIC X(3) .

01 SNA-FIELDS.
    05 SNA-SUBC            PIC S9(9) COMP
    05 SNA-CONNECTION-NAME PIC X(8) VALUE SPACES.

01 EMPLOYEE-FIELDS.
    05 EMPLOYEE-LNM.
        49 EMPLOYEE-LNM-LEN PIC S9(4) COMP.
        49 EMPLOYEE-LNM-TEXT PIC X(15) .
    05 EMPLOYEE-DEPT      PIC X(3) .
    05 EMPLOYEE-PH        PIC X(4) .
    05 EMPLOYEE-SALARY    PIC S9(6)V9(2) COMP-3.

01 COLUMN-NAME-FIELDS.
    05 CN-LNM              PIC X(10) VALUE 'LAST_NAME '.
    05 CN-DEPT             PIC X(8)  VALUE 'EMP_DEPT' .
    05 CN-PH               PIC X(9)  VALUE 'EMP_PHONE' .

```

```

05 CN-SALARY                PIC X(6)  VALUE 'SALARY' .

01 DESCRIBE-BIND-FIELDS .
05 DB-HOST-TYPE            PIC S9(9)  COMP .
05 DB-CLIENT-TYPE        PIC S9(9)  COMP .
05 DB-NULL-INDICATOR     PIC S9(4)  COMP VALUE 0 .

01 COUNTER-FIELDS .
05 CTR-COLUMN            PIC S9(9)  COMP VALUE 1 .
05 CTR-ROWS              PIC S9(9)  COMP VALUE 0 .

01 WORK-FIELDS .
05 WRKLEN1                PIC S9(9)  COMP .
05 WRKLEN2                PIC S9(9)  COMP .
05 WRK-DONE-STATUS       PIC S9(9)  COMP .

01 MESSAGE-FIELDS .
05 MSG-TYPE                PIC S9(9)  COMP .
05 MSG-SEVERITY           PIC S9(9)  COMP VALUE 11 .
05 MSG-NR                  PIC S9(9)  COMP VALUE 2 .
05 MSG-RPC                 PIC X(4)  VALUE 'SYD2' .
05 MSG-RPC-L              PIC S9(9)  COMP .
05 MSG-TEXT                PIC X(100) .
05 MSG-TEXT-L             PIC S9(9)  COMP .
05 MSG-BAD-CURSOR        PIC X(27)
    VALUE 'ERROR - can not open cursor' .
05 MSG-BAD-FETCH          PIC X(24)
    VALUE 'ERROR - fetch row failed' .
05 MSG-SQL-ERROR .
    10 FILLER                PIC X(10) VALUE 'Sqlcode = ' .
    10 MSG-SQL-ERROR-C     PIC -9(3) DISPLAY .
    10 FILLER                PIC X(16)
    VALUE ', Error Tokens: ' .
    10 MSG-SQL-ERROR-K     PIC X(70) .
    10 MSG-SQL-ERROR-K-CHARS
    REDEFINES MSG-SQL-ERROR-K
    OCCURS 70 TIMES
    PIC X .
05 MSG-SQL-ERROR-SS       PIC S9(4)  COMP .

01 CALL-ERROR-MESSAGE .
05 FILLER                  PIC X(5)  VALUE SPACES .
05 CALL-PROG               PIC X(10) VALUE 'SYICSAD2' .
05 FILLER                  PIC X(5)  VALUE SPACES .
05 CALL-ERROR              PIC X(10) VALUE SPACES .
05 FILLER                  PIC X(5)  VALUE ' RC= ' .

```



```

05 CALL-ERROR-RC          PIC -ZZZZ.
01 SWITCHES.
05 ALL-DONE-SW            PIC X    VALUE 'N' .
   88 NOT-ALL-DONE        VALUE 'N' .
   88 ALL-DONE            VALUE 'Y' .
05 SEND-DONE-SW          PIC X    VALUE 'Y' .
   88 SEND-DONE-ERROR     VALUE 'N' .
   88 SEND-DONE-OK        VALUE 'Y' .

*-----
*  DECLARE CURSOR
*-----
EXEC SQL
  DECLARE ECURSOR CURSOR
    FOR SELECT LASTNAME,
              WORKDEPT, PHONENO, SALARY
  FROM SYBASE.SAMPLETB
  WHERE WORKDEPT = :PARAM-DEPT

END-EXEC.

LINKAGE SECTION.

01 IO-PCB.
05 LTERM-NAME            PIC X(8) .
05 TERM-RESERVE          PIC XX.
05 TERM-STATSUS         PIC XX.
05 TERM-PREFIX.
   15 FILLER              PIC X.
   15 JULIAN-DATE         PIC S9(5) COMP-3.
   15 TIME-O-DAY         PIC S9(7) COMP-3.
   15 FILLER              PIC XXXX.
05 MODNAME               PIC X(08) .

*****
PROCEDURE DIVISION.
*****

  ENTRY 'DLITCBL' USING IO-PCB.

*-----
INITIALIZE-PROGRAM.
*-----

  SET GWL-SPA-PTR TO NULL.

*-----

```

```
* reset db2 error handlers
* -----
EXEC SQL WHENEVER SQLWARNING CONTINUE END-EXEC.
EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.

* -----
* establish gateway environment
* -----
CALL 'TDINIT' USING IO-PCB, GWL-RC, GWL-INIT-HANDLE.

IF GWL-RC NOT EQUAL TO ZEROES THEN
    MOVE 'TDINIT' TO CALL-ERROR
    PERFORM DISPLAY-CALL-ERROR
END-IF.

* -----
* Set program type
* -----
CALL 'TDSETPT' USING GWL-INIT-HANDLE, GWL-RC, GWL-PROG-TYPE
                GWL-SPA-PTR, TDS-NULL, TDS-NULL.

IF GWL-RC NOT EQUAL TO ZEROES THEN
    MOVE 'TDSETPT' TO CALL-ERROR
    PERFORM DISPLAY-CALL-ERROR
END-IF.

* -----
* accept client request
* -----

CALL 'TDACCEPT' USING GWL-PROC, GWL-RC, GWL-INIT-HANDLE,
                SNA-CONNECTION-NAME,
                SNA-SUBC.

IF GWL-RC NOT EQUAL TO ZEROES THEN
    MOVE 'TDACCEPT' TO CALL-ERROR
    PERFORM DISPLAY-CALL-ERROR
END-IF.

* -----
READ-IN-USER-PARM.
* -----
MOVE 'Y' TO SEND-DONE-SW.
```

```
MOVE 'N' TO ALL-DONE-SW.
MOVE SPACES TO CALL-ERROR.
MOVE ZEROES TO CALL-ERROR-RC CTR-ROWS.
MOVE 1 TO CTR-COLUMN.
```

```
MOVE LENGTH OF PARM-DEPT TO WRKLEN1.
```

```
CALL 'TDRCVPRM' USING GWL-PROC, GWL-RC,
    PARM-ID1,
    PARM-DEPT,
    TDSCHAR,
    WRKLEN1,
    PARM-L.
```

```
IF GWL-RC NOT EQUAL TO ZEROES THEN
    MOVE 'TDRCVPRM' TO CALL-ERROR
    PERFORM DISPLAY-CALL-ERROR
END-IF.
```

```
*-----
    OPEN-DB2-CURSOR.
```

```
*-----
EXEC SQL OPEN ECURSOR END-EXEC.
```

```
IF SQLCODE NOT = 0
    DISPLAY 'SQLCODE = ' SQLCODE
    PERFORM OPEN-ERROR
GO TO FINISH-REPLY
END-IF.
```

```
*-----
    SETUP-REPLY-COLUMNS.
```

```
*-----
MOVE TDSVARYCHAR TO DB-HOST-TYPE.
MOVE TDSCHAR TO DB-CLIENT-TYPE.
MOVE LENGTH OF EMPLOYEE-LNM-TEXT TO WRKLEN1.
MOVE LENGTH OF CN-LNM TO WRKLEN2.
```

```
CALL 'TDESCRIB' USING GWL-PROC, GWL-RC,
    CTR-COLUMN,
    DB-HOST-TYPE,
    WRKLEN1,
    EMPLOYEE-LNM,
    DB-NUL-INDICATOR,
    TDS-FALSE,
    DB-CLIENT-TYPE,
```

```
WRKLEN1 ,  
CN-LNM ,  
WRKLEN2 .
```

```
IF GWL-RC NOT EQUAL TO ZEROES THEN  
  MOVE 'TDESCRIB' TO CALL-ERROR  
  PERFORM DISPLAY-CALL-ERROR  
END-IF.
```

```
ADD 1 TO CTR-COLUMN.  
MOVE TDSCHAR TO DB-HOST-TYPE.  
MOVE TDSCHAR TO DB-CLIENT-TYPE.  
MOVE LENGTH OF EMPLOYEE-DEPT TO WRKLEN1 .  
MOVE LENGTH OF CN-DEPT TO WRKLEN2 .
```

```
CALL 'TDESCRIB' USING GWL-PROC, GWL-RC,  
  CTR-COLUMN,  
  DB-HOST-TYPE,  
  WRKLEN1 ,  
  EMPLOYEE-DEPT ,  
  DB-NULL-INDICATOR ,  
  TDS-FALSE ,  
  DB-CLIENT-TYPE ,  
  WRKLEN1 , CN-DEPT ,WRKLEN2 .
```

```
IF GWL-RC NOT EQUAL TO ZEROES THEN  
  MOVE 'TDESCRIB' TO CALL-ERROR  
  PERFORM DISPLAY-CALL-ERROR  
END-IF.
```

```
ADD 1 TO CTR-COLUMN.  
MOVE LENGTH OF EMPLOYEE-PH TO WRKLEN1 .  
MOVE LENGTH OF CN-PH TO WRKLEN2 .
```

```
CALL 'TDESCRIB' USING GWL-PROC, GWL-RC,  
  CTR-COLUMN,  
  DB-HOST-TYPE,  
  WRKLEN1 ,  
  EMPLOYEE-PH ,  
  DB-NULL-INDICATOR ,  
  TDS-FALSE ,  
  DB-CLIENT-TYPE ,
```

```

WRKLEN1, CN-PH, WRKLEN2.

IF GWL-RC NOT EQUAL TO ZEROES THEN
  MOVE 'TDESCRIB' TO CALL-ERROR
  PERFORM DISPLAY-CALL-ERROR
END-IF.

* -----
* Here we let TDESCRIB convert from TDSDECIMAL to TDSMONEY.
* Note we're taking the default scaling (2) for TDSDECIMAL
* input, though we could override with TDSETBCD if necessary.
* -----
ADD 1 TO CTR-COLUMN.
MOVE LENGTH OF EMPLOYEE-SALARY TO WRKLEN1.
MOVE LENGTH OF CN-SALARY TO WRKLEN2.

CALL 'TDESCRIB' USING GWL-PROC, GWL-RC, CTR-COLUMN,
TDSDECIMAL,
  WRKLEN1,
  EMPLOYEE-SALARY,
  DB-NULL-INDICATOR,
  TDS-FALSE,
  TDSMONEY,
  TDS-DEFAULT-LENGTH,
  CN-SALARY, WRKLEN2.
IF GWL-RC NOT EQUAL TO ZEROES THEN
  MOVE 'TDESCRIB' TO CALL-ERROR
  PERFORM DISPLAY-CALL-ERROR
END-IF.

* -----
  SEND-ROWS.
* -----
PERFORM FETCH-AND-SEND-ROWS
  UNTIL ALL-DONE.

  FINISH-REPLY.
* -----
* close cursor
* -----
EXEC SQL CLOSE ECURSOR END-EXEC.

IF SEND-DONE-OK
  MOVE TDS-DONE-COUNT TO WRK-DONE-STATUS
ELSE
  MOVE TDS-DONE-ERROR TO WRK-DONE-STATUS

```

```
MOVE ZERO TO CTR-ROWS
END-IF.
```

```
CALL 'TDSNDDON' USING GWL-PROC, GWL-RC,
      WRK-DONE-STATUS, CTR-ROWS, TDS-ZERO, TDS-ENDRPC.
```

```
IF GWL-RC NOT EQUAL TO ZEROES THEN
  MOVE 'TDSNDDON' TO CALL-ERROR
  PERFORM DISPLAY-CALL-ERROR
END-IF.
```

```
* -----
* Get next client request
* -----
```

```
MOVE TDS-TRUE TO GWL-WAIT-OPTION.
MOVE ZEROES TO GWL-REQ-TYPE.
MOVE SPACES TO GWL-RPC-NAME.
CALL 'TDGETREQ' USING GWL-PROC, GWL-RC, GWL-WAIT-OPTION,
      GWL-REQ-TYPE, GWL-TRAN-NAME.
```

```
EVALUATE GWL-RC
  WHEN ZEROES
    GO TO READ-IN-USER-PARM
  WHEN TDS-RESULTS-COMPLETE
    PERFORM FREE-ALL-STORAGE
  WHEN TDS-CONNECTION-TERMINATED
    PERFORM FREE-ALL-STORAGE
  WHEN OTHER
    MOVE 'TDGETREQ' TO CALL-ERROR
    PERFORM DISPLAY-CALL-ERROR
END-EVALUATE.
```

```
GOBACK.
```

```
* -----
* FETCH-AND-SEND-ROWS.
* -----
```

```
EXEC SQL FETCH ECURSOR INTO :EMPLOYEE-FIELDS
END-EXEC.
```

```
IF SQLCODE = 0 THEN
```

```
* -----
* send a row to the client
* -----
```

```
CALL 'TDSNDROW' USING GWL-PROC, GWL-RC
```

```

ADD 1 TO CTR-ROWS

IF GWL-RC = TDS-CANCEL-RECEIVED THEN
    MOVE 'Y' TO ALL-DONE-SW
ELSE
IF GWL-RC NOT EQUAL TO ZEROES THEN
    PERFORM DISPLAY-CALL-ERROR
    MOVE 'Y' TO ALL-DONE-SW
END-IF

ELSE IF SQLCODE = +100 THEN
    MOVE 'Y' TO ALL-DONE-SW

ELSE IF SQLCODE < 0 THEN
    MOVE 'Y' TO ALL-DONE-SW
    PERFORM FETCH-ERROR
END-IF.

* -----
    DISPLAY-CALL-ERROR.
* -----

MOVE GWL-RC TO CALL-ERROR-RC.
MOVE CALL-ERROR-MESSAGE TO MSG-TEXT.
MOVE LENGTH OF CALL-ERROR-MESSAGE TO MSG-TEXT-L.
PERFORM SEND-MESSAGE.
DISPLAY CALL-ERROR-MESSAGE.
PERFORM FREE-ALL-STORAGE.
GOBACK.

* -----
FREE-ALL-STORAGE.
* -----

CALL 'TDFREE' USING GWL-PROC, GWL-RC.

IF GWL-RC NOT EQUAL TO ZEROES THEN
    MOVE GWL-RC TO CALL-ERROR-RC
MOVE 'TDFREE' TO CALL-ERROR
DISPLAY CALL-ERROR-MESSAGE
END-IF.

CALL 'TDTERM' USING GWL-INIT-HANDLE, GWL-RC.

IF GWL-RC NOT EQUAL TO ZEROES THEN
    MOVE GWL-RC TO CALL-ERROR-RC

```

```

MOVE 'TDTERM' TO CALL-ERROR
DISPLAY CALL-ERROR-MESSAGE
END-IF.

```

```

* -----
OPEN-ERROR.

```

```

* -----
MOVE MSG-BAD-CURSOR TO MSG-TEXT.
MOVE LENGTH OF MSG-BAD-CURSOR TO MSG-TEXT-L.
PERFORM SEND-ERROR-MESSAGE.
PERFORM SEND-SQL-ERROR.

```

```

* -----
FETCH-ERROR.

```

```

* -----
MOVE MSG-BAD-FETCH TO MSG-TEXT.
MOVE LENGTH OF MSG-BAD-FETCH TO MSG-TEXT-L.
PERFORM SEND-ERROR-MESSAGE.
PERFORM SEND-SQL-ERROR.

```

```

* -----
SEND-SQL-ERROR.

```

```

* -----
MOVE SQLCODE TO MSG-SQL-ERROR-C.
MOVE SQLERRMC TO MSG-SQL-ERROR-K.

```

```

* -----
* ensure possible non-printables translated to spaces
* -----

```

```

PERFORM VARYING MSG-SQL-ERROR-SS FROM 1 BY 1
UNTIL MSG-SQL-ERROR-SS > SQLERRML

```

```

IF MSG-SQL-ERROR-K-CHARS(MSG-SQL-ERROR-SS) < SPACE OR
MSG-SQL-ERROR-K-CHARS(MSG-SQL-ERROR-SS) > '9' THEN
MOVE SPACE TO MSG-SQL-ERROR-K-CHARS(MSG-SQL-ERROR-SS)
END-IF

```

```

END-PERFORM.

```

```

MOVE MSG-SQL-ERROR TO MSG-TEXT.
MOVE LENGTH OF MSG-SQL-ERROR TO MSG-TEXT-L.
PERFORM SEND-ERROR-MESSAGE.

```

```

* -----
SEND-ERROR-MESSAGE.
* -----

```



```

MOVE 'N'                TO SEND-DONE-SW.
MOVE TDS-ERROR-MSG     TO MSG-TYPE.
MOVE LENGTH OF MSG-RPC TO MSG-RPC-L.

```

* Ensure we're in right state to send a message

```

CALL 'TDSTATUS' USING 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 AND
    GWL-STATUS-COMM = TDS-RECEIVE) THEN

```

```

    CALL 'TDSNDMSG' USING 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-IF.

```


Sample RPC Application for IMS™ (Explicit)

This appendix contains a sample long-running transaction that runs under the IMS™ explicit API. This transaction processes a client RPC. The COBOL 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™ programs, particularly those designed to handle long-running transactions. 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 uses DB2, VS COBOL II and Gateway-Library.

Sample program SYIXSAM2

```

IDENTIFICATION DIVISION.
*-----
PROGRAM-ID. SYIXSAM2.

***** SYIXSAM2 - RPC REQUEST APPLICATION - COBOL2 - IMS *****
*
*  TRANID:      SYIXSAM2
*  PROGRAM:     SYIXSAM2
*  PLAN NAME:   N/A
*  FILES:       N/A
*  TABLES:     N/A
*
*  This program is an example of a long-running transaction.
*  It may also be used to stress test IMS Open Server. The
*  program is executed via isql. The first parameter is
*  a one byte character that is used to set up a reply
*  row. The second parameter is the number of rows to

```

```

*   return to the client.
*
* To execute from isql type:
*
* >isql -Usa -Sservername
*
* >exec SYIXSAM2 X, 100
*
* >go
*
* To end SYIXSAM2 type:
*
* >exec SYIXSAM2 X,0
*
* >go
*
* The SYIXSAM2 tran returns a 80 byte row containing the name
* client that initiated the RPC and a 71 byte pattern.
*
*   Server Library calls:
*
*           TDACCEPT      accept request from client
*           TDESCRIB     describe a column
*           TDFREE       free TDPROC structure
*           TDGETREQ     get next set of parms
*           TDINIT       establish environment
*           TDRCVPRM     retrieve rpc parameter from client
*           TDSNDDON     send results-completion to client
*           TDSNDMSG     send message to client
*           TDSNDROW     send row to client
*           TDSTATUS     get status information
*           TDSETPT      pass type of program to gwlib
*           TDTERM       clean up control blocks
* CHANGE ACTIVITY:
*   9/93      - created for IMS MSP
*****

ENVIRONMENT DIVISION.
DATA DIVISION.
*****

WORKING-STORAGE SECTION.
*****

*-----
*   SERVER LIBRARY COBOL COPY BOOK
*-----

```

COPY SYGWCOB.

```

*-----
*   WORK AREAS
*-----
01  GW-LIB-MISC-FIELDS.
    05  GWL-SPA-PTR          POINTER.
    05  GWL-PROC            POINTER.
    05  GWL-INIT-HANDLE    POINTER.
    05  GWL-RC              PIC S9(9) COMP VALUE +0.
    05  GWL-REQ-TYPE       PIC S9(9) COMP VALUE +0.
    05  GWL-WAIT-OPTION    PIC S9(9) COMP VALUE +0.
    05  GWL-STATUS-NR     PIC S9(9) COMP VALUE +0.
    05  GWL-STATUS-DONE   PIC S9(9) COMP VALUE +0.
    05  GWL-STATUS-COUNT  PIC S9(9) COMP VALUE +0.
    05  GWL-STATUS-COMM   PIC S9(9) COMP VALUE +0.
    05  GWL-STATUS-RETURN-CODE PIC S9(9) COMP VALUE +0.
    05  GWL-STATUS-SUBCODE    PIC S9(9) COMP VALUE +0.
    05  GWL-PROG-TYPE       PIC X(04) VALUE 'MPP '.
    05  GWL-TRAN-NAME      PIC X(30) VALUE SPACES.

01  CPIC-RC                PIC S9(9) COMP VALUE +0.

01  PARM-FIELDS.
    05  PARM-L              PIC S9(9) COMP VALUE +0.
    05  PARM-ID1           PIC S9(9) COMP VALUE 1.
    05  PARM-ID2           PIC S9(9) COMP VALUE 2.
    05  PARM-PATTERN       PIC X(1) .
    05  PARM-NR-ROWS       PIC S9(9) COMP.

01  SNA-FIELDS.
    05  SNA-SUBC           PIC S9(9) COMP VALUE +0.
    05  SNA-CONNECTION-NAME PIC X(8)  VALUE SPACES.

01  COLUMN-NAME-FIELDS.
    05  BANANA             PIC X(06) VALUE 'BANANA'.

01  DESCRIBE-BIND-FIELDS.
    05  DB-HOST-TYPE       PIC S9(9) COMP VALUE +0.
    05  DB-CLIENT-TYPE    PIC S9(9) COMP VALUE +0.
    05  DB-NUL-INDICATOR  PIC S9(4) COMP VALUE 0.

01  COUNTER-FIELDS.
    05  CTR-COLUMN        PIC S9(9) COMP VALUE 1.
    05  CTR-ROWS          PIC S9(9) COMP VALUE 0.

01  WROW.

```

```

05 WROW-LU                PIC X(09) .
05 WROW-PATTERN OCCURS 71 TIMES PIC X(01) .

01 WORK-FIELDS .
05 WRKLEN1                PIC S9(9) COMP VALUE +0 .
05 WRKLEN2                PIC S9(9) COMP VALUE +0 .
05 WRK-DONE-STATUS       PIC S9(9) COMP VALUE +0 .
05 I                      PIC S9(9) COMP VALUE +0 .

01 MESSAGE-FIELDS .
05 MSG-TYPE               PIC S9(9) COMP VALUE +0 .
05 MSG-SEVERITY          PIC S9(9) COMP VALUE 11 .
05 MSG-NR                 PIC S9(9) COMP VALUE 2 .
05 MSG-RPC                PIC X(8) VALUE 'SYIXSAM2' .
05 MSG-RPC-L              PIC S9(9) COMP VALUE +0 .
05 MSG-TEXT               PIC X(100) .
05 MSG-TEXT-L            PIC S9(9) COMP VALUE +0 .

01 CANCEL-RCV-MSG .
05 FILLER                 PIC X(40) VALUE 'CANCEL RECEIVED' .

01 CALL-ERROR-MESSAGE .
05 FILLER                 PIC X(5) VALUE SPACES .
05 CALL-PROG              PIC X(10) VALUE 'SYIXSAM2' .
05 FILLER                 PIC X(5) VALUE SPACES .
05 CALL-ERROR             PIC X(10) VALUE SPACES .
05 FILLER                 PIC X(5) VALUE ' RC= ' .
05 CALL-ERROR-RC         PIC -9999 .

01 SWITCHES .
05 ALL-DONE-SW            PIC X          VALUE 'N' .
    88 NOT-ALL-DONE          VALUE 'N' .
    88 ALL-DONE              VALUE 'Y' .
05 SEND-DONE-SW          PIC X          VALUE 'Y' .
    88 SEND-DONE-ERROR      VALUE 'N' .
    88 SEND-DONE-OK         VALUE 'Y' .

01 APSB                   PIC X(04) VALUE 'APSB' .
01 DPSB                   PIC X(04) VALUE 'DPSB' .

01 AIB .
05 AIBID                 PIC X(08) .
05 AIBLEN                PIC S9(9) COMP .
05 AIBSFUNC              PIC X(08) .

```

```

05 AIBRSNM1          PIC X(08) .
05 FILLER            PIC X(16) .
05 AIBOALEN         PIC S9(9) COMP.
05 AIBOAUSE         PIC S9(9) COMP.
05 FILLER            PIC X(12) .
05 AIBRETRN         PIC S9(9) COMP.
05 AIBREASN         PIC S9(9) COMP.
05 FILLER            PIC X(04) .
05 AIBRSA1          PIC S9(9) COMP.
05 FILLER REDEFINES AIBRSA1.
    10 AIBPTR        POINTER.
05 FILLER            PIC X(44) .

```

LINKAGE SECTION.

```

01 PCB-ADDRESSES.
    05 PCB-ADDRESS-LIST USAGE IS POINTER OCCURS 3 TIMES.

01 IO-PCB.
    05 LTERM-NAME          PIC X(8) .
    05 TERM-RESERVE       PIC XX.
    05 TERM-STATSUS       PIC XX.
    05 TERM-PREFIX.
        15 FILLER          PIC X.
        15 JULIAN-DATE     PIC S9(5) COMP-3.
        15 TIME-O-DAY      PIC S9(7) COMP-3.
        15 FILLER          PIC XXXX.
    05 MODNAME            PIC X(08) .

```

```

*****
PROCEDURE DIVISION.
*****

```

```

*-----+-----
INITIALIZE-PROGRAM.
*-----

```

PERFORM ALLOC-AIB.

```

*-----
* Establish Open Server environment
*-----
CALL 'TDINIT' USING IO-PCB, GWL-RC, GWL-INIT-HANDLE.

```

```

IF GWL-RC NOT EQUAL TO ZEROES THEN
    MOVE 'TDINIT' TO CALL-ERROR

```

```
        PERFORM DISPLAY-CALL-ERROR
END-IF.

* -----
*   Set program type
* -----

MOVE 'EXPL' TO GWL-PROG-TYPE.

CALL 'TDSETPT' USING GWL-INIT-HANDLE, GWL-RC, GWL-PROG-TYPE
                GWL-SPA-PTR, TDS-NULL, TDS-NULL.

IF GWL-RC NOT EQUAL TO ZEROES THEN
    MOVE 'TDSETPT' TO CALL-ERROR
    PERFORM DISPLAY-CALL-ERROR
END-IF.

* -----
*   accept client request
* -----

CALL 'TDACCEPT' USING GWL-PROC, GWL-RC, GWL-INIT-HANDLE,
                SNA-CONNECTION-NAME, SNA-SUBC.

IF GWL-RC NOT EQUAL TO ZEROES THEN
    MOVE 'TDACCEPT' TO CALL-ERROR
    PERFORM DISPLAY-CALL-ERROR
END-IF.

PERFORM READ-IN-USER-PARMS THRU READ-IN-EXIT
        UNTIL (GWL-RC NOT EQUAL TO ZEROES).

GOBACK.

* -----
READ-IN-USER-PARMS.
* -----
*   INITIALIZATION
* -----

MOVE 'Y' TO SEND-DONE-SW.
MOVE 'N' TO ALL-DONE-SW.
MOVE SPACES TO CALL-ERROR.
MOVE ZEROES TO CALL-ERROR-RC CTR-ROWS.
MOVE 1 TO CTR-COLUMN.
```



```

*-----
* GET PARM 1 - CHARACTER TO USE IN PATTERN
*-----
    MOVE LENGTH OF PARM-PATTERN TO WRKLEN1.

    CALL 'TDRCVPRM' USING GWL-PROC, GWL-RC,
                        PARM-ID1,
                        PARM-PATTERN,
                        TDSCHAR,
                        WRKLEN1,
                        PARM-L.

    IF GWL-RC NOT EQUAL TO ZEROES THEN
        MOVE 'TDRCVPRM-1' TO CALL-ERROR
        PERFORM DISPLAY-CALL-ERROR
    END-IF.

    MOVE BANANA TO WROW-LU.

    PERFORM SET-UP-ROW-PATTERN
        VARYING I FROM 1 BY 1
        UNTIL I > 71.

*-----
* GET PARM 2 - NUMBER OF ROWS TO SEND TO CLIENT
*-----
    MOVE LENGTH OF PARM-NR-ROWS TO WRKLEN1.

    CALL 'TDRCVPRM' USING GWL-PROC, GWL-RC,
                        PARM-ID2,
                        PARM-NR-ROWS,
                        TDSINT4,
                        WRKLEN1,
                        PARM-L.

    IF GWL-RC NOT EQUAL TO ZEROES THEN
        MOVE 'TDRCVPRM-2' TO CALL-ERROR
        PERFORM DISPLAY-CALL-ERROR
    END-IF.

    IF PARM-NR-ROWS = ZEROES THEN
        GO TO SEND-DONE.

*-----
* SETUP REPLY
*-----
    MOVE TDSCHAR          TO DB-HOST-TYPE.

```

```
MOVE TDSCHAR          TO DB-CLIENT-TYPE .
MOVE LENGTH OF WROW   TO WRKLEN1 .
MOVE LENGTH OF BANANA TO WRKLEN2 .
```

```
CALL 'TDESCRIB' USING  GWL-PROC, GWL-RC,
                       CTR-COLUMN,
                       DB-HOST-TYPE,
                       WRKLEN1,
                       WROW,
                       DB-NULL-INDICATOR,
                       TDS-FALSE,
                       DB-CLIENT-TYPE,
                       WRKLEN1,
                       BANANA,
                       WRKLEN2 .
```

```
IF GWL-RC NOT EQUAL TO ZEROES THEN
    MOVE 'TDESCRIB' TO CALL-ERROR
    PERFORM DISPLAY-CALL-ERROR
END-IF .
```

```
*-----
*   SEND ROWS TO CLIENT
*-----
MOVE ZEROES TO CTR-ROWS .
```

```
IF PARM-NR-ROWS = ZEROES THEN
    MOVE 'Y' TO ALL-DONE-SW
ELSE
PERFORM SEND-ROWS
UNTIL ALL-DONE OR CTR-ROWS >= PARM-NR-ROWS .
```

```
IF SEND-DONE-OK
    MOVE TDS-DONE-COUNT TO WRK-DONE-STATUS
ELSE
    MOVE TDS-DONE-ERROR TO WRK-DONE-STATUS
    MOVE ZERO           TO CTR-ROWS
END-IF .
```

```
SEND-DONE .
```

```
IF PARM-NR-ROWS = ZEROES THEN
    MOVE TDS-ENDRPC TO GWL-SEND-DONE
ELSE
    MOVE TDS-ENDREPLY TO GWL-SEND-DONE .
```

```

*-----
*          ISSUE SEND DONE TO CLIENT
*-----
CALL 'TDSNDDON' USING  GWL-PROC, GWL-RC,
                        WRK-DONE-STATUS,
                        CTR-ROWS,
                        TDS-ZERO,
                        GWL-SEND-DONE.

IF GWL-RC NOT EQUAL TO ZEROES THEN
    MOVE 'TDSNDDON' TO CALL-ERROR
    PERFORM DISPLAY-CALL-ERROR
END-IF.

IF PARM-NR-ROWS = ZEROES THEN
    PERFORM FREE-ALL-STORAGE
    GOBACK.

*-----
*  GET NEXT CLIENT REQUEST
*-----
    MOVE TDS-TRUE TO GWL-WAIT-OPTION.
    MOVE ZEROES TO GWL-REQ-TYPE.
    MOVE SPACES TO GWL-TRAN-NAME.
CALL 'TDGETREQ' USING  GWL-PROC, GWL-RC, GWL-WAIT-OPTION,
                        GWL-REQ-TYPE, GWL-TRAN-NAME.

EVALUATE GWL-RC

    WHEN ZEROES
        GO TO READ-IN-USER-PARMS

    WHEN TDS-RESULTS-COMPLETE
        PERFORM FREE-ALL-STORAGE

    WHEN TDS-CONNECTION-TERMINATED
        PERFORM FREE-ALL-STORAGE

    WHEN TDS-CONNECTION-FAILED
        PERFORM FREE-ALL-STORAGE

    WHEN OTHER
        MOVE 'TDGETREQ' TO CALL-ERROR
        PERFORM DISPLAY-CALL-ERROR

```

```
        END-EVALUATE.

        GOBACK.

READ-IN-EXIT.
    EXIT.

SET-UP-ROW-PATTERN.

    MOVE PARM-PATTERN TO WROW-PATTERN (I).

SET-UP-ROW-PATTERN-EXIT.
    EXIT.

*-----
SEND-ROWS.
*-----
CALL 'TDSNDROW' USING      GWL-PROC, GWL-RC

    EVALUATE GWL-RC

    WHEN ZEROES
        ADD 1 TO CTR-ROWS

    WHEN TDS-CANCEL-RECEIVED
        MOVE 'Y' TO ALL-DONE-SW
        MOVE CANCEL-RCV-MSG TO MSG-TEXT
        MOVE LENGTH OF CANCEL-RCV-MSG TO MSG-TEXT-L
        PERFORM SEND-MESSAGE

    WHEN OTHER
        PERFORM DISPLAY-CALL-ERROR
        MOVE 'Y' TO SEND-DONE-SW
        MOVE 'Y' TO ALL-DONE-SW

    END-EVALUATE.

SEND-ROWS-EXIT.
    EXIT.

*-----
DISPLAY-CALL-ERROR.
*-----

    MOVE GWL-RC TO CALL-ERROR-RC.
```

```

        MOVE CALL-ERROR-MESSAGE TO MSG-TEXT.
        MOVE LENGTH OF CALL-ERROR-MESSAGE TO MSG-TEXT-L.
        PERFORM SEND-MESSAGE.
        DISPLAY CALL-ERROR-MESSAGE.
        PERFORM FREE-ALL-STORAGE.
        GOBACK.

DISPLAY-CALL-ERROR-EXIT.
        EXIT.

*-----
        FREE-ALL-STORAGE.
*-----

CALL 'TDFREE' USING GWL-PROC, GWL-RC

IF GWL-RC NOT EQUAL TO ZEROES THEN
        MOVE GWL-RC TO CALL-ERROR-RC
        MOVE 'TDFREE' TO CALL-ERROR
        DISPLAY CALL-ERROR-MESSAGE
END-IF.

CALL 'TDTERM' USING GWL-INIT-HANDLE, GWL-RC.

IF GWL-RC NOT EQUAL TO ZEROES THEN
        MOVE GWL-RC TO CALL-ERROR-RC
        MOVE 'TDTERM' TO CALL-ERROR
        DISPLAY CALL-ERROR-MESSAGE
END-IF.

        PERFORM DEALLOC-AIB.

FREE-ALL-STORAGE-EXIT.
        EXIT.

*-----

SEND-ERROR-MESSAGE.

*-----

        MOVE 'N'                TO SEND-DONE-SW.

```

```
MOVE TDS-ERROR-MSG      TO MSG-TYPE.

MOVE LENGTH OF MSG-RPC TO MSG-RPC-L.

*   Ensure we're in right state to send a message

CALL 'TDSTATUS' USING 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 AND

    GWL-STATUS-COMM = TDS-RECEIVE) THEN

CALL 'TDSNDMSG' USING 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-IF.

SEND-MESSAGE-EXIT.
EXIT.

ALLOC-AIB.
* -----
* Allocate AIB
* -----

MOVE `DFSAIB ` TO AIBID.
MOVE `SYIXSAM2' TO AIBRSNM1.
MOVE 128 TO AIBLEN.

CALL `AIBTDLI' USING APSB AIB.

IF AIBRETRN IS EQUAL TO ZEROES THEN
    SET ADDRESS OF PCB-ADDRESSES TO AIBPTR
    SET ADDRESS OF IO-PCB TO PCB-ADDRESS-LIST (1)
ELSE
    DISPLAY `SYIXSAM2 - APSB CALL FAILED RC= ` AIBRETRN
    DISPLAY `SYIXSAM2 - APSB CALL FAILED REASON= ` AIBREASN
    GOBACK.

ALLOC-AIB-EXIT.
EXIT.

DEALLOC-AIB.

```

```
* -----
*   ISSUE SRRCMIT CALL
* -----
CALL 'SRRCMIT' USING CPIC-RC.

IF CPIC-RC IS NOT EQUAL TO ZEROES THEN
    DISPLAY 'SYIXSAM2 SRRCMIT CALL FAILED CPIC-RC=' CPIC-RC.

* -----
*   Deallocate AIB
* -----
CALL 'AIBTDLI' USING DPSB AIB.

IF AIBRETRN IS NOT EQUAL TO ZEROES THEN
    DISPLAY 'SYIXSAM2 - DPSB CALL FAILED RC= ' AIBRETRN
    DISPLAY 'SYIXSAM2 - DPSB CALL FAILED REASON= ' AIBREASN.

DEALLOC-AIB-EXIT.
EXIT.
```


Sample Mixed-Mode Application

This appendix contains a sample COBOL application that uses both Client-Library and Gateway-Library functions. In other words, this program acts as both client and server.

The purpose of this sample program is to demonstrate the use of Gateway-Library functions in a conversational IMS TM program that handles 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. Running it requires Open ServerConnect. SYCTSAX5 uses VS COBOL II and Gateway-Library.

Sample program SYCTSAX5

```

*****
*
* Confidential property of Sybase, Inc.
* (c) Copyright Sybase, Inc. 1985 TO ???
* All rights reserved.
*****

***** SYCTSAX5 - OpenServerOpenClient - COBOL - CICS *****
*
*
* TRANID:          SYX5
*
* PROGRAM:        SYCTSAX5
*
*
* TABLE:         SYBASE.SAMPLETB
*

```

```
*
* PURPOSE: Demonstrates Open Server/Open Client CALLs.
*
* FUNCTION: Illustrates 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"
*
* PREREQS: Before running SYCTSAX5, make sure that the server
*           you wish to access has an entry in the Connection
*           Router Table for that Server and the MCC(s) that
*           you wish to use.
*
* INPUT:   On the input, make sure to enter the Server name,
*           user id, and password for the target server that
*           executes the RPC - SYX5.
*
* Open 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:
*
* 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#      Description *
*
* =====
* Feb1795      Create *
* Nov1595 99999 Rewrite and add front end to the program *

```

IDENTIFICATION DIVISION.
PROGRAM-ID. SYCTSAX5.

ENVIRONMENT DIVISION.

DATA DIVISION.
WORKING-STORAGE SECTION.

```

*-----
* Client Library Cobol Copy Book
*-----

```

COPY CTPUBLIC.

```

*-----
* Server Library Cobol Copy Book
*-----

```

COPY SYGWCOB.

```

*-----
* Standard CICS Attribute and Print Control Chararcter List
*-----

```

COPY DFHBMSCA.

```

*-----

```

* CICS Standard Attention Identifiers Cobol Copy Book

*-----

COPY DFHAID.

*-----

* Work Area

*-----

```
01  GW-LIB-MISC-FIELDS.
      05  GWL-TDPROC          POINTER.
      05  GWL-RC              PIC S9(9) COMP SYNC VALUE 0.

01  INTERNAL-FIELDS.
      05  CF-FOUR              PIC S9(9) COMP VALUE 4.

01  SWITCHES.
      05  SW-RESULTS          PIC X(01) value 'Y'.
          88  NO-MORE-RESULTS VALUE 'N'.
      05  SW-FETCH            PIC X(01) value 'Y'.
          88  NO-MORE-ROWS VALUE 'N'.
      05  SW-DIAG              PIC X(01) VALUE 'N'.
          88  DIAG-MSGS-INITIALIZED VALUE 'Y'.

01  PARM-FIELDS.
      05  PF-PARM-ID          PIC S9(9) COMP SYNC.
      05  PF-DATATYPE         PIC S9(9) COMP SYNC.
      05  PF-ACTUAL-DATA-LENGTH PIC S9(9) COMP SYNC.
      05  PF-MAX-DATA-LENGTH  PIC S9(9) COMP SYNC.
      05  PF-PARM-STATUS      PIC S9(9) COMP SYNC.
      05  PF-PARM-NAME        PIC X(30) .
      05  PF-PARM-NAME-LENGTH PIC S9(9) COMP SYNC.
      05  PF-USER-DATATYPE    PIC S9(9) COMP SYNC.
      05  PF-NUM-OF-PARMS     PIC S9(9) COMP SYNC.
      05  PF-MSGLIMIT         PIC S9(9) COMP.

01  SNA-FIELDS.
      05  SNA-SUBC            PIC S9(9) COMP SYNC.
      05  SNA-CONNECTION-NAME PIC X(8) VALUE IS SPACES.

01  WORK-FIELDS.
      05  WRK-DONE-STATUS     PIC S9(9) COMP SYNC.

01  DESCRIBE-FIELDS.
      05  DF-COLUMN-NUMBER    PIC S9(9) COMP SYNC VALUE 0.
      05  DF-HOST-VARIABLE-TYPE PIC S9(9) COMP SYNC VALUE 0.
```

```

05 DF-HOST-VARIABLE-MAXLEN PIC S9(9) COMP SYNC VALUE 0.
05 DF-HOST-VARIABLE-NAME   POINTER.
05 DF-NULL-INDICATOR-VAR  PIC S9(9) COMP SYNC VALUE 0.
05 DF-NULLS-ALLOWED      PIC S9(9) COMP SYNC VALUE 0.
05 DF-COLUMN-TYPE         PIC S9(9) COMP SYNC VALUE 0.
05 DF-COLUMN-MAXLEN       PIC S9(9) COMP SYNC VALUE 0.
05 DF-COLUMN-NAME         PIC X(30) .
05 DF-COLUMN-NAME-LEN     PIC S9(9) COMP SYNC VALUE 0.

01 SNDMSG-FIELDS.
    05 SF-MESSAGE-TYPE     PIC S9(9) COMP SYNC.
    05 SF-MESSAGE-NUMBER   PIC S9(9) COMP SYNC.
    05 SF-SEVERITY         PIC S9(9) COMP SYNC.
    05 SF-ERROR-STATE      PIC S9(9) COMP SYNC.
    05 SF-LINE-ID          PIC S9(9) COMP SYNC.
    05 SF-TRANSACTION-ID   PIC X(4) VALUE 'SYX5' .
    05 SF-TRANSACTION-ID-LEN PIC S9(9) COMP SYNC.
    05 SF-MESSAGE-TEXT     PIC X(80) .
    05 SF-MESSAGE-LENGTH   PIC S9(9) COMP SYNC.

01 CTX                      PIC S9(9) COMP SYNC.

01 ROW-DATA                 PIC X(80) VALUE IS SPACES.

*-----
* Work Areas Open Client
*-----

01 CS-LIB-MISC-FIELDS.
    05 CSL-CMD-HANDLE      PIC S9(9) COMP SYNC VALUE 0.
    05 CSL-CON-HANDLE      PIC S9(9) COMP SYNC VALUE 0.
    05 CSL-CTX-HANDLE      PIC S9(9) COMP SYNC VALUE 0.
    05 CSL-RC              PIC S9(9) COMP SYNC.

01 PROPS-FIELDS.
    05 PF-SERVER           PIC X(30) .
    05 PF-SERVER-SIZE     PIC S9(9) COMP.
    05 PF-USER             PIC X(30) .
    05 PF-USER-SIZE       PIC S9(9) COMP.
    05 PF-PWD             PIC X(30) .
    05 PF-PWD-SIZE        PIC S9(9) COMP.
    05 PF-OUTLEN          PIC S9(9) COMP SYNC.
    05 PF-STRLEN          PIC S9(9) COMP SYNC.

01 QUERY-FIELDS.
    05 QF-LEN             PIC S9(9) VALUE 1.

```

```

05 QF-MAXLEN          PIC S9(9) VALUE 1.
05 QF-ANSWER          PIC X(01) VALUE ' '.

01  FETCH-FIELDS.
    05  FF-ROWS-READ    PIC S9(9) COMP SYNC VALUE 0.
    05  FF-ROW-NUM     PIC S9(9) COMP SYNC VALUE 0.

01  COLUMN-FIELDS.
    05  CF-COL-FIRSTNME PIC X(12) VALUE SPACES.
    05  CF-COL-NUMBER   PIC S9(9) COMP SYNC VALUE 0.
    05  CF-COL-INDICATOR PIC S9(9) COMP SYNC VALUE 0.
    05  CF-COL-OUTLEN   PIC S9(9) COMP SYNC VALUE 0.

01  LANG-FIELDS.
    05  LF-LANG          PIC X(36)
          VALUE 'SELECT FIRSTNME FROM SYBASE.SAMPLET'B'.

01  ERROR-MSG.
    05  ERROR-TEXT      PIC X(50) VALUE ' '.
    05  ERROR-LITERAL   PIC X(06) VALUE ' RC = '.
    05  ERROR-RC        PIC -ZZZ9.

01  ERROR-MSG-STR REDEFINES ERROR-MSG PIC X(61).

01  INFO-MSG-STR          PIC X(80) VALUE ' '.

01  RESULTS-FIELDS.
    05  RF-TYPE          PIC S9(9) COMP SYNC VALUE 0.

01  DATAFMT.
    05  DF-NAME          PIC X(132).
    05  DF-NAMELEN      PIC S9(9) COMP SYNC.
    05  DF-DATATYPE     PIC S9(9) COMP SYNC.
    05  DF-FORMAT       PIC S9(9) COMP SYNC.
    05  DF-MAXLENGTH    PIC S9(9) COMP SYNC.
    05  DF-SCALE        PIC S9(9) COMP SYNC.
    05  DF-PRECISION    PIC S9(9) COMP SYNC.
    05  DF-STATUS       PIC S9(9) COMP SYNC.
    05  DF-COUNT        PIC S9(9) COMP SYNC.
    05  DF-USERTYPE     PIC S9(9) COMP SYNC.
    05  DF-LOCALE       PIC X(68).

```

```

*-----
* Common Work Areas
*-----

```

```

01  MSG-FIELDS.

```

```

05 MSG-END-MSG          PIC X(25)
    VALUE 'All done processing rows.'.
05 MSG-NOT-RPC          PIC X(35)
    VALUE 'SYX5 must be begun via rpc request.'.
05 MSG-WRONG-NR-PARMS   PIC X(40)
    VALUE 'Number of parameters must be 2 or 3.'.
05 MSG-NOT-INT4-PARM    PIC X(33)
    VALUE 'Parameter must be a INTEGER type.'.
05 MSG-CANCELED         PIC X(17)
    VALUE 'Cancel requested.'.
05 MSG-TDRCVPRM-FAIL    PIC X(16)
    VALUE 'TDRCVPRM failed.'.

01 CICS-FIELDS.
    05 CICS-RESPONSE     PIC S9(9) COMP SYNC.

01 MISC-FIELDS.
    05 I                 PIC S9(9) COMP.
    05 LCV               PIC S9(9) COMP SYNC.
    05 TMP-DATE          PIC X(08).
    05 TMP-TIME          PIC X(08).
    05 UTIME            PIC S9(15) COMP-3.

01 X5-HEADER.
    05 X5-DATE-HDR       PIC X(06) VALUE ' DATE '.
    05 X5-DATE-DATA     PIC X(08).
    05 X5-HDR           PIC X(56).
01 X5-HEADER-STR REDEFINES X5-HEADER PIC X(70).

01 X5-HEADER2.
    05 X5-TIME-HDR       PIC X(06) VALUE ' TIME '.
    05 X5-TIME-DATA     PIC X(08).
01 X5-HEADER2-STR REDEFINES X5-HEADER2 PIC X(14).

01 DISP-MSG.
    05 TEST-CASE        PIC X(08) VALUE IS 'SYCTSAA5'.
    05 FILLER           PIC X(01) VALUE IS SPACES.
    05 MSG.
        10 SAMP-LIT     PIC X(05) VALUE IS 'rc = '.
        10 SAMP-RC     PIC -Z9.
        10 FILLER      PIC X(02) VALUE IS ', '.
        10 REST-LIT    PIC X(12) VALUE IS
            'Result Type:'.
        10 REST-TYPE   PIC Z(3)9.
        10 FILLER     PIC X(03) VALUE IS SPACES.
        10 MSGSTR     PIC X(40) VALUE IS SPACES.

```

```

01  DIAG-FIELDS.
    05  DG-MSGNO          PIC S9(9) COMP VALUE +1.
    05  DG-NUM-OF-MSGS  PIC S9(9) COMP VALUE +0.

01  DISP-SERVER.
    05  SERVER-HDR       PIC X(09) VALUE IS
                          ' SERVER: '.
    05  SERVER-DATA     PIC X(20).
    05  USER-HDR        PIC X(10) VALUE IS
                          ' USER-ID: '.
    05  USER-DATA       PIC X(30).

```

```

*-----
* Client Message Structure
*-----

```

```

01  CLIENT-MSG.
    05  CM-SEVERITY      PIC S9(9) COMP SYNC.
    05  CM-MSGNO        PIC S9(9) COMP SYNC.
    05  CM-TEXT         PIC X(256).
    05  CM-TEXT-LEN     PIC S9(9) COMP SYNC.
    05  CM-OS-MSGNO     PIC S9(9) COMP SYNC.
    05  CM-OS-MSGTXT    PIC X(256).
    05  CM-OS-MSGTEXT-LEN PIC S9(9) COMP SYNC.
    05  CM-STATUS       PIC S9(9) COMP.

01  DISP-CLIENT-MSG-HDR.
    05  CLIENT-MSG-HDR  PIC X(15) VALUE IS
                          'Client Message:'.

01  DISP-CLIENT-MSG-1.
    05  FILLER          PIC X(02) VALUE IS SPACES.
    05  CM-SEVERITY-HDR PIC X(09) VALUE IS 'Severity:'.
    05  FILLER          PIC X(02) VALUE IS SPACES.
    05  CM-SEVERITY-DATA PIC Z(8)9.
    05  CM-STATUS-HDR  PIC X(12) VALUE IS
                          ', Status: '.
    05  FILLER          PIC X(02) VALUE IS SPACES.
    05  CM-STATUS-DATA PIC Z(8)9.

01  DISP-CLIENT-MSG-2.
    05  FILLER          PIC X(02) VALUE IS SPACES.
    05  CM-OC-MSGNO-HDR PIC X(09) VALUE IS 'OC MsgNo:'.

```



```

05 FILLER PIC X(02) VALUE IS SPACES.
05 CM-OC-MSGNO-DATA PIC Z(8)9.

01 DISP-CLIENT-MSG-3.
05 FILLER PIC X(02) VALUE IS SPACES.
05 CM-OC-MSG-HDR PIC X(09) VALUE IS 'OC MsgTx: '.
05 FILLER PIC X(02) VALUE IS SPACES.
05 CM-OC-MSG-DATA PIC X(66) .

01 DISP-CLIENT-MSG-3A.
05 CM-OC-MSG-DATA-1 PIC X(66) .
05 CM-OC-MSG-DATA-2 PIC X(66) .
05 CM-OC-MSG-DATA-3 PIC X(66) .
05 CM-OC-MSG-DATA-4 PIC X(58) .

01 DISP-CLIENT-MSG-3B.
05 FILLER PIC X(13) VALUE IS SPACES.
05 CM-OC-MSG-DATA-X PIC X(66) .

01 DISP-EMPTY-CLIENT-MSG-3.
05 FILLER PIC X(02) VALUE IS SPACES.
05 CM-OC-MSG-HDR PIC X(09) VALUE IS 'OC MsgTx: '.
05 FILLER PIC X(02) VALUE IS SPACES.
05 NO-DATA PIC X(11) VALUE IS 'No Message!'.

01 DISP-CLIENT-MSG-4.
05 FILLER PIC X(02) VALUE IS SPACES.
05 CM-OS-MSG-HDR PIC X(09) VALUE IS 'OS MsgNo: '.
05 FILLER PIC X(02) VALUE IS SPACES.
05 CM-OS-MSGNO-DATA PIC Z(8)9.

01 DISP-CLIENT-MSG-5.
05 FILLER PIC X(02) VALUE IS SPACES.
05 CM-OS-MSG-HDR PIC X(09) VALUE IS 'OS MsgTx: '.
05 FILLER PIC X(02) VALUE IS SPACES.
05 CM-OS-MSG-DATA PIC X(66) .

01 DISP-CLIENT-MSG-5A.
05 CM-OS-MSG-DATA-1 PIC X(66) .
05 CM-OS-MSG-DATA-2 PIC X(66) .
05 CM-OS-MSG-DATA-3 PIC X(66) .
05 CM-OS-MSG-DATA-4 PIC X(58) .

01 DISP-EMPTY-CLIENT-MSG-5.
05 FILLER PIC X(02) VALUE IS SPACES.
05 CM-OS-MSG-HDR PIC X(09) VALUE IS 'OS MsgTx: '.

```

```

05 FILLER PIC X(02) VALUE IS SPACES.
05 NO-DATA PIC X(11) VALUE IS 'No Message!'.

*-----
* Server Message Structure
*-----

01 SERVER-MSG.
05 SM-MSGNO PIC S9(9) COMP.
05 SM-STATE PIC S9(9) COMP.
05 SM-SEV PIC S9(9) COMP.
05 SM-TEXT PIC X(256).
05 SM-TEXT-LEN PIC S9(9) COMP.
05 SM-SVRNAME PIC X(256).
05 SM-SVRNAME-LEN PIC S9(9) COMP.
05 SM-PROC PIC X(256).
05 SM-PROC-LEN PIC S9(9) COMP.
05 SM-LINE PIC S9(9) COMP.
05 SM-STATUS PIC S9(9) COMP.

01 DISP-SERVER-MSG-HDR.
05 SERVER-MSG-HDR PIC X(15) VALUE IS
    'Server Message:'.

01 DISP-SERVER-MSG-1.
05 FILLER PIC X(02) VALUE IS SPACES.
05 SM-MSG-NO-HDR PIC X(09) VALUE IS
    'Message#:'.
05 FILLER PIC X(02) VALUE IS SPACES.
05 SM-MSG-NO-DATA PIC Z(8)9.
05 SM-SEVERITY-HDR PIC X(12) VALUE IS
    ', Severity:'.
05 FILLER PIC X(02) VALUE IS SPACES.
05 SM-SEVERITY-DATA PIC Z(8)9.
05 SM-STATE-HDR PIC X(12) VALUE IS
    ', State No:'.
05 FILLER PIC X(02) VALUE IS SPACES.
05 SM-STATE-DATA PIC Z(8)9.

01 DISP-SERVER-MSG-2.
05 FILLER PIC X(02) VALUE IS SPACES.
05 SM-LINE-NO-HDR PIC X(09) VALUE IS
    'Line No:'.
05 FILLER PIC X(02) VALUE IS SPACES.
05 SM-LINE-NO-DATA PIC Z(8)9.
05 SM-STATUS-HDR PIC X(12) VALUE IS

```

```

                                ', Status :'.
05  FILLER                       PIC X(02) VALUE IS SPACES.
05  SM-STATUS-DATA               PIC Z(8)9.

01  DISP-SERVER-MSG-3.
05  FILLER                       PIC X(02) VALUE IS SPACES.
05  SM-SVRNAME-HDR              PIC X(09) VALUE IS 'Serv Nam:'.
05  FILLER                       PIC X(02) VALUE IS SPACES.
05  SM-SVRNAME-DATA            PIC X(66).
05  FILLER                       PIC X(03) VALUE IS '...'.

01  DISP-SERVER-MSG-4.
05  FILLER                       PIC X(02) VALUE IS SPACES.
05  SM-PROC-ID-HDR             PIC X(09) VALUE IS 'Proc ID:'.
05  FILLER                       PIC X(02) VALUE IS SPACES.
05  SM-PROC-ID-DATA           PIC X(66).

01  DISP-SERVER-MSG-5.
05  FILLER                       PIC X(02) VALUE IS SPACES.
05  SM-MSG-HDR                 PIC X(09) VALUE IS 'Message :'.
05  FILLER                       PIC X(02) VALUE IS SPACES.
05  SM-MSG-DATA                PIC X(66).

01  DISP-SERVER-MSG-5A.
05  SM-MSG-DATA-1              PIC X(66).
05  SM-MSG-DATA-2              PIC X(66).
05  SM-MSG-DATA-3              PIC X(66).
05  SM-MSG-DATA-4              PIC X(58).

01  DISP-SERVER-MSG-5X.
05  FILLER                       PIC X(13) VALUE IS SPACES.
05  SM-MSG-DATA-X              PIC X(66).

```

PROCEDURE DIVISION.

```

*-----
* Begin program here
*-----

```

```

MOVE LOW-VALUES TO PARM-FIELDS DATAFMT.
MOVE 'Y'          TO SW-DIAG.

EXEC CICS ASKTIME
        ABSTIME(UTIME)
END-EXEC.

```

```
EXEC CICS FORMATTIME
      ABSTIME(UTIME)
      DATESEP('/')
      MMDDYY(TMP-DATE)
      TIME(TMP-TIME)
      TIMESEP
END-EXEC.

MOVE
  ' SYBASE COBOL SAMPLE PROGRAM SYCTSAX5 SQL RESULT OUTPUT '
      TO X5-HDR.
MOVE TMP-DATE TO X5-DATE-DATA.
MOVE TMP-TIME TO X5-TIME-DATA.
```

```
*-----
* initialize the TDS environment for a client
*-----
```

```
CALL 'TDINIT' USING DFHEIBLK,
      GWL-RC,
      CSL-CTX-HANDLE.
```

```
IF GWL-RC NOT = TDS-OK
  THEN
    MOVE 'TDINIT failed' TO ERROR-TEXT
    MOVE GWL-RC          TO ERROR-RC
    PERFORM SEND-ERROR-MESSAGE
    PERFORM ALL-DONE
END-IF.
```

```
*-----
* accept request from a remote client
*-----
```

```
CALL 'TDACCEPT' USING GWL-TDPROC,
      GWL-RC,
      CSL-CTX-HANDLE,
      SNA-CONNECTION-NAME,
      SNA-SUBC.
```

```
IF GWL-RC NOT = TDS-OK
  THEN
    MOVE 'TDACCEPT failed' TO ERROR-TEXT
    MOVE GWL-RC          TO ERROR-RC
    PERFORM SEND-ERROR-MESSAGE
    PERFORM ALL-DONE
END-IF.
```

```
*-----  
* display date and time  
*-----  
  
    MOVE SPACES TO INFO-MSG-STR.  
    PERFORM SEND-INFO-MESSAGE.  
  
    MOVE X5-HEADER-STR TO INFO-MSG-STR.  
    PERFORM SEND-INFO-MESSAGE.  
  
    MOVE X5-HEADER2-STR TO INFO-MSG-STR.  
    PERFORM SEND-INFO-MESSAGE.  
  
    MOVE SPACES TO INFO-MSG-STR.  
    PERFORM SEND-INFO-MESSAGE.  
  
*-----  
* determine how many parameters were sent with the current RPC  
*   by the remote client or server  
*-----  
  
    CALL 'TDNUMPRM' USING GWL-TDPROC,  
                        PF-NUM-OF-PARMS.  
  
    IF PF-NUM-OF-PARMS = 2 OR PF-NUM-OF-PARMS = 3  
    THEN  
        MOVE SPACES TO INFO-MSG-STR  
    ELSE  
        MOVE MSG-WRONG-NR-PARMS TO INFO-MSG-STR  
        PERFORM SEND-INFO-MESSAGE  
  
        MOVE SPACES TO INFO-MSG-STR  
        PERFORM SEND-INFO-MESSAGE  
  
        MOVE  
        'syntax is: SYX5 server-nm, user-id OR'  
        TO INFO-MSG-STR  
        PERFORM SEND-INFO-MESSAGE  
  
        MOVE  
        '          SYX5 server-nm, user-id, password'  
        TO INFO-MSG-STR  
        PERFORM SEND-INFO-MESSAGE  
  
    PERFORM ALL-DONE
```

END-IF.

```
*-----
* retrieves parameter type, datatype, and length information
* about the 1st RPC parameter( server-name parameter )
*-----
```

MOVE 1 TO PF-PARM-ID.

```
CALL 'TDINFPRM' USING 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 NOT = TDS-OK
  THEN
    MOVE 'TDINFPRM for server-name parameter failed'
        TO ERROR-TEXT
    MOVE GWL-RC TO ERROR-RC
    PERFORM SEND-ERROR-MESSAGE
    PERFORM ALL-DONE
  END-IF.
```

```
IF PF-DATATYPE NOT = TDSCHAR AND
   PF-DATATYPE NOT = TDSVARYCHAR
  THEN
    MOVE 'server-name datatype must be TDSCHAR'
        TO INFO-MSG-STR
    PERFORM SEND-INFO-MESSAGE
    PERFORM ALL-DONE
  END-IF.
```

```
*-----
* retrieves the data from an RPC parameter sent by a remote
* client
*-----
```

MOVE LENGTH OF PF-SERVER TO PF-STRLEN.

```
CALL 'TDRCVPRM' USING GWL-TDPROC,
```

```

                                GWL-RC,
                                PF-PARM-ID,
                                PF-SERVER,
                                TDSCHAR,
                                PF-STRLEN,
                                PF-ACTUAL-DATA-LENGTH.

IF GWL-RC NOT = TDS-OK
  THEN
    MOVE 'TDRCVPRM for server-name parameter failed'
      TO ERROR-TEXT
    MOVE GWL-RC TO ERROR-RC
    PERFORM SEND-ERROR-MESSAGE
    PERFORM ALL-DONE
  END-IF.

```

```

MOVE PF-ACTUAL-DATA-LENGTH TO PF-SERVER-SIZE.

```

```

*-----
* retrieves parameter type, datatype, and length information
* about the 2nd RPC parameter( user-id parameter )
*-----

```

```

MOVE 2 TO PF-PARM-ID.

```

```

CALL 'TDINFPRM' USING 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 NOT = TDS-OK
  THEN
    MOVE 'TDINFPGM for user-id parameter failed'
      TO ERROR-TEXT
    MOVE GWL-RC TO ERROR-RC
    PERFORM SEND-ERROR-MESSAGE
    PERFORM ALL-DONE
  END-IF.

```

```

IF PF-DATATYPE NOT = TDSCHAR AND

```

```
PF-DATATYPE NOT = TDSVARYCHAR
THEN
  MOVE 'user-id datatype must be TDSCHAR'
      TO INFO-MSG-STR
  PERFORM SEND-INFO-MESSAGE
  PERFORM ALL-DONE
END-IF.
```

```
*-----
* retrieves the data from an RPC parameter sent by a remote
* client
*-----
```

```
MOVE LENGTH OF PF-USER TO PF-STRLEN.
```

```
CALL 'TDRCVPRM' USING GWL-TDPROC,
                    GWL-RC,
                    PF-PARM-ID,
                    PF-USER,
                    TDSCHAR,
                    PF-STRLEN,
                    PF-ACTUAL-DATA-LENGTH.
```

```
IF GWL-RC NOT = TDS-OK
THEN
  MOVE 'TDRCVPRM for user-id failed' TO ERROR-TEXT
  MOVE GWL-RC TO ERROR-RC
  PERFORM SEND-ERROR-MESSAGE
  PERFORM ALL-DONE
END-IF.
```

```
MOVE PF-ACTUAL-DATA-LENGTH TO PF-USER-SIZE.
```

```
IF PF-NUM-OF-PARMS = 3
THEN
```

```
*-----
* retrieves parameter type, datatype, and length information
* about the 3rd RPC parameter ( password parameter )
*-----
```

```
MOVE 3 TO PF-PARM-ID
```



```

CALL 'TDINFPRM' USING 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 NOT = TDS-OK
  THEN
    MOVE 'TDINFPRM for server-name parameter failed'
      TO ERROR-TEXT
    MOVE GWL-RC TO ERROR-RC
    PERFORM SEND-ERROR-MESSAGE
    PERFORM ALL-DONE
  END-IF

IF PF-DATATYPE NOT = TDSCHAR AND
  PF-DATATYPE NOT = TDSVARYCHAR
  THEN
    MOVE 'server-name datatype must be TDSCHAR'
      TO INFO-MSG-STR
    PERFORM SEND-INFO-MESSAGE
    PERFORM ALL-DONE
  END-IF

```

```

*-----
* retrieves the data from an RPC parameter sent by a remote
*   client
*-----

```

```

MOVE LENGTH OF PF-PWD TO PF-STRLEN

CALL 'TDRCVPRM' USING GWL-TDPROC,
                    GWL-RC,
                    PF-PARM-ID,
                    PF-PWD,
                    TDSCHAR,
                    PF-STRLEN,
                    PF-ACTUAL-DATA-LENGTH

IF GWL-RC NOT = TDS-OK
  THEN

```

```

        MOVE 'TDRCVPRM for password parameter failed'
              TO ERROR-TEXT
        MOVE GWL-RC TO ERROR-RC
        PERFORM SEND-ERROR-MESSAGE
        PERFORM ALL-DONE
    END-IF

        MOVE PF-ACTUAL-DATA-LENGTH TO PF-PWD-SIZE
    ELSE
        MOVE SPACES TO PF-PWD
        MOVE 0      TO PF-PWD-SIZE
    END-IF.

*-----
* display server and user-id heading
*-----

        MOVE PF-SERVER  TO SERVER-DATA.
        MOVE PF-USER    TO USER-DATA.
        MOVE DISP-SERVER TO INFO-MSG-STR.
        PERFORM SEND-INFO-MESSAGE.

        MOVE SPACES TO INFO-MSG-STR.
        PERFORM SEND-INFO-MESSAGE.

*-----
* describe the 1st column in a result row and the mainframe
* server program variable where it is stored
*-----

        MOVE 1                TO DF-COLUMN-NUMBER.
        MOVE TDSVARYCHAR      TO DF-HOST-VARIABLE-TYPE.
        MOVE LENGTH OF CF-COL-FIRSTNME TO DF-HOST-VARIABLE-MAXLEN.
        MOVE TDS-ZERO         TO DF-NULL-INDICATOR-VAR.
        MOVE TDS-FALSE        TO DF-NULLS-ALLOWED.
        MOVE TDSVARYCHAR      TO DF-COLUMN-TYPE.
        MOVE LENGTH OF CF-COL-FIRSTNME TO DF-COLUMN-MAXLEN.
        MOVE 'FIRST NAME'     TO DF-COLUMN-NAME.
        MOVE 10               TO DF-COLUMN-NAME-LEN.

        CALL 'TDESCRIB' USING 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 NOT = TDS-OK
  THEN
    MOVE 'TDESCRIB failed' TO ERROR-TEXT
    MOVE GWL-RC TO ERROR-RC
    PERFORM SEND-ERROR-MESSAGE
    PERFORM ALL-DONE
  END-IF.

PERFORM OC-INIT.

PERFORM OC-CONNECT.

PERFORM OC-SEND-LANG.

PERFORM OC-PROCESS-RESULTS.

PERFORM OC-ALL-DONE.

*=====
*==
*== Subroutine to send a results completion indication ==
*== to the client, free up a previously allocated ==
*== GWL_TDPROC structure, and return back to CICS ==
*==
*=====
  ALL-DONE.

*-----
* send a results completion indication to the client
*-----

CALL 'TDSNDDON' USING GWL-TDPROC,
                      GWL-RC,
                      TDS-DONE-FINAL,
                      TDS-NULL,
                      TDS-ZERO,
                      TDS-ENDRPC.

```

```

IF GWL-RC NOT = TDS-OK
  THEN
    MOVE 'TDSNDDON failed' TO ERROR-TEXT
    MOVE GWL-RC           TO ERROR-RC
    PERFORM SEND-ERROR-MESSAGE
  END-IF.
*-----
* free up a previously allocated GWL_TDPROC structure after
*   returning results to a client
*-----

CALL 'TDFREE' USING GWL-TDPROC,
                    GWL-RC.

IF GWL-RC NOT = TDS-OK
  THEN
    MOVE 'TDFREE failed' TO ERROR-TEXT
    MOVE GWL-RC           TO ERROR-RC
    PERFORM SEND-ERROR-MESSAGE
  END-IF.

*-----
* return back to CICS
*-----

EXEC CICS RETURN END-EXEC.

*=====
*==
*== Subroutine to initialize the Client-Library ==
*==
*=====
OC-INIT.

*-----
* initialize the Client-Library
*-----

CALL 'CTBINIT' USING CSL-CTX-HANDLE,
                    CSL-RC,
                    CS-VERSION-46.

IF CSL-RC NOT = CS-SUCCEED
  THEN
    MOVE 'CTBINIT failed' TO MSGSTR
    PERFORM ERROR-OUT

```

```

        PERFORM ALL-DONE
    END-IF.

*=====
*==
*== 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.

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

        CALL 'CTBCONAL' USING CSL-CTX-HANDLE,
                                CSL-RC,
                                CSL-CON-HANDLE.

        IF CSL-RC NOT = CS-SUCCEED
            THEN
                MOVE 'CTBCONAL failed' TO MSGSTR
                PERFORM ERROR-OUT
                PERFORM ALL-DONE
        END-IF.

*-----
*alter properties of the connection
*-----

        CALL 'CTBCONPR' USING CSL-CON-HANDLE,
                                CSL-RC,
                                CS-SET,
                                CS-USERNAME,
                                PF-USER,
                                PF-USER-SIZE,
                                CS-FALSE,
                                CS-UNUSED.

        IF CSL-RC NOT = CS-SUCCEED
            THEN
                MOVE 'CTBCONPR for user-id failed' TO MSGSTR
                PERFORM ERROR-OUT
                PERFORM ALL-DONE

```

END-IF.

```
CALL 'CTBCONPR' USING CSL-CON-HANDLE,
                        CSL-RC,
                        CS-SET,
                        CS-PASSWORD,
                        PF-PWD,
                        PF-PWD-SIZE,
                        CS-FALSE,
                        CS-UNUSED.
```

```
IF CSL-RC NOT = CS-SUCCEED
  THEN
    MOVE 'CTBCONPR for password failed' TO MSGSTR
    PERFORM ERROR-OUT
    PERFORM ALL-DONE
  END-IF.
```

```
*-----*
* setup retrieval of All Messages
*-----*
```

```
CALL 'CTBDIAG' USING CSL-CON-HANDLE,
                        CSL-RC,
                        CS-UNUSED,
                        CS-INIT,
                        CS-ALLMSG-TYPE,
                        CS-UNUSED,
                        CS-UNUSED.
```

```
IF CSL-RC NOT = CS-SUCCEED
  THEN
    MOVE 'CTBDIAG CS-INIT failed' TO MSGSTR
    PERFORM ERROR-OUT
    PERFORM ALL-DONE
  END-IF.
```

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

```
MOVE 5 TO PF-MSGLIMIT.
```

```
CALL 'CTBDIAG' USING CSL-CON-HANDLE,
                        CSL-RC,
                        CS-UNUSED,
```

```

                                CS-MSGLIMIT,
                                CS-ALLMSG-TYPE,
                                CS-UNUSED,
                                PF-MSGLIMIT.

IF CSL-RC NOT = CS-SUCCEED
  THEN
    MOVE 'CTBDIAG CS-MSGLIMIT failed' TO MSGSTR
    PERFORM ERROR-OUT
    PERFORM ALL-DONE
  END-IF.

*-----*
* open connection to the server
*-----*

CALL 'CTBCONNE' USING CSL-CON-HANDLE,
                    CSL-RC,
                    PF-SERVER,
                    PF-SERVER-SIZE,
                    CS-FALSE.

IF CSL-RC NOT = CS-SUCCEED
  THEN
    MOVE 'CTBCONNE failed' TO MSGSTR
    PERFORM ERROR-OUT
    PERFORM ALL-DONE
  END-IF.

*=====  

*==  

*== Subroutine to allocate command handler, prepare ==  

*== and send the language request ==  

*==  

*=====  

OC-SEND-LANG.

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

CALL 'CTBCMDAL' USING CSL-CON-HANDLE,
                    CSL-RC,
                    CSL-CMD-HANDLE.

IF CSL-RC NOT = CS-SUCCEED

```

```

THEN
  MOVE 'CTBCMDAL failed' TO MSGSTR
  PERFORM ERROR-OUT
  PERFORM ALL-DONE
END-IF.

```

```

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

```

```

MOVE LENGTH OF LF-LANG TO PF-STRLEN.

CALL 'CTBCOMMA' USING CSL-CMD-HANDLE,
                      CSL-RC,
                      CS-LANG-CMD,
                      LF-LANG,
                      PF-STRLEN,
                      CS-UNUSED.

```

```

IF CSL-RC NOT = CS-SUCCEED
  THEN
    MOVE 'CTBCOMMA failed' TO MSGSTR
    PERFORM ERROR-OUT
    PERFORM ALL-DONE
  END-IF.

```

```

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

```

```

CALL 'CTBSEND' USING CSL-CMD-HANDLE,
                    CSL-RC.

IF CSL-RC NOT = CS-SUCCEED
  THEN
    MOVE 'CTBSEND failed' TO MSGSTR
    PERFORM ERROR-OUT
    PERFORM ALL-DONE
  END-IF.

```

```

*=====
*==
*== Subroutine to process the result ==
*==
*=====

```


OC-PROCESS-RESULTS.

PERFORM RESULTS-PROCESSING UNTIL NO-MORE-RESULTS.

```
*=====
*==                                                    ==
*== Subroutine to set up the results data              ==
*==                                                    ==
*=====
RESULTS-PROCESSING.
```

```
CALL 'CTBRESUL' USING CSL-CMD-HANDLE
                        CSL-RC
                        RF-TYPE.
```

EVALUATE CSL-RC

WHEN CS-SUCCEED

EVALUATE RF-TYPE

WHEN CS-ROW-RESULT

```
PERFORM ROW-RESULT-PROCESSING
MOVE 'Y' TO SW-FETCH
PERFORM FETCH-PROCESSING UNTIL NO-MORE-ROWS
```

WHEN CS-STATUS-RESULT

```
PERFORM STATUS-PROCESSING
```

WHEN CS-CMD-FAIL

```
MOVE 'RESULTS-PROCESSING CMD-FAIL' TO MSGSTR
PERFORM ERROR-OUT
MOVE 'bad user-id or password'      TO INFO-MSG-STR
PERFORM SEND-INFO-MESSAGE
MOVE SPACES                          TO INFO-MSG-STR
PERFORM SEND-INFO-MESSAGE
```

WHEN CS-CMD-DONE

```
MOVE 'RESULTS-PROCESSING CMD-DONE' TO INFO-MSG-STR
MOVE RF-TYPE TO                      ERROR-RC
```

WHEN OTHER

```
MOVE 'RESULTS-PROCESSING unknown return code'
                        TO MSGSTR
PERFORM ERROR-OUT
```

END-EVALUATE

```

WHEN CS-FAIL
  MOVE 'N'                TO SW-RESULTS
  MOVE 'CTBRESULTS failed' TO MSGSTR
  PERFORM ERROR-OUT

```

```

WHEN CS-END-RESULTS
  MOVE 'N' TO SW-RESULTS

```

```

WHEN OTHER
  MOVE 'N'                TO SW-RESULTS
  MOVE 'CTBRESULTS failed' TO MSGSTR
  PERFORM ERROR-OUT

```

```

END-EVALUATE.

```

```

*=====
*==
*== Subroutine to process row result and bind ==
*==
*=====

```

```

ROW-RESULT-PROCESSING.

```

```

CALL 'CTBRESUL' USING CSL-CMD-HANDLE
                        CSL-RC
                        RF-TYPE.

```

```

MOVE CS-VARCHAR-TYPE   TO DF-DATATYPE.
MOVE CS-FMT-UNUSED     TO DF-FORMAT.
MOVE LENGTH OF CF-COL-FIRSTNME TO DF-MAXLENGTH.
MOVE 1                  TO DF-COUNT.

```

```

*-----
* bind the first column
*-----

```

```

MOVE 1 TO CF-COL-NUMBER.

```

```

CALL 'CTBBIND' USING 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 NOT = CS-SUCCEED
  THEN
    MOVE 'CTBBIND first name failed' TO MSGSTR
    PERFORM ERROR-OUT
    PERFORM ALL-DONE
  END-IF.

*=====
*==                                                    ==
*== Subroutine to fetch the result                      ==
*==                                                    ==
*=====
  FETCH-PROCESSING.

  CALL 'CTBFETCH' USING CSL-CMD-HANDLE,
                        CSL-RC,
                        CS-UNUSED,
                        CS-UNUSED,
                        CS-UNUSED,
                        FF-ROWS-READ.

  EVALUATE CSL-RC

  WHEN CS-SUCCEED
    MOVE 'Y' TO SW-FETCH
    COMPUTE FF-ROW-NUM = FF-ROW-NUM + 1

*-----
* send a row of data back to the requesting client
*-----

  CALL 'TDSNDROW' USING GWL-TDPROC,
                      GWL-RC

  MOVE SPACES TO CF-COL-FIRSTNME

  IF GWL-RC NOT = TDS-OK
    THEN
      MOVE MSG-CANCELED TO INFO-MSG-STR
      MOVE CSL-RC        TO ERROR-RC
      PERFORM SEND-INFO-MESSAGE
    END-IF

```

```

WHEN CS-END-DATA
    MOVE 'N'      TO SW-FETCH

    MOVE SPACES TO INFO-MSG-STR
    PERFORM SEND-INFO-MESSAGE

    MOVE MSG-END-MSG TO INFO-MSG-STR
    PERFORM SEND-INFO-MESSAGE

    MOVE SPACES TO INFO-MSG-STR
    PERFORM SEND-INFO-MESSAGE

WHEN CS-FAIL
    MOVE 'N'      TO SW-FETCH
    MOVE 'FETCH-PROCESSING return CS-FAIL ' TO MSGSTR
    PERFORM ERROR-OUT

WHEN CS-ROW-FAIL
    MOVE 'N'      TO SW-FETCH
    MOVE 'FETCH-PROCESSING retuen CS-ROW-FAIL'
    TO MSGSTR
    PERFORM ERROR-OUT

WHEN CS-CANCELLED
    MOVE 'N'      TO SW-FETCH
    MOVE MSG-CANCELED TO MSGSTR
    PERFORM ERROR-OUT

WHEN OTHER
    MOVE 'N'      TO SW-FETCH
    MOVE 'CTBFETCH UNEXPECTED RETURN CODE'
    TO MSGSTR
    PERFORM ERROR-OUT

```

END-EVALUATE.

```

*=====
*==
*== dummy routine ==
*==
*=====
STATUS-PROCESSING.

*STATUS-PROCESSING-EXIT.
EXIT.

```

```

*=====
*==
*== Subroutine to drop the command handler, to close ==
*== the server connection, to drop the connection ==
*== handler and exit ==
*==
*=====
OC-ALL-DONE.

CALL 'CTBCMDR' USING CSL-CMD-HANDLE,
                        CSL-RC.

IF CSL-RC NOT = CS-SUCCEED
  THEN
    MOVE 'CTBCMDR failed' TO MSGSTR
    PERFORM ERROR-OUT
    PERFORM ALL-DONE
  END-IF.

CALL 'CTBCLOSE' USING CSL-CON-HANDLE,
                        CSL-RC,
                        CS-UNUSED.

IF CSL-RC NOT = CS-SUCCEED
  THEN
    MOVE 'CTBCLOSE failed' TO MSGSTR
    PERFORM ERROR-OUT
    PERFORM ALL-DONE
  END-IF.

CALL 'CTBCOND' USING CSL-CON-HANDLE,
                        CSL-RC.

IF CSL-RC NOT = CS-SUCCEED
  THEN
    MOVE 'CTBCOND failed' TO MSGSTR
    PERFORM ERROR-OUT
    PERFORM ALL-DONE
  END-IF.

*=====
*==
*== Subroutine to send an error message to the client ==
*==
*=====

```

SEND-ERROR-MESSAGE.

```

MOVE TDS-ERROR-MSG          TO SF-MESSAGE-TYPE.
MOVE 0                      TO SF-MESSAGE-NUMBER.
MOVE 10                     TO SF-SEVERITY.
MOVE 0                      TO SF-ERROR-STATE.
MOVE 0                      TO SF-LINE-ID.
MOVE LENGTH OF SF-TRANSACTION-ID TO SF-TRANSACTION-ID-LEN.
MOVE ERROR-MSG-STR         TO SF-MESSAGE-TEXT.
MOVE LENGTH OF SF-MESSAGE-TEXT TO SF-MESSAGE-LENGTH.

```

```

CALL 'TDSNDMSG' USING GWL-TDPROC,
                    GWL-RC,
                    SF-MESSAGE-TYPE,
                    SF-MESSAGE-NUMBER,
                    SF-SEVERITY,
                    SF-ERROR-STATE,
                    SF-LINE-ID,
                    SF-TRANSACTION-ID,
                    SF-TRANSACTION-ID-LEN,
                    SF-MESSAGE-TEXT,
                    SF-MESSAGE-LENGTH.

```

```

*=====
*==
*== Subroutine to send an informational message to the ==
*== client ==
*==
*=====

```

SEND-INFO-MESSAGE.

```

MOVE TDS-INFO-MSG          TO SF-MESSAGE-TYPE.
MOVE 0                      TO SF-MESSAGE-NUMBER.
MOVE 0                      TO SF-SEVERITY.
MOVE 0                      TO SF-ERROR-STATE.
MOVE 0                      TO SF-LINE-ID.
MOVE LENGTH OF SF-TRANSACTION-ID TO SF-TRANSACTION-ID-LEN.
MOVE INFO-MSG-STR         TO SF-MESSAGE-TEXT.
MOVE LENGTH OF SF-MESSAGE-TEXT TO SF-MESSAGE-LENGTH.

```

```

CALL 'TDSNDMSG' USING GWL-TDPROC,
                    GWL-RC,
                    SF-MESSAGE-TYPE,
                    SF-MESSAGE-NUMBER,
                    SF-SEVERITY,
                    SF-ERROR-STATE,

```

```

SF-LINE-ID,
SF-TRANSACTION-ID,
SF-TRANSACTION-ID-LEN,
SF-MESSAGE-TEXT,
SF-MESSAGE-LENGTH.

*=====
*==                                                    ==
*== Subroutine to print output messages.              ==
*==                                                    ==
*=====
ERROR-OUT.

    IF DIAG-MSGS-INITIALIZED
      THEN
        PERFORM GET-DIAG-MESSAGES
      END-IF.

*-----
* Display The Message
*-----

    MOVE CSL-RC      TO SAMP-RC.
    MOVE RF-TYPE     TO REST-TYPE.

    MOVE SPACES TO INFO-MSG-STR.
    PERFORM SEND-INFO-MESSAGE.

    MOVE DISP-MSG   TO INFO-MSG-STR.
    PERFORM SEND-INFO-MESSAGE.

    MOVE SPACES TO INFO-MSG-STR.
    PERFORM SEND-INFO-MESSAGE.

    MOVE SPACES TO MSGSTR.
    MOVE ZERO   TO SAMP-RC.
    MOVE ZERO   TO REST-TYPE.

PRINT-MSG-EXIT.
EXIT.

*=====
*==                                                    ==
*== Subroutine to retrieve any diagnostic messages ==
*==                                                    ==
*=====

```

GET-DIAG-MESSAGES.

* Disable calls to this subroutine *

MOVE 'N' TO SW-DIAG.

* First, get client messages *

CALL 'CTBDIAG' USING CSL-CON-HANDLE,
 CSL-RC,
 CS-UNUSED,
 CS-STATUS,
 CS-CLIENTMSG-TYPE,
 CS-UNUSED,
 DG-NUM-OF-MSGS.

IF CSL-RC NOT EQUAL CS-SUCCEED
 THEN
 STRING 'CTBDIAG CS-STATUS CS-CLIENTMSG-TYP failed'
 DELIMITED BY SIZE INTO ERROR-TEXT
 MOVE CSL-RC TO ERROR-RC
 PERFORM SEND-ERROR-MESSAGE
 PERFORM ALL-DONE
 ELSE
 IF DG-NUM-OF-MSGS > 0
 THEN
 PERFORM RETRIEVE-CLIENT-MSGS
 VARYING I FROM 1 BY 1
 UNTIL I IS GREATER THAN DG-NUM-OF-MSGS
 END-IF
 END-IF.
END-IF.

* Then, get server messages *

CALL 'CTBDIAG' USING CSL-CON-HANDLE,
 CSL-RC,
 CS-UNUSED,
 CS-STATUS,
 CS-SERVERMSG-TYPE,
 CS-UNUSED,


```

                                DG-NUM-OF-MSGS .

IF CSL-RC NOT EQUAL CS-SUCCEED
  THEN
    STRING 'CTBDIAG CS-STATUS CS-SERVERMSG-TYP fail'
              DELIMITED BY SIZE INTO ERROR-TEXT
    MOVE CSL-RC TO ERROR-RC
    PERFORM SEND-ERROR-MESSAGE
    PERFORM ALL-DONE
  ELSE
    IF DG-NUM-OF-MSGS > 0
      THEN
        PERFORM RETRIEVE-SERVER-MSGS
          VARYING I FROM 1 BY 1
            UNTIL I IS GREATER THAN DG-NUM-OF-MSGS
        END-IF
      END-IF.

GET-DIAG-MESSAGES-EXIT.
EXIT.

*=====
*==
*== Subroutine to retrieve diagnostic messages from client ==
*==
*=====
RETRIEVE-CLIENT-MSGS.

CALL 'CTBDIAG' USING CSL-CON-HANDLE,
                    CSL-RC,
                    CS-UNUSED,
                    CS-GET,
                    CS-CLIENTMSG-TYPE,
                    DG-MSGNO,
                    CLIENT-MSG.

IF CSL-RC NOT EQUAL CS-SUCCEED
  THEN
    STRING 'CTBDIAG CS-GET CS-CLIENTMSG-TYPE failed'
              DELIMITED BY SIZE INTO ERROR-TEXT
    MOVE CSL-RC TO ERROR-RC
    PERFORM SEND-ERROR-MESSAGE
    PERFORM ALL-DONE
  END-IF.

*-----

```

* display message text

*-----

```
MOVE DISP-CLIENT-MSG-HDR TO INFO-MSG-STR.
PERFORM SEND-INFO-MESSAGE.
```

```
MOVE SPACES TO INFO-MSG-STR.
PERFORM SEND-INFO-MESSAGE.
```

```
MOVE CM-SEVERITY          TO CM-SEVERITY-DATA.
MOVE CM-STATUS           TO CM-STATUS-DATA.
MOVE DISP-CLIENT-MSG-1 TO INFO-MSG-STR.
PERFORM SEND-INFO-MESSAGE.
```

```
MOVE CM-MSGNO            TO CM-OC-MSGNO-DATA.
MOVE DISP-CLIENT-MSG-2 TO INFO-MSG-STR.
PERFORM SEND-INFO-MESSAGE.
```

```
IF CM-MSGNO NOT EQUAL 0
```

```
  THEN
```

```
    MOVE SPACES          TO CM-OC-MSG-DATA
    MOVE CM-TEXT         TO CM-OC-MSG-DATA
    MOVE CM-TEXT         TO DISP-CLIENT-MSG-3A
    MOVE DISP-CLIENT-MSG-3 TO INFO-MSG-STR
    PERFORM SEND-INFO-MESSAGE
```

```
IF CM-TEXT-LEN > 66
```

```
  THEN
```

```
    MOVE CM-OC-MSG-DATA-2 TO CM-OC-MSG-DATA-X
    MOVE DISP-CLIENT-MSG-3B TO INFO-MSG-STR
    PERFORM SEND-INFO-MESSAGE
```

```
IF CM-TEXT-LEN > 132
```

```
  THEN
```

```
    MOVE SPACES          TO CM-OC-MSG-DATA-X
    MOVE CM-OC-MSG-DATA-3 TO CM-OC-MSG-DATA-X
    MOVE DISP-CLIENT-MSG-3B TO INFO-MSG-STR
    PERFORM SEND-INFO-MESSAGE
```

```
IF CM-TEXT-LEN > 198
```

```
  THEN
```

```
    MOVE SPACES          TO CM-OC-MSG-DATA-X
    MOVE CM-OC-MSG-DATA-4 TO CM-OC-MSG-DATA-X
    MOVE DISP-CLIENT-MSG-3B TO INFO-MSG-STR
    PERFORM SEND-INFO-MESSAGE
```

```
END-IF
```

```
        END-IF
      END-IF
    ELSE
      MOVE DISP-EMPTY-CLIENT-MSG-3 TO INFO-MSG-STR
      PERFORM SEND-INFO-MESSAGE
    END-IF.

MOVE CM-OS-MSGNO          TO CM-OS-MSGNO-DATA.
MOVE DISP-CLIENT-MSG-4 TO INFO-MSG-STR.
PERFORM SEND-INFO-MESSAGE.

IF CM-OS-MSGNO NOT EQUAL 0
  THEN
    MOVE SPACES           TO CM-OS-MSG-DATA
    MOVE CM-OS-MSGTXT     TO CM-OS-MSG-DATA
    MOVE SPACES           TO DISP-CLIENT-MSG-5A
    MOVE CM-OS-MSGTXT     TO DISP-CLIENT-MSG-5A
    MOVE DISP-CLIENT-MSG-5 TO INFO-MSG-STR
    PERFORM SEND-INFO-MESSAGE

    IF CM-OS-MSGTEXT-LEN > 66
      THEN
        MOVE SPACES           TO CM-OC-MSG-DATA-X
        MOVE CM-OS-MSG-DATA-2 TO CM-OC-MSG-DATA-X
        MOVE DISP-CLIENT-MSG-3B TO INFO-MSG-STR
        PERFORM SEND-INFO-MESSAGE

        IF CM-OS-MSGTEXT-LEN > 132
          THEN
            MOVE SPACES           TO CM-OC-MSG-DATA-X
            MOVE CM-OS-MSG-DATA-3 TO CM-OC-MSG-DATA-X
            MOVE DISP-CLIENT-MSG-3B TO INFO-MSG-STR
            PERFORM SEND-INFO-MESSAGE

            IF CM-OS-MSGTEXT-LEN > 198
              THEN
                MOVE SPACES           TO CM-OC-MSG-DATA-X
                MOVE CM-OS-MSG-DATA-4 TO CM-OC-MSG-DATA-X
                MOVE DISP-CLIENT-MSG-3B TO INFO-MSG-STR
                PERFORM SEND-INFO-MESSAGE
              END-IF
            END-IF
          END-IF
        ELSE
          MOVE DISP-EMPTY-CLIENT-MSG-5 TO INFO-MSG-STR
          PERFORM SEND-INFO-MESSAGE
        END-IF
      END-IF
    ELSE
      MOVE DISP-EMPTY-CLIENT-MSG-5 TO INFO-MSG-STR
      PERFORM SEND-INFO-MESSAGE
    END-IF
  END-IF
ELSE
  MOVE DISP-EMPTY-CLIENT-MSG-5 TO INFO-MSG-STR
  PERFORM SEND-INFO-MESSAGE
END-IF
```

```

        END-IF.

RETRIEVE-CLIENT-MSGS-EXIT.
        EXIT.

*=====
*==
*== Subroutine to retrieve diagnostic messages from server ==
*==
*=====
RETRIEVE-SERVER-MSGS.

        CALL 'CTBDIAG' USING CSL-CON-HANDLE,
                                CSL-RC,
                                CS-UNUSED,
                                CS-GET,
                                CS-SERVERMSG-TYPE,
                                DG-MSGNO,
                                SERVER-MSG.

        IF CSL-RC NOT EQUAL CS-SUCCEED
            THEN
                STRING 'CTBDIAG CS-GET CS-SERVERMSG-TYPE failed'
                                DELIMITED BY SIZE INTO ERROR-TEXT
                MOVE CSL-RC TO ERROR-RC
                PERFORM SEND-ERROR-MESSAGE
                PERFORM ALL-DONE
        END-IF.

*-----
* display message text
*-----

        MOVE SM-MSGNO      TO SM-MSG-NO-DATA.
        MOVE SM-SEV        TO SM-SEVERITY-DATA.
        MOVE SM-STATE      TO SM-STATE-DATA.

        MOVE SM-LINE       TO SM-LINE-NO-DATA.
        MOVE SM-STATUS     TO SM-STATUS-DATA.

        MOVE SPACES        TO SM-SVRNAME-DATA.
        MOVE SM-SVRNAME    TO SM-SVRNAME-DATA.

        MOVE SPACES        TO SM-PROC-ID-DATA.
        MOVE SM-PROC       TO SM-PROC-ID-DATA.

```

```
MOVE SPACES          TO SM-MSG-DATA.
MOVE SM-TEXT         TO SM-MSG-DATA.

MOVE SPACES          TO DISP-SERVER-MSG-5A.
MOVE SM-TEXT         TO DISP-SERVER-MSG-5A.

MOVE DISP-SERVER-MSG-HDR TO INFO-MSG-STR.
PERFORM SEND-INFO-MESSAGE.

MOVE DISP-SERVER-MSG-1   TO INFO-MSG-STR.
PERFORM SEND-INFO-MESSAGE.

MOVE DISP-SERVER-MSG-2   TO INFO-MSG-STR.
PERFORM SEND-INFO-MESSAGE.

MOVE DISP-SERVER-MSG-3   TO INFO-MSG-STR.
PERFORM SEND-INFO-MESSAGE.

MOVE DISP-SERVER-MSG-4   TO INFO-MSG-STR.
PERFORM SEND-INFO-MESSAGE.

MOVE DISP-SERVER-MSG-5   TO INFO-MSG-STR.
PERFORM SEND-INFO-MESSAGE.

IF SM-TEXT-LEN > 66
  THEN
    MOVE SPACES          TO SM-MSG-DATA-X
    MOVE SM-MSG-DATA-2   TO SM-MSG-DATA-X
    MOVE DISP-SERVER-MSG-5X TO INFO-MSG-STR
    PERFORM SEND-INFO-MESSAGE

    IF SM-TEXT-LEN > 132
      THEN
        MOVE SPACES          TO SM-MSG-DATA-X
        MOVE SM-MSG-DATA-3   TO SM-MSG-DATA-X
        MOVE DISP-SERVER-MSG-5X TO INFO-MSG-STR
        PERFORM SEND-INFO-MESSAGE

        IF SM-TEXT-LEN > 198
          THEN
            MOVE SPACES          TO SM-MSG-DATA-X
            MOVE SM-MSG-DATA-4   TO SM-MSG-DATA-X
            MOVE DISP-SERVER-MSG-5X TO INFO-MSG-STR
            PERFORM SEND-INFO-MESSAGE
          END-IF
        END-IF
      END-IF
    END-IF
  END-IF
END-IF
```

```
        END-IF  
    END-IF.  
  
RETRIEVE-SERVER-MSG-EXIT.  
EXIT.
```

Sample Tracing and Accounting Program

This appendix contains a sample mainframe server application program that a system programmer can use to perform mainframe-based tracing and accounting functions.

Note This program is not included with Open ServerConnect.

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.

This sample program uses VS COBOL II and Gateway-Library and runs under CICS.

This program demonstrates the use of the following Gateway-Library functions listed in Table G-1.

Table G-1: Functions used in SYCCSAS2

Name	Action
TDACCEPT	Accept a client request.
TDESCRIB	Describe a column.
TDFREE	Free up the TDPROC structure for the connection.
TDINFACT	Get current accounting information.
TDINFLOG	Get current trace settings for trace log.
TDINFSPT	Get specific tracing information.
TDINIT	Initialize the Gateway-Library environment.
TDLSTSPT	Get list of active specific trace transaction IDs.
TDRCVPRM	Receive RPC parameter from client program.
TDRESULT	Describe next communication from client.
TDSETACT	Set accounting on or off.
TDSETLOG	Set trace log on or off.

Name	Action
TDSETSPT	Set tracing on or off for a specific transaction.
TDSNDDON	Send results-completion to client.
TDSNDMSG	Send message to client.
TDSNDROW	Send row to client.
TDSTATUS	Get status information.
TWRTLOG	Write a user record to the trace log.

Sample program SYCCSAS2

This program uses the Gateway-Library system programmer calls to do tracing and accounting at the mainframe.

IDENTIFICATION DIVISION

*-----
PROGRAM-ID. SYCCSAS2.

ENVIRONMENT DIVISION.
DATA DIVISION.

WORKING-STORAGE SECTION.

*-----
* SERVER LIBRARY COBOL COPY BOOK
*-----
COPY SYGWCOP.

*-----
* WORK AREAS
*-----

```
01  GW-LIB-MISC-FIELDS.
    05  GWL-PROC                POINTER.
    05  GWL-INIT-HANDLE         POINTER.
    05  GWL-INFOACT-STATUS      PIC S9(9) COMP.
    05  GWL-INFOACT-FILENAME    PIC X(8) .
    05  GWL-INFOACT-RECORDS     PIC S9(9) COMP.
    05  GWL-INFOFLOG-GLOBAL     PIC S9(9) COMP.
    05  GWL-INFOFLOG-API        PIC S9(9) COMP.
    05  GWL-INFOFLOG-HEADER     PIC S9(9) COMP.
```



```

05  GWL-INFLOG-DATA          PIC S9(9) COMP.
05  GWL-INFLOG-TRACEID      PIC S9(9) COMP.
05  GWL-INFLOG-FILENAME     PIC X(8) .
05  GWL-INFLOG-RECORDS     PIC S9(9) COMP.
05  GWL-INFSPPT-STATUS      PIC S9(9) COMP.
05  GWL-INFSPPT-OPTIONS    PIC S9(9) COMP.
05  GWL-INFSPPT-TRANID     PIC X(4) .
05  GWL-INFSPPT-TRANID-L   PIC S9(9) COMP.
05  GWL-LSTSPT-LIST        OCCURS 8 TIMES
                           PIC X(8) .

05  GWL-RC                  PIC S9(9) COMP.
05  GWL-RCVPRM-ID          PIC S9(9) COMP VALUE +1.
05  GWL-RCVPRM-MAX-DATA-L  ThinSpaceThinSpacePIC S9(9) COMP VALUE +2.
05  GWL-RCVPRM-DATA-L     PIC S9(9) COMP VALUE +2
05  GWL-SETSPT-OPTIONS    PIC S9(9) COMP.
05  GWL-STATUS-NR         PIC S9(9) COMP.
05  GWL-STATUS-DONE       PIC S9(9) COMP.
05  GWL-STATUS-COUNT      PIC S9(9) COMP.
05  GWL-STATUS-COMM       PIC S9(9) COMP.
05  GWL-STATUS-RETURN-CODE PIC S9(9) COMP.
05  GWL-STATUS-SUBCODE    PIC S9(9) COMP.
05  GWL-WRTLOG-MSG-L      PIC S9(9) COMP VALUE +34.
05  GWL-WRTLOG-MSG        PIC X(34)
                           VALUE 'TEST MESSAGE FROM SYS2 TRANSACTION'.

01  PARM-FIELDS.
05  PARM-REQUEST           PIC X(2) .
   88 PARM-REQUEST-INFOCT VALUE 'IA'.
   88 PARM-REQUEST-INFLOG VALUE 'IL'.
   88 PARM-REQUEST-LSTSPT VALUE 'IS'.
   88 PARM-REQUEST-SETACT-ON VALUE 'YA'.
   88 PARM-REQUEST-SETACT-OFF VALUE 'NA'.
   88 PARM-REQUEST-SETLOG-ON VALUE 'YL'.
   88 PARM-REQUEST-SETLOG-OFF VALUE 'NL'.
   88 PARM-REQUEST-SETSPT-ON VALUE 'YS'.
   88 PARM-REQUEST-SETSPT-OFF VALUE 'NS'.
   88 PARM-REQUEST-WRTLOG VALUE 'WL'.

01  SNA-FIELDS.
05  SNA-SUBC              PIC S9(9) COMP.
05  SNA-CONNECTION-NAME   PIC X(8) VALUE SPACES.

01  COLUMN-NAME-FIELDS.
   05 CN-INFOCT-STATUS    PIC X(13) VALUE 'ACT_STATUS'
05  CN-INFOCT-FILENAME    PIC X(12) VALUE 'ACT_FILENAME'.
   05 CN-INFOCT-RECORDS   PIC X(11) VALUE 'ACT_RECORDS'.

```

```

05 CN-INFLOG-GLOBAL      PIC X(10) VALUE 'LOG GLOBAL'.
05 CN-INFLOG-API        PIC X(7)  VALUE 'LOG API'.
05 CN-INFLOG-HEADER     PIC X(10) VALUE 'LOG HEADER'.
05 CN-INFLOG-DATA       PIC X(8)  VALUE 'LOG DATA'.
05 CN-INFLOG-TRACEID    PIC X(11) VALUE 'LOG TRACEID'.
05 CN-INFLOG-FILENAME   PIC X(12) VALUE 'LOG FILENAME'.
05 CN-INFLOG-RECORDS    PIC X(11) VALUE 'LOG RECORDS'.
05 CN-LSTSPT-TRANID     PIC X(06) VALUE 'TRANID'.

01 COUNTER-FIELDS.
05 CTR-COLUMN           PIC S9(9) COMP VALUE 0.
05 CTR-ROWS             PIC S9(9) COMP VALUE 0.

01 WORK-FIELDS.
05 WRKLEN1              PIC S9(9) COMP.
05 WRKLEN2              PIC S9(9) COMP.
05 WRK-DONE-STATUS      PIC S9(9) COMP.
05 WRK-RPC              PIC X(4)  VALUE 'SYS2'.
05 WRK-TRANID           PIC X(4)  VALUE SPACE.
05 WRK-LSTSPT-SS        PIC S9(4) COMP.

01 MESSAGE-FIELDS.
05 MSG-TYPE             PIC S9(9) COMP.
05 MSG-SEVERITY-ERROR   PIC S9(9) COMP VALUE 11.
05 MSG-NR-ERROR         PIC S9(9) COMP VALUE 2.
05 MSG-RPC              PIC X(4) .
05 MSG-RPC-L            PIC S9(9) COMP VALUE 4.
05 MSG-TEXT             PIC X(20) .
05 MSG-TEXT-L           PIC S9(9) COMP.
05 MSG-SRVLIB.
   10 MSG-SRVLIB-FUNC    PIC X(8)  VALUE SPACE.
   10 FILLER             PIC X(6)  VALUE ' RC = '.
   10 MSG-SRVLIB-RC      PIC Z(4)9+.

01 SWITCHES.
05 SEND-DONE-SW         PIC X    VALUE 'Y'.
   88 SEND-DONE-ERROR    ThinSpaceThinSpaceThinSpaceThinSpaceVALUE
'N'.
   88 SEND-DONE-OK       ThinSpaceThinSpaceThinSpaceThinSpaceVALUE
'Y'.
05 TRACING-SW           PIC X    VALUE 'N'.
   88 TRACING-OFF        VALUE 'N'.
   88 TRACING-ON         VALUE 'Y'.

```

```

*****
PROCEDURE DIVISION.

```

```

*****
*-----
INITIALIZE-PROGRAM.
*-----

*
* -----
* Establish gateway environment.
* -----
CALL 'TDINIT' USING DFHEIBLK, GWL-RC, GWL-INIT-HANDLE.

*
* -----
* Accept client request.
* -----
CALL 'TDACCEPT' USING GWL-PROC, GWL-RC, GWL-INIT-HANDLE,
                    SNA-CONNECTION-NAME,
                    SNA-SUBC.

*
* -----
* Call TDRESULT to validate that request is an RPC.
* -----
CALL 'TDRESULT' USING GWL-PROC, GWL-RC.

IF GWL-RC NOT = TDS-PARM-PRESENT THEN
    MOVE 'N'          TO SEND-DONE-SW
    MOVE 'TDRESULT' TO MSG-SRVLIB-FUNC
    GO TO END-PROGRAM
END-IF.

*-----
GET-PARM.
*-----

CALL 'TDRCVPRM' USING GWL-PROC, GWL-RC,
                    GWL-RCVPRM-ID,
                    PARM-REQUEST,
                    TDSCHAR,
                    GWL-RCVPRM-MAX-DATA-L,
                    GWL-RCVPRM-DATA-L.

IF PARM-REQUEST-INFACCT THEN
    PERFORM TDINFACCT THRU TDINFACCT-EXIT

ELSE IF PARM-REQUEST-INFLOG THEN

```

```
PERFORM TDINFLOG THRU TDINFLOG-EXIT

ELSE IF PARM-REQUEST-LSTSPT THEN
  PERFORM TDLSTSPT THRU TDLSTSPT-EXIT

ELSE IF PARM-REQUEST-SETACT-ON THEN
  PERFORM TDSETACT-ON THRU TDSETACT-ON-EXIT

ELSE IF PARM-REQUEST-SETACT-OFF THEN
  PERFORM TDSETACT-OFF THRU TDSETACT-OFF-EXIT

ELSE IF PARM-REQUEST-SETLOG-ON THEN
  PERFORM TDSETLOG-ON THRU TDSETLOG-ON-EXIT

ELSE IF PARM-REQUEST-SETLOG-OFF THEN
  PERFORM TDSETLOG-OFF THRU TDSETLOG-OFF-EXIT

ELSE IF PARM-REQUEST-SETSPT-ON THEN
  PERFORM TDSETSPT-ON THRU TDSETSPT-ON-EXIT

ELSE IF PARM-REQUEST-SETSPT-OFF THEN
  PERFORM TDSETSPT-OFF THRU TDSETSPT-OFF-EXIT

ELSE IF PARM-REQUEST-WRTLOG THEN
  PERFORM TDWRTLOG THRU TDWRTLOG-EXIT
END-IF.
```

```
*-----
END-PROGRAM.
```

```
*-----
```

```
IF SEND-DONE-OK THEN
  MOVE TDS-DONE-COUNT TO WRK-DONE-STATUS
ELSE
  PERFORM SRVLIB-ERROR THRU SRVLIB-ERROR-EXIT
  MOVE TDS-DONE-ERROR TO WRK-DONE-STATUS
  MOVE ZERO           TO CTR-ROWS
END-IF.
```

```
CALL 'TDSNDDON' USING GWL-PROC, GWL-RC,
                    WRK-DONE-STATUS,
                    CTR-ROWS,
                    TDS-ZERO,
                    TDS-ENDRPC.
```

```
CALL 'TDFREE' USING GWL-PROC, GWL-RC.
STOP RUN.
```

```

*-----
TDINFACT.
*-----

MOVE LENGTH OF GWL-INFACT-STATUS TO WRKLEN1.
MOVE LENGTH OF CN-INFACT-STATUS TO WRKLEN2.
ADD +1                                TO CTR-COLUMN.
MOVE 'TDESCRIB'                      TO MSG-SRVLIB-FUNC.

CALL 'TDESCRIB' USING GWL-PROC, GWL-RC,
                    CTR-COLUMN,
                    TDSINT4,
                    WRKLEN1,
                    GWL-INFACT-STATUS,
                    TDS-ZERO,
                    TDS-FALSE,
                    TDSINT4,
                    WRKLEN1,
                    CN-INFACT-STATUS,
                    WRKLEN2.

IF GWL-RC NOT = TDS-OK THEN
    MOVE 'N' TO SEND-DONE-SW
    GO TO TDINFACT-EXIT
END-IF.

MOVE LENGTH OF GWL-INFACT-FILENAME TO WRKLEN1.
MOVE LENGTH OF CN-INFACT-FILENAME TO WRKLEN2.
ADD +1                                TO CTR-COLUMN.
CALL 'TDESCRIB' USING GWL-PROC, GWL-RC,
                    CTR-COLUMN,
                    TDSCHAR,
                    WRKLEN1,
                    GWL-INFACT-FILENAME,
                    TDS-ZERO,
                    TDS-FALSE,
                    TDSCHAR,
                    WRKLEN1,
                    CN-INFACT-FILENAME,
                    WRKLEN2.

IF GWL-RC NOT = TDS-OK THEN
    MOVE 'N' TO SEND-DONE-SW
    GO TO TDINFACT-EXIT

```

```
END-IF.

MOVE LENGTH OF GWL-INFACT-RECORDS TO WRKLEN1.
MOVE LENGTH OF CN-INFACT-RECORDS TO WRKLEN2.
ADD +1                                TO CTR-COLUMN.

CALL 'TDESCRIB' USING GWL-PROC, GWL-RC,
                      CTR-COLUMN,
                      TDSINT4,
                      WRKLEN1,
                      GWL-INFACT-RECORDS,
                      TDS-ZERO,
                      TDS-FALSE,
                      TDSINT4,
                      WRKLEN1,
                      CN-INFACT-RECORDS,
                      WRKLEN2.

IF GWL-RC NOT = TDS-OK THEN
    MOVE 'N' TO SEND-DONE-SW
    GO TO TDINFACT-EXIT
END-IF.

CALL 'TDINFACT' USING GWL-INIT-HANDLE, GWL-RC,
                      GWL-INFACT-STATUS,
                      GWL-INFACT-FILENAME,
                      GWL-INFACT-RECORDS.

IF GWL-RC NOT = TDS-OK THEN
    MOVE 'N'          TO SEND-DONE-SW
    MOVE 'TDINFACT' TO MSG-SRVLIB-FUNC
    GO TO TDINFACT-EXIT
END-IF.

CALL 'TDSNDROW' USING GWL-PROC, GWL-RC.

IF GWL-RC NOT = TDS-OK THEN
    MOVE 'N'          TO SEND-DONE-SW
    MOVE 'TDSNDROW' TO MSG-SRVLIB-FUNC
    GO TO TDINFACT-EXIT
END-IF.

ADD +1 TO CTR-ROWS.
```

```
*-----
TDINFACT-EXIT.
```

```

*-----
EXIT.

*-----
TDINFLOG.
*-----
MOVE LENGTH OF GWL-INFLOG-GLOBAL TO WRKLEN1.
MOVE LENGTH OF CN-INFLOG-GLOBAL TO WRKLEN2.
ADD +1                                TO CTR-COLUMN.
MOVE 'TDESCRIB'                      TO MSG-SRVLIB-FUNC.

CALL 'TDESCRIB' USING GWL-PROC, GWL-RC,
                      CTR-COLUMN,
                      TDSINT4,
                      WRKLEN1,
                      GWL-INFLOG-GLOBAL,
                      TDS-ZERO,
                      TDS-FALSE,
                      TDSINT4,
                      WRKLEN1,
                      CN-INFLOG-GLOBAL,
                      WRKLEN2.

IF GWL-RC NOT = TDS-OK THEN
    MOVE 'N' TO SEND-DONE-SW
    GO TO TDINFLOG-EXIT
END-IF.

MOVE LENGTH OF GWL-INFLOG-API TO WRKLEN1.
MOVE LENGTH OF CN-INFLOG-API TO WRKLEN2.
ADD +1                                TO CTR-COLUMN.

CALL 'TDESCRIB' USING GWL-PROC, GWL-RC,
                      CTR-COLUMN,
                      TDSINT4,
                      WRKLEN1,
                      GWL-INFLOG-API,
                      TDS-ZERO,
                      TDS-FALSE,
                      TDSINT4,
                      WRKLEN1,
                      CN-INFLOG-API,
                      WRKLEN2.

IF GWL-RC NOT = TDS-OK THEN
    MOVE 'N' TO SEND-DONE-SW

```

```
GO TO TDINFLOG-EXIT
END-IF.

MOVE LENGTH OF GWL-INFLOG-HEADER TO WRKLEN1.
MOVE LENGTH OF CN-INFLOG-HEADER TO WRKLEN2.
ADD +1 TO CTR-COLUMN.
```

```
CALL 'TDESCRIB' USING GWL-PROC, GWL-RC,
                      CTR-COLUMN,
                      TDSINT4,
                      WRKLEN1,
                      GWL-INFLOG-HEADER,
                      TDS-ZERO,
                      TDS-FALSE,
                      TDSINT4,
                      WRKLEN1,
                      CN-INFLOG-HEADER,
                      WRKLEN2.
```

```
IF GWL-RC NOT = TDS-OK THEN
    MOVE 'N' TO SEND-DONE-SW
    GO TO TDINFLOG-EXIT
END-IF.
```

```
MOVE LENGTH OF GWL-INFLOG-DATA TO WRKLEN1.
MOVE LENGTH OF CN-INFLOG-DATA TO WRKLEN2.
ADD +1 TO CTR-COLUMN.
```

```
CALL 'TDESCRIB' USING GWL-PROC, GWL-RC,
                      CTR-COLUMN,
                      TDSINT4,
                      WRKLEN1,
                      GWL-INFLOG-DATA,
                      TDS-ZERO,
                      TDS-FALSE,
                      TDSINT4,
                      WRKLEN1,
                      CN-INFLOG-DATA, WRKLEN2.
```

```
IF GWL-RC NOT = TDS-OK THEN
    MOVE 'N' TO SEND-DONE-SW
    GO TO TDINFLOG-EXIT
END-IF.
```



```

MOVE LENGTH OF GWL-INFLOG-TRACEID TO WRKLEN1.
MOVE LENGTH OF CN-INFLOG-TRACEID TO WRKLEN2.
ADD +1                                TO CTR-COLUMN.

```

```

CALL 'TDESCRIB' USING GWL-PROC, GWL-RC,
                      CTR-COLUMN,
                      TDSINT4,
                      WRKLEN1,
                      GWL-INFLOG-TRACEID,
                      TDS-ZERO,
                      TDS-FALSE,
                      TDSINT4,
                      WRKLEN1,
                      CN-INFLOG-TRACEID,
                      WRKLEN2.

```

```

IF GWL-RC NOT = TDS-OK THEN
    MOVE 'N' TO SEND-DONE-SW
    GO TO TDINFLOG-EXIT
END-IF.

```

```

MOVE LENGTH OF GWL-INFLOG-FILENAME TO WRKLEN1.
MOVE LENGTH OF CN-INFLOG-FILENAME TO WRKLEN2.
ADD +1                                TO CTR-COLUMN.

```

```

CALL 'TDESCRIB' USING GWL-PROC, GWL-RC,
                      CTR-COLUMN,
                      TDSCHAR,
                      WRKLEN1,
                      GWL-INFLOG-FILENAME,
                      TDS-ZERO,
                      TDS-FALSE,
                      TDSCHAR,
                      WRKLEN1,
                      CN-INFLOG-FILENAME,
                      WRKLEN2.

```

```

IF GWL-RC NOT = TDS-OK THEN
    MOVE 'N' TO SEND-DONE-SW
    GO TO TDINFLOG-EXIT
END-IF.

```

```

MOVE LENGTH OF GWL-INFLOG-RECORDS TO WRKLEN1.
MOVE LENGTH OF CN-INFLOG-RECORDS TO WRKLEN2.
ADD +1                                TO CTR-COLUMN.

```

```
CALL 'TDESCRIB' USING GWL-PROC, GWL-RC,  
                     CTR-COLUMN,  
                     TDSINT4,  
                     WRKLEN1,  
                     GWL-INFLOG-RECORDS,  
                     TDS-ZERO,  
                     TDS-FALSE,  
                     TDSINT4,  
                     WRKLEN1,  
                     CN-INFLOG-RECORDS,  
                     WRKLEN2.
```

```
IF GWL-RC NOT = TDS-OK THEN  
    MOVE 'N' TO SEND-DONE-SW  
    GO TO TDINFLOG-EXIT  
END-IF.
```

```
CALL 'TDINFLOG' USING 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 NOT = TDS-OK THEN  
    MOVE 'N'          TO SEND-DONE-SW  
    MOVE 'TDINFLOG' TO MSG-SRVLIB-FUNC  
    GO TO TDINFLOG-EXIT  
END-IF.
```

```
CALL 'TDSNDROW' USING GWL-PROC, GWL-RC.
```

```
IF GWL-RC NOT = TDS-OK THEN  
    MOVE 'N'          TO SEND-DONE-SW  
    MOVE 'TDSNDROW' TO MSG-SRVLIB-FUNC  
    GO TO TDINFLOG-EXIT  
END-IF.
```

```
ADD +1 TO CTR-ROWS.
```

```

*-----
TDINFLOG-EXIT.
*-----
      EXIT.

*-----
TDLSTSPT.
*-----
      MOVE LENGTH OF WRK-TRANID          TO WRKLEN1.
      MOVE LENGTH OF CN-LSTSPT-TRANID TO WRKLEN2.
      ADD +1                             TO CTR-COLUMN.

      CALL 'TDESCRIB' USING GWL-PROC, GWL-RC,
                          CTR-COLUMN,
                          TDSCHAR,
                          WRKLEN1,
                          WRK-TRANID,
                          TDS-ZERO,
                          TDS-FALSE,
                          TDSCHAR,
                          WRKLEN1,
                          CN-LSTSPT-TRANID,
                          WRKLEN2.

      IF GWL-RC NOT = TDS-OK THEN
          MOVE 'N'          TO SEND-DONE-SW
          MOVE 'TDESCRIB' TO MSG-SRVLIB-FUNC
          GO TO TDLSTSPT-EXIT
      END-IF.

*
* Find global status.
*

      CALL 'TDINFLOG' USING 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 there are any errors, then assume tracing has been disabled.
*

```

```

IF GWL-RC NOT = TDS-OK THEN
    MOVE 'N'          TO SEND-DONE-SW
    MOVE 'TDINFLOG' TO MSG-SRVLIB-FUNC
    GO TO TDLSTSPT-EXIT
END-IF.

*
* If specific tracing is not on, then return nothing.
*
IF GWL-INFLOG-GLOBAL NOT = TDS-TRACE-SPECIFIC-RPCS THEN
    GO TO TDLSTSPT-EXIT
END-IF.

*
* Return rows.
*
CALL 'TDLSTSPT' USING GWL-INIT-HANDLE, GWL-RC,
                    GWL-LSTSPT-LIST(1) .

IF GWL-RC NOT = TDS-OK THEN
    MOVE 'N'          TO SEND-DONE-SW
    MOVE 'TDLSTSPT' TO MSG-SRVLIB-FUNC
    GO TO TDLSTSPT-EXIT
END-IF.
PERFORM VARYING WRK-LSTSPT-SS FROM 1 BY 1
    UNTIL WRK-LSTSPT-SS = 8

    MOVE GWL-LSTSPT-LIST(WRK-LSTSPT-SS) TO WRK-TRANID
    CALL 'TDSNDROW' USING GWL-PROC, GWL-RC

    IF GWL-RC NOT = TDS-OK THEN
        MOVE 'N'          TO SEND-DONE-SW
        MOVE 'TDSNDROW' TO MSG-SRVLIB-FUNC
        MOVE 8            TO WRK-LSTSPT-SS
    END-IF

    ADD +1 TO CTR-ROWS

END-PERFORM.

*-----
TDLSTSPT-EXIT.
*-----

EXIT.

*-----
TDSETACT-ON.
*-----

```

```

CALL 'TDINFACT' USING GWL-INIT-HANDLE, GWL-RC,
                    GWL-INFACT-STATUS,
                    GWL-INFACT-FILENAME,
                    GWL-INFACT-RECORDS.

IF GWL-RC NOT = TDS-OK THEN
    MOVE 'N'          TO SEND-DONE-SW
    MOVE 'TDINFACT' TO MSG-SRVLIB-FUNC
    GO TO TDSETACT-ON-EXIT
END-IF.
*
* Turn on host accounting.
*
CALL 'TDSETACT' USING GWL-INIT-HANDLE, GWL-RC,
                    TDS-TRUE,
                    GWL-INFACT-FILENAME,
                    GWL-INFACT-RECORDS.

IF GWL-RC NOT = TDS-OK THEN
    MOVE 'N'          TO SEND-DONE-SW
    MOVE 'TDSETACT' TO MSG-SRVLIB-FUNC
    GO TO TDSETACT-ON-EXIT
END-IF.

*-----
TDSETACT-ON-EXIT.
*-----

EXIT.

*-----
TDSETACT-OFF.
*-----

CALL 'TDINFACT' USING GWL-INIT-HANDLE, GWL-RC,
                    GWL-INFACT-STATUS,
                    GWL-INFACT-FILENAME,
                    GWL-INFACT-RECORDS.

IF GWL-RC NOT = TDS-OK THEN
    MOVE 'N'          TO SEND-DONE-SW
    MOVE 'TDINFACT' TO MSG-SRVLIB-FUNC
    GO TO TDSETACT-OFF-EXIT
END-IF.
*
* Turn off host accounting if it is on.

```

```
*
  IF GWL-INFACCT-STATUS = TDS-TRUE THEN
    CALL 'TDSETACT' USING GWL-INIT-HANDLE, GWL-RC, TDS-FALSE,
                        GWL-INFACCT-FILENAME, GWL-INFACCT-RECORDS

    IF GWL-RC NOT = TDS-OK THEN
      MOVE 'N'          TO SEND-DONE-SW
      MOVE 'TDSETACT' TO MSG-SRVLIB-FUNC
      GO TO TDSETACT-OFF-EXIT
    END-IF
  END-IF.

*-----
  TDSETACT-OFF-EXIT.
*-----

  EXIT.

*-----
  TDSETLOG-ON.
*-----

  CALL 'TDINFLOG' USING 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 NOT = TDS-OK THEN
    MOVE 'N'          TO SEND-DONE-SW
    MOVE 'TDINFLOG' TO MSG-SRVLIB-FUNC
    GO TO TDSETLOG-ON-EXIT
  END-IF.

*
* Turn on API (CICS Aux Trace) and header tracing.
*

  CALL 'TDSETLOG' USING 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 NOT = TDS-OK THEN
    MOVE 'N'          TO SEND-DONE-SW
    MOVE 'TDSETLOG' TO MSG-SRVLIB-FUNC
    GO TO TDSETLOG-ON-EXIT
END-IF.

```

```

*-----
TDSETLOG-ON-EXIT.
*-----
EXIT.

```

```

*-----
TDSETLOG-OFF.
*-----

```

```

CALL 'TDINFLOG' USING 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 NOT = TDS-OK THEN
    MOVE 'N'          TO SEND-DONE-SW
    MOVE 'TDINFLOG' TO MSG-SRVLIB-FUNC
    GO TO TDSETLOG-OFF-EXIT
END-IF.

```

```

*
* Turn off API (CICS Aux Trace) and header tracing.
*

```

```

CALL 'TDSETLOG' USING 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 NOT = TDS-OK THEN
    MOVE 'N'          TO SEND-DONE-SW
    MOVE 'TDSETLOG' TO MSG-SRVLIB-FUNC
    GO TO TDSETLOG-OFF-EXIT
END-IF.

```

```
*-----
TDSETLOG-OFF-EXIT.
*-----
      EXIT.

*-----
TDSETSPT-ON.
*-----
      CALL 'TDINFLOG' USING 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 NOT = TDS-OK THEN
          MOVE 'N'          TO SEND-DONE-SW
          MOVE 'TDINFLOG' TO MSG-SRVLIB-FUNC
          GO TO TDSETSPT-ON-EXIT
      END-IF.

*
* Turn on tracing for specific transactions.
*
      CALL 'TDSETLOG' USING 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 NOT = TDS-OK THEN
          MOVE 'N'          TO SEND-DONE-SW
          MOVE 'TDSETLOG' TO MSG-SRVLIB-FUNC
          GO TO TDSETSPT-ON-EXIT
      END-IF.

*
* Enable error log recording for this tranid.
*
      MOVE 2                TO GWL-SETSPT-OPTIONS.
      MOVE LENGTH OF WRK-RPC TO WRKLEN1.

      CALL 'TDSETSPT' USING GWL-INIT-HANDLE, GWL-RC,
```



```

                                TDS-TRUE,
                                GWL-SETSPT-OPTIONS,
                                WRK-RPC,
                                WRKLEN1.

IF GWL-RC NOT = TDS-OK THEN
    MOVE 'N'          TO SEND-DONE-SW
    MOVE 'TDSETSPT' TO MSG-SRVLIB-FUNC
    GO TO TDSETSPT-ON-EXIT
END-IF.

*-----
TDSETSPT-ON-EXIT.
*-----
    EXIT.

*-----
TDSETSPT-OFF.
*-----
*
*   Assume specific tracing is on for this transaction,
*   and turn it off.
*
    MOVE LENGTH OF WRK-RPC TO WRKLEN1.

    CALL 'TDSETSPT' USING GWL-INIT-HANDLE, GWL-RC,
                        TDS-FALSE,
                        GWL-SETSPT-OPTIONS,
                        WRK-RPC,
                        WRKLEN1.

IF GWL-RC NOT = TDS-OK
    AND GWL-RC NOT = TDS-ENTRY-NOT-FOUND THEN
    MOVE 'N'          TO SEND-DONE-SW
    MOVE 'TDSETSPT' TO MSG-SRVLIB-FUNC
    GO TO TDSETSPT-OFF-EXIT
END-IF.

*-----
TDSETSPT-OFF-EXIT.
*-----
    EXIT.

*-----
TDWRTLOG.
*-----

```

```
*
* Write a log entry only if logging is enabled.
*
PERFORM GET-TRACE-STATUS THRU GET-TRACE-STATUS-EXIT.

IF TRACING-ON THEN
    CALL 'TDWRTLOG' USING GWL-PROC, GWL-RC,
                        TDS-TRUE,
                        GWL-WRTLOG-MSG,
                        GWL-WRTLOG-MSG-L

    IF GWL-RC NOT = TDS-OK THEN
        MOVE 'N'          TO SEND-DONE-SW
        MOVE 'TDWRTLOG' TO MSG-SRVLIB-FUNC
        GO TO TDWRTLOG-EXIT
    END-IF
ELSE
    MOVE 'N'          TO SEND-DONE-SW
    MOVE 'LOGNOTON' TO MSG-SRVLIB-FUNC
END-IF.

*-----
TDWRTLOG-EXIT.
*-----

EXIT.

*-----
GET-TRACE-STATUS.
*-----
*
* Find global status.
*
CALL 'TDINFLOG' USING 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 there are any errors, then assume tracing has been disabled.
*
IF GWL-RC NOT = TDS-OK THEN
    MOVE 'N'          TO SEND-DONE-SW
    MOVE 'TDINFLOG' TO MSG-SRVLIB-FUNC
```

```

        GO TO GET-TRACE-STATUS-EXIT
    END-IF.
*
*   If global tracing is on, then tracing is enabled.
*
    IF GWL-INFLOG-GLOBAL = TDS-TRACE-ALL-RPCS THEN
        MOVE 'Y' TO TRACING-SW
        GO TO GET-TRACE-STATUS-EXIT
    END-IF.
*
*   If error logging is on, then tracing is enabled.
*
    IF GWL-INFLOG-GLOBAL = TDS-TRACE-ERRORS-ONLY THEN
        MOVE 'Y' TO TRACING-SW
        GO TO GET-TRACE-STATUS-EXIT
    END-IF.
*
*   If specific tracing is not on, then no tracing is on.
*
    IF GWL-INFLOG-GLOBAL NOT = TDS-TRACE-SPECIFIC-RPCS THEN
        GO TO GET-TRACE-STATUS-EXIT
    END-IF.
*
*   Specific tracing is on, see if on for this transaction.
*
    MOVE LENGTH OF WRK-RPC TO WRKLEN1.

    CALL 'TDINFSPT' USING GWL-INIT-HANDLE, GWL-RC,
                        GWL-INFSPPT-STATUS,
                        GWL-INFSPPT-OPTIONS,
                        WRK-RPC,
                        WRKLEN1.

    IF GWL-RC NOT = TDS-OK AND
       GWL-RC NOT = TDS-ENTRY-NOT-FOUND THEN
        MOVE 'N'           TO SEND-DONE-SW
        MOVE 'TDINFSPT' TO MSG-SRVLIB-FUNC
        GO TO GET-TRACE-STATUS-EXIT
    END-IF.

    IF GWL-INFSPPT-STATUS = TDS-TRUE THEN
        MOVE 'Y' TO TRACING-SW
    END-IF.
*-----
    GET-TRACE-STATUS-EXIT.
*-----

```

```
EXIT.
*-----
SRVLIB-ERROR.
*-----
MOVE GWL-RC                TO MSG-SRVLIB-RC.
MOVE MSG-SRVLIB            TO MSG-TEXT.
MOVE LENGTH OF MSG-SRVLIB TO MSG-TEXT-L.
MOVE TDS-ERROR-MSG        TO MSG-TYPE.
MOVE WRK-RPC               TO MSG-RPC.
*-----
SEND-ERROR-MESSAGE.
*-----
MOVE 'N'                    TO SEND-DONE-SW.
MOVE TDS-ERROR-MSG         TO MSG-TYPE.
MOVE LENGTH OF MSG-RPC     TO MSG-RPC-L.

*   Ensure we're in right state to send a message

CALL 'TDSTATUS' USING 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 AND
    GWL-STATUS-COMM = TDS-RECEIVE) THEN

    CALL 'TDSNDMSG' USING 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-IF.
*-----
SRVLIB-ERROR-EXIT.
*-----
EXIT.
```

Index

A

- Access code, customization 35
- Accounting
 - retrieving accounting information at mainframe 112
 - sample program 385
 - setting on and off at mainframe 170
 - tracing functions used in 15
- Accounting log
 - changing log name under CICS 173
 - IMS TM log 170
 - no record limit under IMS TM 113, 171
 - querying log name under CICS 113, 115
 - querying record number under CICS 113
 - setting record limit under CICS 171
 - specifying log name under CICS 170
 - uses IMS TM log 113
 - uses IMS TM system log 115, 173
- Alphabetic characters
 - single-byte characters treated as 58

C

- Character sets
 - client, list of supported 18
 - datatypes used with Japanese characters 60
 - differences in Japanese 59
 - discussion 17, 19
 - double-byte, mainframe supported 19
 - double-byte, setting up support 58
 - double-byte, workstation supported 19
 - how single-byte are handled at mainframe 110
 - Japanese, querying Shift Out/Shift In settings 65
 - querying client character set 53, 109
 - querying support for double-byte 110
 - single-byte, mainframe supported 19
 - single-byte, setting default 35
 - single-byte, workstation supported 18

- table translation 59
- Character sets. See also Kanji, Katakana, Japanese, DBCS and SBCS 67
- Client
 - character set, supported 18
 - querying character set 53, 109
 - querying information 108
 - querying name 53
 - querying national language 109
 - querying password 53, 109
 - querying userID 109
- Client requests
 - processing 6, 8, 10
 - query types 53
 - querying type of 167
 - querying types of 134
- Client requests, processing with Gateway-Library 5
- Client-Library
 - Gateway-Library application functions 56
 - Gateway-Library datatypes 36, 40
- COBOL data descriptions, Gateway-Library datatypes 36, 40
- Column
 - describing host variable for 87
 - describing name 87
 - describing with TDESCRIB 87
 - specifying decimal places 174
 - specifying length 87, 174, 180
 - specifying type 87
- Communication state
 - determining 135, 228
- Communication states 19
- Connections
 - handles for allocating 70
 - logical, definition 4
- Connections, establishing 5
- Conversation, defined 4
- Conversion
 - to DB-Library datatypes 77
- CTBCTXALLOC

Index

in mixed-mode applications, do not use 57
CURSOR-DESC structure 26

Cursors

benefits of using 21
close 25
command types 22
CURSOR-DESC structure 26
declare 23
definition 21
fetch rows 24
handling cursor requests 31, 34
request status 25
statements defined 21
TDS-CURSOR event handler 31

Customization

client password access code 35
Gateway-Library options 35
national language 35
truncating LONG VARCHAR strings 35
Customization, dynamic network drivers 35

D

Datatypes

binary, Client-Library equivalent 37
binary, DB-Library equivalent 37
binary, description 37
binary, TDSVARYBIN 42
character, Client-Library equivalent 37
character, conversion from decimal 42
character, DB2 LONG VARCHAR 41
character, DB-Library equivalent 37
character, description 37
character, TDSVARYCHAR 41
datetime, description 37
DB2 LONG VARCHAR 41
DB2 LONG VARCHAR, rejection or truncation 35
decimal, conversion to character 42
discussion 36, 45
float, 4-byte and 8-byte supported 37
graphic, description 37
Japanese character sets 60
list of supported 36
packed decimal, converting and unpacking 42
TDS-PACKED-DECIMAL, formula for unpacking

42

used with Japanese character sets 60, 62
variable, TDSVARYBIN 42
variable, TDSVARYCHAR 41

DBCS

customization 35
mainframe supported 19
setting up support 58
workstation supported 19

DBCS. See also Kanji, Katakana, Japanese, Character sets and SBCS 67

dec_kanji, support 19

Decimal

converting packed decimal with TDCONVRT 77
converting packed decimal with TDRCVPRM 160

Decimal places

assigning during conversion 77

DFHEIBLK

defining 142

DONE

sending to client 206

Double Byte Character Set. See DBCS 67

Dynamic network drivers, customization 35

E

End of results

indicating 206

Error log

tracing errors 183
tracing RPCs 183
turn off tracing 183
used for tracing 124, 186

Errors

sending error messages to client 213

eucljjs, supported 19

Example

accounting 385
language handler 307
RPC application 253, 319, 333
tracing 385

F

Function call sequence 9, 19

Functions

- definition in Gateway-Library 2
- how called with Gateway-Library 2
- paired with client functions 5
- TDACCEPT 70
- TDCONVRT 77
- TDCURPRO 83
- TDESCRIB 87, 95
- TDFREE 95, 98
- TDGETREQ 98, 104
- TDGETSOI 104, 108
- TDGETUSR 108, 112
- TDINFACT 112, 116
- TDINFBCD 116, 120
- TDINFLOG 121, 124
- TDINFPGM 125, 129
- TDINFPRM 129, 133
- TDINFRPC 134
- TDINFSPT 136, 140
- TDINFUDT 140, 142
- TDINIT 142, 146
- TDLOCPRM 146
- TDLSTSPT 149, 152
- TDNUMPRM 152, 154
- TDRCVPRM 154, 162
- TDRCVSQL 162
- TDRESULT 167, 169
- TDSETACT 170, 173
- TDSETBCD 174, 180
- TDSETLEN 180, 182
- TDSETLOG 183, 188
- TDSETPRM 188, 193
- TDSETPT 193, 196
- TDSETSOI 196, 200
- TDSETSPT 200, 204
- TDSETUDT 204, 206
- TDSNDDON 206, 213
- TDSNDMSG 213, 220
- TDSNDROW 220, 224
- TDSQLLEN 224, 227
- TDSTATUS 227, 232
- TDTERM 232, 234
- TDWRTLOG 238, 241
- TDYNAMIC 234, 238

G

Gateway-Library

- functions 3
- functions, list of 67
- initializing 4
- overview 1, 15

Global tracing

- querying 99, 100, 105, 197

H

Handles

- allocating context handles 143
- connection, allocating 70
- Gateway-Library equivalent command handle 70
- Gateway-Library equivalent connection handle 70
- Gateway-Library equivalent context handle 70

Hankaku katakana

- datatypes used with 60
- single-byte characters treated as 58
- string lengths 61

I

ibm_kanji, support 19

IHANDLE

- in TDINIT 142
- in TDACCEPT 70

IMS TM

- coding differences with CICS 2
- implicit API, sample RPC program 319
- simulating long-running transactions in implicit API 99

IMS TM implicit API

- no support for long-running transactions 102

Initialization

- customization 36
- of TDS environment 142
- TDS environment 4

J

Japanese characters

Index

- differences between character sets 59
- Japanese Conversion Module. See JCM 58
- Japanese support
 - character sets 59
 - characters length considerations 62
 - datatypes 60, 62
 - discussion 58
 - for DBCS 58
 - string lengths 60
 - table translation 59
- Japanese. See also Kanji, Katakana, Character sets, DBCS and SBCS 67
- JCM
 - how it works 59
 - table translation 59

K

- Kanji
 - character set support 59
 - datatypes used with 60
 - differences among character sets 59
 - discussion 58
 - length considerations 62
 - string lengths 60
- Katakana
 - hankaku, datatypes used with 60
 - length considerations 62
 - single-byte datatypes used with 60
- Katakana. See also Kanji, Japanese, Character sets, DBCS and SBCS 67

L

- Language request, long-running transaction 98
- Language requests
 - accepting 70
 - custom-written sample program 6
 - processing 10
 - sample program 307
- Logical connection, defined 4
- Login information, querying 53
- Login packet
 - description and contents 53

- retrieving contents with TDGETUSR 53
- Long datatypes
 - character, handling 41
- Long-running transaction
 - definition 20, 54
- Long-running transactions
 - no support for IMS TM implicit API 102, 212
- Lower-case letters
 - single-byte characters treated as 58

M

- Message
 - sending to client 213
- Mixed-mode applications
 - discussion 56

N

- National languages
 - client, querying 53
 - discussion 57
 - returned by TDGETUSR 57
 - setting during customization 35
- Native language, customization 35
- Net-Gateway
 - identifying 53

P

- Packed decimal
 - converting with TDCONVRT 77
 - converting with TDRCVPRM 160
- Parameter
 - counting 152
 - determining datatype 129
 - determining ID 146
 - determining length 129
 - determining name 129
 - identifying return parameter 129
 - retrieving 154
 - specifying decimal places 174
 - specifying length 174

- specifying return parameter data 188
- specifying return parameter datatype 188
- specifying return parameter length 188
- Parameters omitted, denoting with TDSVOID 40
- Password
 - client access code customization 35
 - client, querying 53

R

- Receive state 19
- Receiving
 - client requests, general 5
 - language requests 70
 - RPCs 70
- Reset state 20
- Row
 - describing column 87
 - sending row data to client 220
- RPC
 - accepting 70
 - identifying 167
 - language request in long-running transaction 98
 - number of parameters 152
 - processing 6, 9
 - processing, sample program 253, 319, 333
 - querying RPC name 134
 - querying RPC parameters 129
 - retrieving status 227

S

- Sample program
 - accounting 385
 - language handler 307
 - RPC application 253, 319, 333
 - tracing 385
- SBCS
 - host, default 58
 - mainframe supported 19
 - workstation supported 18
- SBCS. See also Kanji, Katakana, Japanese, Character set and DBCS 67
- Send state 19

- Session, establishing 5
- Shift Out/Shift In codes
 - stripping 65
- Short transaction, definition 20
- Single Byte Character Set. See SBCS 67
- sjis
 - supported 18
- SNA conversation
 - handle for 70
- SNA conversation, establishing 5
- Specific tracing
 - querying 99, 100, 105, 197
- SQL request
 - processing 6
 - retrieving 162
 - retrieving status 227
- SQL text
 - accepting 70
 - determining length 224
- States. See Communication, Receive and Send states 19
- Status
 - retrieving RPC status 227
 - retrieving SQL request status 227
- SYL2
 - listing 307
- SYR2
 - listing 253, 319, 333
- SYS2
 - listing 385

T

- Table translation 59
- Tabular Data Stream. See TDS 2
- TCP/IP
 - embedded calls 2
- TDACCEPT
 - description 70
 - in mixed-mode application 57
 - used with long-running transactions 54, 55
- TDCONVRT
 - description 77
- TDCURPRO
 - description 83

Index

- TDESCRIB
 - description 87, 95
 - use with JCM 95
- TDFREE
 - description 95, 98
 - in mixed-mode application 57
- TDGETREQ
 - description 98, 104
 - use with IMS TM 102
 - used with long-running transactions 54
- TDGETSOI
 - description 104, 108
- TDGETUSR
 - description 108, 112
 - retrieving login packet data 53
- TDINFACT
 - description 112, 116
- TDINFBCD
 - description 116, 120
- TDINFLOG
 - description 121, 124
- TDINFPGM
 - description 125, 129
- TDINFPRM
 - description 129, 133
- TDINFRPC
 - description 134
- TDINFSPPT
 - description 136, 140
- TDINFUDT
 - description 140, 142
- TDINIT
 - description 142, 146
 - in mixed-mode application, must be first call 56
 - use with JCM 146
- TDLOCPRM
 - description 146
- TDLSTSPPT
 - description 149, 152
- TDNUMPRM
 - description 152, 154
- TDPROC
 - allocating 70
 - deallocating 95
 - initializing 70
- TDRCVPRM
 - description 154, 162
 - use with JCM 161
- TDRCVSQL
 - description 162
 - use with JCM 166
- TDRESULT
 - description 167, 169
- TDS
 - protocol 2
- TDS environment
 - initializing 142
- TDSBINARY
 - Client-Library equivalent 37
 - COBOL data description 37
 - DB-Library equivalent 37
 - description 37
- TDSCHAR
 - Client-Library equivalent 37
 - COBOL data description 37
 - DB-Library equivalent 37
 - description 37
 - used with Japanese characters 60, 61
- TDSDATETIME
 - 4-byte and 8-byte supported 37
 - Client-Library equivalent 37
 - DB-Library equivalent 37
 - description 37
- TDSECTACT
 - description 170, 173
- TDSECTBCD
 - description 174, 180
- TDSETLEN
 - description 180, 182
- TDSETLOG
 - description 183, 188
- TDSETPRM
 - description 188, 193
- TDSETPT
 - description 193, 196
- TDSETSOI
 - description 65, 196, 200
- TDSETSPT
 - description 200, 204
- TDSETUDT
 - description 204, 206
- TDSFLT

- 4-byte and 8-byte supported 37
- Client-Library equivalent 37
- COBOL data description 37
- DB-Library equivalent 37
- description 37
- TDSGRAPHIC
 - description 37
 - used with Japanese characters 60, 61
- TDSIMAGE
 - Client-Library equivalent 37
 - COBOL data description 37
 - converting long graphic types to 45
 - DB-Library equivalent 37
 - description 37
 - supported for workstation only 37
- TDSINT2
 - Client-Library equivalent 37, 38, 40
 - COBOL data description 37
 - DB-Library equivalent 37
 - description 37
- TDSINT4
 - COBOL data description 38
 - DB-Library equivalent 38
 - description 38
- TDSLONGVARBIN
 - Client-Library equivalent 38
 - COBOL data description 38
 - description 38
- TDSLONGVARCHAR
 - Client-Library equivalent 38
 - COBOL data description 38
 - description 38
 - differences from TDSVARYCHAR 41
 - used with Japanese characters 60
- TDSMONEY
 - COBOL data description 38
 - DB-Library equivalent 38
 - description 38
 - used with client data only 38
- TDSMONEY4
 - COBOL data description 38
 - DB-Library equivalent 38
 - description 38
 - used with client data only 38
- TDSNDDON
 - description 206, 213
 - use with JCM 213
 - used with long-running transactions 54, 55
 - when to use TDS-ENDREPLY 55
- TDSNDMSG
 - description 213, 220
 - use with JCM 219
- TDSNDROW
 - description 220, 224
 - use with JCM 224
- TDSNUMERIC
 - Client-Library equivalent 39
 - COBOL data description 39
 - description 39
 - used for client data only 39
- TDS-PACKED-DECIMAL
 - Client-Library equivalent 39
 - COBOL data description 39
 - converting to workstation datatype 42
 - description 39
 - used for mainframe data only 39
- TDSQLLEN
 - description 224, 227
- TDS-SYBASE-DECIMAL
 - Client-Library equivalent 39
 - COBOL data description 39
 - description 39
 - used for client data only 39
- TDSTATUS
 - description 227, 232
- TDSTEXT
 - Client-Library equivalent 39
 - COBOL data description 39
 - converting long varchar types to 41
 - DB-Library equivalent 39
 - description 39
- TDSVARGRAPHIC
 - used with Japanese characters 60, 61
- TDSVARYBIN
 - COBOL data description 40
 - DB-Library equivalent 40
 - description 40
 - why not VARBINARY 42
- TDSVARYCHAR
 - COBOL data description 40
 - DB-Library equivalent 40
 - description 40

Index

- differences from TDSLONGVARCHAR 41
- do not use VARCHAR 41
- used with Japanese characters 60, 61
- TDSVARYGRAPHIC
 - use instead of TDSVARGRAPHIC 45
 - used to represent Japanese double-byte characters 40
 - used with mainframe data only 40
- TDSVOID, denoting omitted parameters 40
- TDTERM
 - description 232, 234
 - in mixed-mode application 57
 - required with IMS TM 233
 - used with long-running transactions 56
- TDWRTLOG
 - description 238, 241
- TDYNAMIC
 - description 234, 238
- Trace flag
 - in TDINFLOG 99, 100, 197, 105
- Trace log
 - adding user message 238
 - trace errors only 183
 - trace specific RPCs 183
 - tracing RPCs 183
 - turn off tracing 183
- Tracing
 - functions used in 15
 - global, definition 183
 - querying API 121
 - querying global 99, 100, 105, 121, 197
 - querying specific 100, 105, 197
 - querying TDS data 121
 - querying TDS headers 121
 - querying transaction level 121, 136
 - sample program 385
 - setting API 183
 - setting global 183
 - setting TDS data 183
 - setting TDS headers 183
 - setting transaction level 149, 183, 200
 - types of 124, 186
 - uses error log 124
 - using error log 186
- Transaction
 - long-running, definition 20
 - long-running, language request 98
 - returned by TDGETREQ 98
 - short, definition 20

V

- VARBINARY
 - do not use 42
- VARCHAR
 - do not use 41