SYBASE®

Programmer's Reference for Remote Stored Procedures

Mainframe Connect Server Option

12.6

IBM CICS, IMS, and MVS

DOCUMENT ID: DC35605-01-1260-01

LAST REVISED: March 2005

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

Contents

About This Book

Remote stored procedures (RSPs) are written by customers to access DB2 in the MVS CICS environment. The Mainframe Connect Server Option *Programmer's Reference for Remote Stored Procedures* describes how to design, code, and test RSPs.

This chapter contains the following topics:

- Audience
- How to use this book
- Other sources of information
- Sybase certifications on the Web
- Sybase EBFs and software maintenance
- Conventions
- If you need help

Audience

This guide is for anyone responsible for the following tasks:

- Designing, coding, and testing RSPs in one of the supported programming languages (COBOL II, assembler, PL/I, and C)
- Preparing client applications
- Implementing RSPs
- Administering Open ClientConnect™, Open ServerConnect™, or DirectConnect™ environments
- Administering database management systems
- Supporting data transfer and staging

Product name changes

The following table describes new names for products in the 12.6 release of the Mainframe Connect Integrated Product Set.

Old product names	New product name
• Open ClientConnect for CICS • Open ClientCONNECT for CICS	Mainframe Connect Client Option for CICS

Old product names	New product name
• Open Client Connect for IMS and MVS • Open ClientCONNECT for IMS and MVS	Mainframe Connect Client Option for IMS and MVS
• Open ServerConnect for CICS • Open ServerCONNECT for CICS	Mainframe Connect Server Option for CICS
• Open ServerConnect for IMS and MVS • Open ServerCONNECT for IMS and MVS	Mainframe Connect Server Option for IMS and MVS
• MainframeConnect™ for DB2 UDB • MainframeCONNECT for DB2/MVS-CICS	Mainframe Connect DB2 UDB Option for CICS
• DirectConnect for OS/390 • DirectCONNECT for DB2/MVS	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.

How to use this book

The majority of Sybase customers using COBOL II write RSPs to access DB2 in the MVS CICS environment. This guide therefore provides COBOL II examples. However, the Open ServerConnect API tape provides examples in all the supported programming languages.

If you are not familiar with CICS and the CICS control tables, ask your CICS programmer or system programmer to make the required CICS entries.

This guide provides a set of tasks and reference information, with each chapter representing a task and each appendix representing reference information to help you accomplish a task. This reference guide provides the following information:

Table 1: Contents of each chapter

Chapter	Contents
Chapter 1, “Overview of RSPs”	Provides an overview of RSPs and how they work.
Chapter 2, “Designing an RSP”	Discusses information to consider before you design an RSP.
Chapter 3, “Writing an RSP”	Explains how to write an RSP.
Chapter 4, “Compiling an RSP”	Explains how to compile an RSP.
Chapter 5, “Testing and invoking an RSP”	Explains how to test and invoke an RSP.
Chapter 6, “Troubleshooting”	Explains how to troubleshoot problems.
Appendix A, “RSP Commands”	Lists and explains the RSP commands.
Appendix B, “MODELRSP DB2 Output Pipe Sample RSP”	Provides and explains a sample RSP with DB2-formatted output pipes or multiple-column rows.
Appendix C, “RSP3C STD Input and Output Pipe Sample RSP”	Provides and explains a sample RSP that sends single-column rows of character strings.
Appendix D, “RSP4C Keyword Variable Sample RSP”	Provides and explains a sample RSP that passes keyword values.
Appendix E, “RSP8C Variable Text Sample RSP”	Provides and explains a sample RSP that reads variable text and uses output pipes to echo data that a client application sends to it.
Appendix F, “The SPAREA”	Explains how the SPAREA is used by RSPs. It includes SPAREA fields and SPAREA definitions.
Appendix G, “The SQLDA”	Explains how the SQLDA is used by RSPs.
Glossary	Provides definitions of technical terms used in this book.

Other sources of information

Use the Sybase Getting Started CD, the Sybase Technical Library CD, and the Technical Library 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 Technical Library CD. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader (downloadable at no charge from the Adobe Web site, using a link provided on the CD).

- The Technical Library CD contains product manuals and is included with your software. The DynaText reader (included on the Technical Library CD) allows you to access technical information about your product in an easy-to-use format.

  Refer to the *Technical Library Installation Guide* in your documentation package for instructions on installing and starting the Technical Library.

- The Technical Library Product Manuals Web site is an HTML version of the Technical Library 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 Technical Library 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. Select Products from the navigation bar on the left.
3. Select a product name from the product list and click Go.
4. Select the Certification Report filter, specify a time frame, and click Go.
5. Click a Certification Report title to display the report.

❖ **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. Enter user name and password information, if prompted (for existing Web accounts) or create a new account (a free service).

3 Select a product.

4 Specify a time frame and click Go.

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

Conventions

The following sections describe syntax and style conventions used in this guide.

Note Throughout this book, all references to MVS refer to native MVS programs, and all references to Adaptive Server™ Enterprise also apply to its predecessor, SQL Server®.

Syntax statements that display options for a command look like this:

```
COMMAND [object_name, [ {TRUE | FALSE} ] ]
```

The following table explains the syntax conventions used in this guide.

Table 2: Syntax conventions

Symbol	Convention
( )	Include parentheses 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.

The following style conventions are used in this guide:

- The names of files and directories are shown as:

  econnect\ServerName\CFG

- The names of programs, utilities, procedures, and commands are shown as:

snrfck

- The names of properties are shown as:

  Allocate

- The names of options are shown as:

  connect

- Code examples and text on screen are shown as:

  ```
  this font
  ```

- In a sample command line display, commands you should enter are shown as:

  ```
  this font
  ```

- In a sample command line display, variables (words you should replace with the appropriate value for your system) are shown as:

  ```
  this font
  ```

If you need help

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

CHAPTER 1

Overview of RSPs

This chapter contains the following topics:

- RSP overview
- How RSPs process
- Exchanging information between RSPs and the client
- System requirements
- Migration considerations
- Summary of RSP programming tasks

RSP overview

This overview answers the following questions:

- What is an RSP?
- What does an RSP do?
- How does an RSP access and return DB2 data?

What is an RSP?

An RSP is a CICS command-level program that contains the Sybase RSP calls to the RSP API. The RSP API converts RSP commands to Open ServerConnect commands.

You can write RSPs in any of the four programming languages supported by CICS:

- COBOL II
- assembler
- PL/I

- C (SAS/C or IBM C/370)

What does an RSP do?

An RSP allows a client application to access data and services on the mainframe. Workstation users or client applications on the LAN use RSPs to send requests through DirectConnect for OS/390 (hereafter called DirectConnect), optionally, using MainframeConnect for DB2 UDB (hereafter called MainframeConnect), and directly using TCP/IP.

An RSP uses standard CICS command-level services to perform its processing. It can receive arguments or data sent from the client and generate results to return to the client. You can write an RSP to do one or more of the following:

- Access DB2 data or other relational databases (such as *ADABAS*), statically or dynamically

  For example, an RSP can update all relevant host tables with a changed part number. In this case, the RSP contains multiple UPDATE statements targeted to each table.

- Access non-relational data (such as *VSAM*, *IDMS*, or *IMS*)

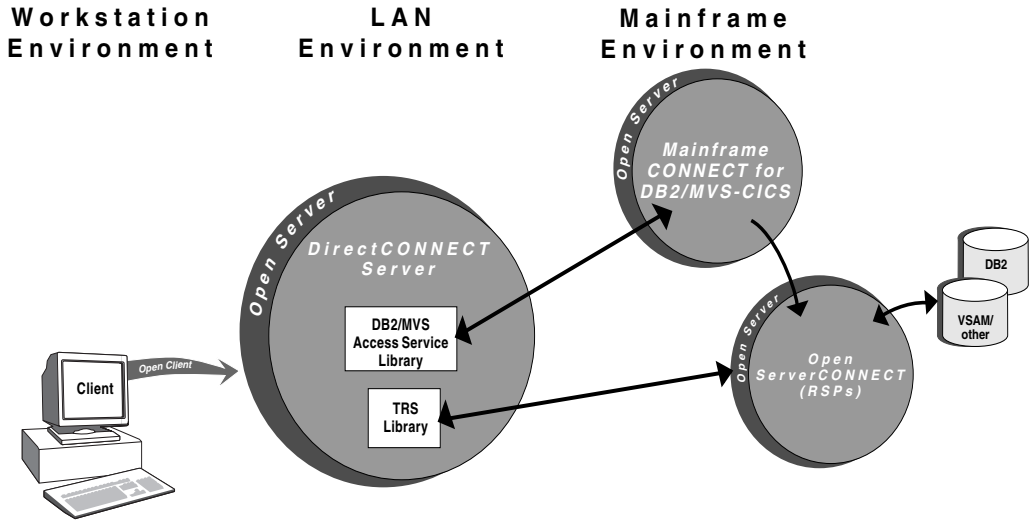
  For example, an RSP could retrieve data from *IMS* and deliver it to the workstation, where the client application converts it into an appropriate format.

- Invoke other CICS programs
- Schedule other CICS tasks for execution
- Issue RSP commands
- Access temporary storage or transient data queues

How does an RSP access and return DB2 data?

This section explains how RSPs access data within the Enterprise Connect structure. The following figure shows how RSPs access and return DB2 data.

Figure 1-1: How RSPs access and return DB2 data

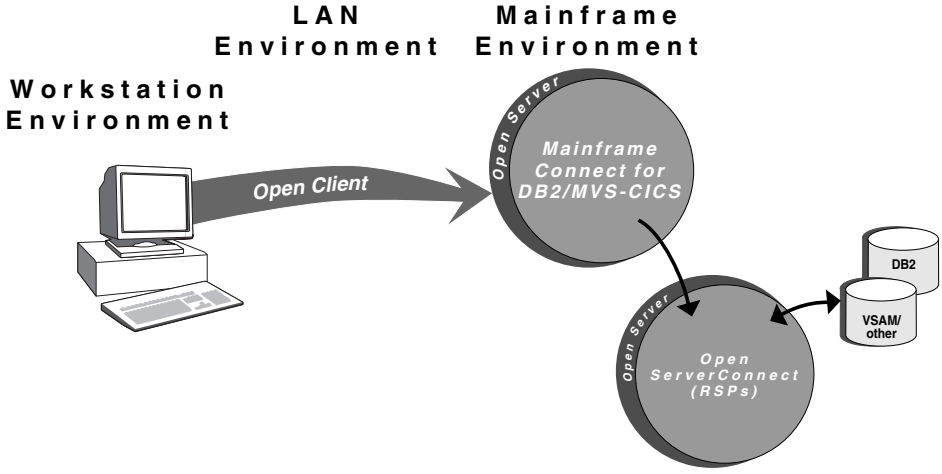


As Figure 1-1 shows, RSPs reside with Open ServerConnect. When one of your client applications invokes an RSP (using Open Client), the request passes to a DirectConnect server. At this point, depending on your configuration, either Transaction Router Service (TRS) Library or the DB2/MVS Access Service Library (hereafter called Access Service Library) invokes the RSP.

TRS accesses DB2 data by directly invoking an RSP through Open ServerConnect. Access Service Library accesses DB2 data by invoking an RSP through MainframeConnect. The software installed on your network determines your application request options and capabilities.

Using TCP/IP for communications allows your client to access the Mainframe environment directly without going through DirectConnect (gateway-less) as indicated in Figure 1-2.

Figure 1-2: Mainframe access without using DirectConnect (gateway-less)



Note You must have Open ServerConnect installed to implement RSPs.

Table 1-1 summarizes the functions available with the possible software configurations.

Table 1-1: Software configuration options

If installed:	You can access:	This software does not support:
DirectConnect and Open ServerConnect	• TRS • RSPs and RPCs through TRS only	• Dynamic SQL access to DB2 • SPTEST utility • The mainframe as a client, either through Open Client or CSAs
DirectConnect, Open ServerConnect, and MainframeConnect	• TRS and Access Service Library • RSPs and RPCs through TRS • RSPs through Access Service Library • Dynamic SQL access to DB2 • SPTEST utility to test RSPs	• The mainframe as a client, either through Open Client or CSAs

How RSPs process

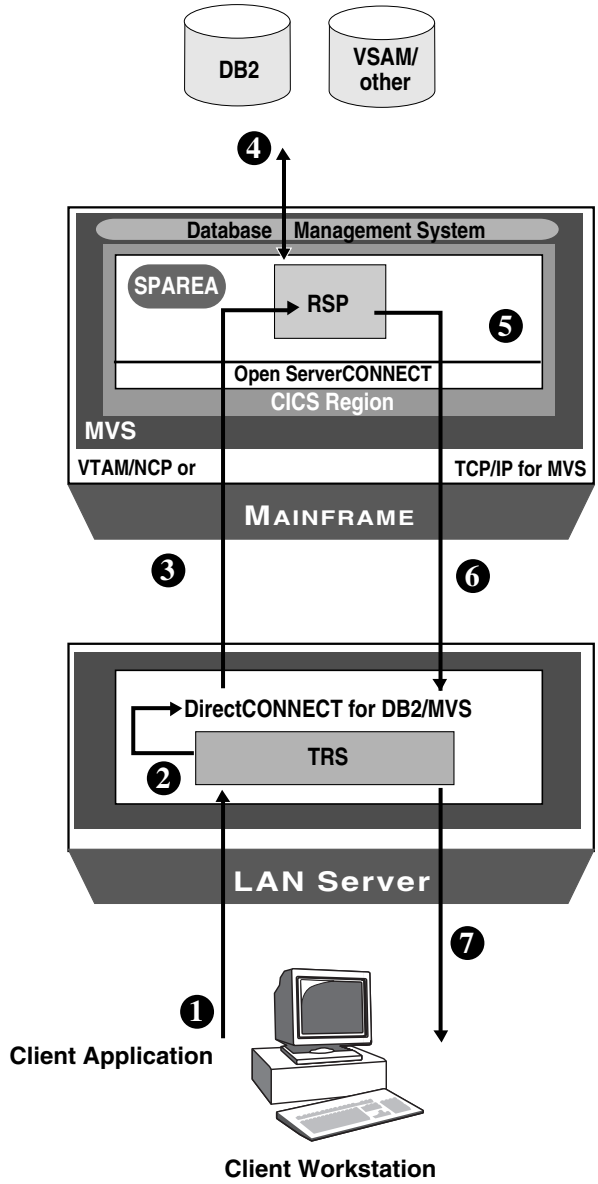
This section explains how RSPs process through TRS and an Access Service Library.

How RSPs are processed through TRS

TRS is a component of DirectConnect. It routes requests from remote clients to Open ServerConnect and returns results to the clients. For more information on TRS, see the Mainframe Connect DirectConnect for z/OS Option *User's Guide for Transaction Router Services*.

The following figure illustrates RSP processing through TRS.

Figure 1-3: RSP processing through TRS



The following explains each step in Figure 1-3:

1 The client application requests a remote procedure call (RPC) with the following command:

```
EXEC rpcname @VARNAME1='value'
```

Note In TRS, you invoke an RSP using the remote procedure call (RPC) name.

2 TRS searches the RPC name for the TP name (transaction program name) and passes the request to DirectConnect. The TP name (which is associated with the RSP program) is invoked in the CICS region.

(The RSP and the Open ServerConnect API use the Stored Procedure Communication Area (SPAREA). For more information on the SPAREA, see "SPAREA" on page 11.

3 DirectConnect invokes the RSP.

4 The RSP performs the desired processing (for example, accessing DB2 data).

5 Open ServerConnect packages the data and messages produced by the RSP.

6 The RSP returns results to TRS.

7 TRS returns the results to the client application.

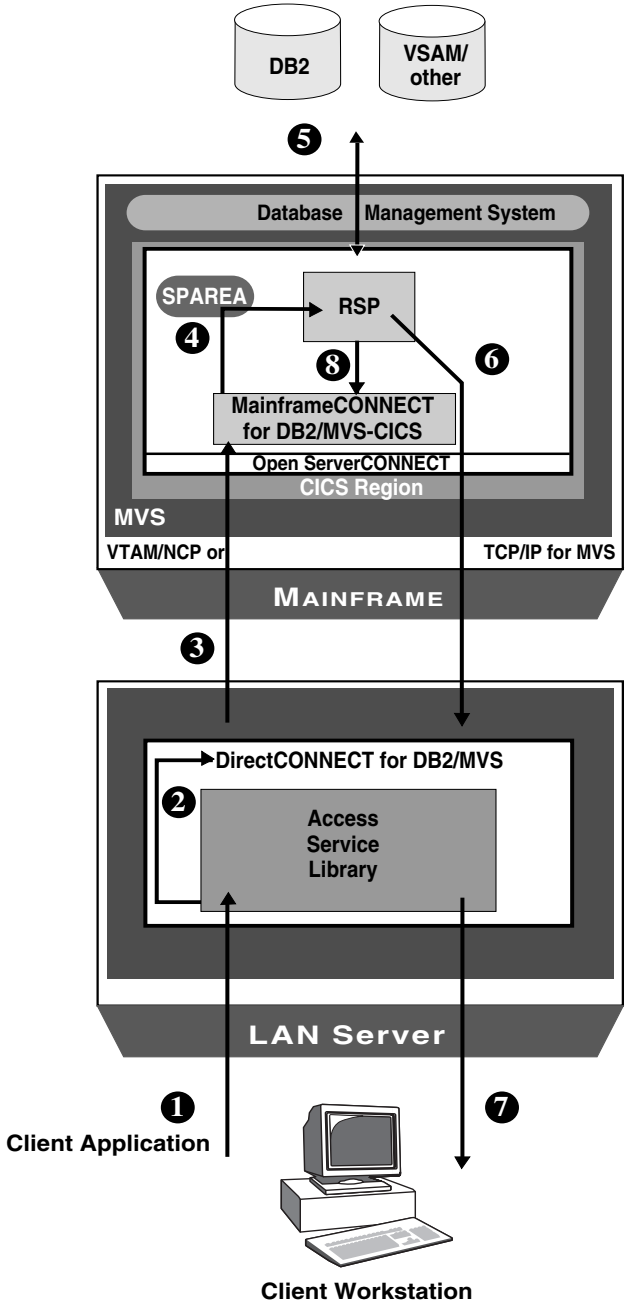
Note The RSP must call RPSETUP and RPDONE.

How RSPs are processed through an Access Service Library

The Access Service Library is the program component of DirectConnect that works with MainframeConnect to provide access to DB2 data. For more information on the Access Service Library, see the Mainframe Connect DirectConnect for z/OS Option *User's Guide for DB2 Access Services* for your database system.

Earlier releases of RSPs used a processing technique similar to the current processing through Access Service Library. The following figure illustrates RSP processing through Access Service Library.

Figure 1-4: RSP processing through Access Service Library



The following explains each step in Figure 1-4:

1 The client application requests a remote procedure call (RPC) using one of the following commands:

```
USE PROCEDURE rspname &VARNAME1=value1
EXECUTE rspname @VARNAME1=value1
```

Note In Access Service Library, you invoke an RSP using the RSP name.

2 Access Service Library passes the request to DirectConnect.

3 DirectConnect passes the command, containing the RSP name and any necessary arguments, to MainframeConnect. The request can contain a number of other statements, any of which can also invoke RSPs.

4 MainframeConnect invokes the RSP through the CICS LINK command. Arguments and other parameters are passed to the RSP using the Stored Procedure Communication Area (SPAREA). For more information on the SPAREA, see “SPAREA” on page 11.

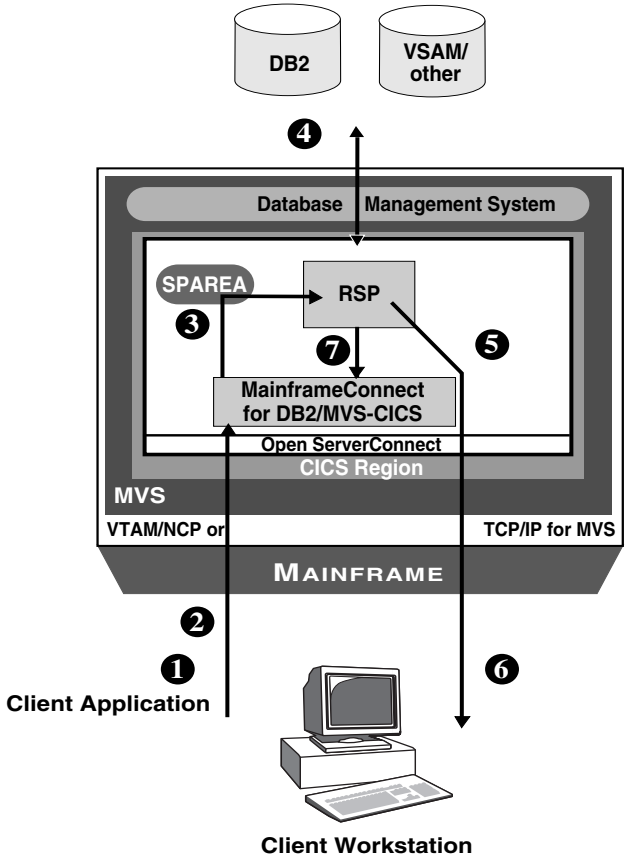
5 The RSP performs the desired processing (for example, accessing DB2 data).

6 Open ServerConnect packages the data and messages produced by the RSP, and sends them to DirectConnect.

7 DirectConnect returns results to the client application.

8 The RSP returns program control to MainframeConnect with a CICS RETURN command

Figure 1-5: Direct RSP processing using TCP/IP



The following explains each step in Figure 1-5:

1 The client application invokes an RSP using the following command:

```
USE PROCEDURE rspname &VARNAME1=value1
```

2 MainframeConnect invokes the RSP through the CICS LINK command.

3 Arguments and other parameters are passed to the RSP using the Stored Procedure Communication Area (SPAREA). For more information on the SPAREA, see "SPAREA" on page 11.

4 The RSP performs the desired processing (for example, accessing DB2 data).

5 Open ServerConnect packages the data and messages produced by the RSP.

6 Open Server sends the data and messages to the Client Workstation.

7 The RSP returns program control to MainframeConnect with a CICS RETURN command.

Exchanging information between RSPs and the client

There are three methods for exchanging information between the RSP and the client application: the SPAREA (keywords or variable text) and the data pipe.

SPAREA

The SPAREA contains all the pointers, codes, and command details that the RSP needs to exchange with the RSP API. Every RSP receives or sends information using the SPAREA.

When an RSP processes through TRS, it creates its own SPAREA through the RPSETUP call. When an RSP processes through Access Service Library, it uses an existing SPAREA on the mainframe to send parameters or data to or from MainframeConnect.

RSP commands (OPENPIPE, PUTPIPE, STATUS, and so on) are small assembler programs that call Open ServerConnect. The RSP commands use the values of fields in the SPAREA as parameters.

Before you issue an RSP command, you first move values to the relevant fields in the SPAREA, then issue a standard system CALL statement. The syntax used for these operations varies with the programming language used. For more information, see Appendix A, "RSP Commands" and Appendix F, "The SPAREA."

Data Pipes

When processing, the RSP uses a data pipe to pass rows of data to or from the client application. The RSP can open a data pipe either to receive or send data. The RSP can only receive data from an input pipe through Access Service Library. Examples of data pipes are provided in “Using data pipes” on page 19.

System requirements

This section lists the system requirements for the:

- Host platform
- DirectConnect platform (optional)

Host platform

The following are system requirements for the host platform:

- Open ServerConnect for CICS must be installed and operational. Detailed system requirements for Open ServerConnect are provided in the Mainframe Connect Server Option *Installation and Administration Guide* (platform-specific).
- MainframeConnect software is optional for RSP use. If your site chooses to use MainframeConnect in RSP processing, the MainframeConnect software must be installed and operational. Detailed system requirements for MainframeConnect are provided in the Mainframe Connect DB2 UDB Option for CICS *Installation and Administration Guide*.
- If the RSP accesses DB2, DB2 packages and plans must be set up for the RSP transaction. If you plan to invoke RSPs with MainframeConnect or through TRS, use plans or packages. See Mainframe Connect DB2 UDB Option for CICS *Installation and Administration Guide* for details on setting up DB2 packages and plans.

DirectConnect platform (optional)

DirectConnect must be installed and operational except when using TCP/IP for communications (gateway-less).

Detailed system requirements for DirectConnect are provided in the Mainframe Connect DirectConnect for z/OS Option *Installation Guide*.

Migration considerations

This section discusses the following migration considerations:

- Necessary coding changes
- Recompiling and relinking existing RSPs
- New data format for RSPs

Coding changes

If you are invoking RSPs through MainframeConnect (using the Access Service Library), there are no changes. If you are invoking RSPs directly through the RSP API (using TRS), you need to make the following coding changes:

- The first API call must be RPSETUP.
- The last API call must be RPDONE.

Recompiling and relinking existing RSPs

If you are migrating from an earlier release of any Sybase product, you must recompile and relink your existing RSPs with the Open ServerConnect RSP stub routines before using those RSPs.

New data format

All data that moves between the RSP, DirectConnect, and MainframeConnect is in Tabular Data Stream™ (TDS) format, which replaces Integrated Exchange Format (IXF). TDS is a Sybase proprietary format, which manages data formatting for you. DirectConnect translates the records it receives into a standard CT-Library format that the client application can handle. DirectConnect no longer converts IXF format input pipes to DB2 format.

Warning! Preformatted IXF data is not converted to DB2-format input pipes any more. Convert your source data to ASCII for DB2-formatted input pipes.

Summary of RSP programming tasks

These are the general steps to build an RSP within a TSO development environment.

1 *Review the design considerations.*

See Chapter 2, “Designing an RSP.”

2 *Prepare a sample RSP* to use as a shell and *write the RSP program*.

See Chapter 3, “Writing an RSP.”

3 *Compile and link-edit* the RSP in the standard manner for CICS command-level programs.

See Chapter 4, “Compiling an RSP.”

4 *Test and invoke the RSP* in the standard manner for CICS command-level programs.

See Chapter 5, “Testing and invoking an RSP.”

If you encounter problems while processing your completed RSP,

See Chapter 6, “Troubleshooting.”

CHAPTER 2

Designing an RSP

This chapter contains the information you must consider when designing an RSP and contains the following topics:

- Using RSP commands
- Reviewing sample RSPs
- Making design decisions
- Considering environmental issues
- Understanding how to invoke an RSP
- Specifying error handling

Using RSP commands

This section is a brief introduction to RSP commands. In addition to reading this introductory material, you should review each command in detail before continuing with the next section, “Reviewing sample RSPs.” See Appendix A, “RSP Commands” for detailed information about each command.

Use the RSP commands to:

- Communicate message and status information to Open ServerConnect and the client application
- Manage COMMITs and ROLLBACKs
- Manage data pipes and exchange data with Open ServerConnect

The following table summarizes the RSP commands and their functions.

Table 2-1: RSP commands and functions

This command:	Performs this function:	See
CLOSPIPE	Closes the data pipe	CLOSPIPE on page 64
COMMIT	Commits a unit of work	COMMIT on page 65

This command:	Performs this function:	See
GETPIPE	Reads a record from the data pipe	GETPIPE on page 65
MESSAGE	Sends a message to the client application	MESSAGE on page 66
OPENPIPE	Opens the data pipe	OPENPIPE on page 67
PUTPIPE	Writes a record to the data pipe	PUTPIPE on page 68
ROLLBACK	Rolls back a unit of work	ROLLBACK on page 69
RPDONE	Ends processing for an RSP initiated using TRS	RPDONE on page 70
RPSETUP	Initializes an RSP	RPSETUP on page 70
STATUS	Indicates the success or failure of processing	STATUS on page 70

Reviewing sample RSPs

Now that you reviewed RSP commands you are ready to review a sample RSP.

Sybase provides sample RSPs for you to use as shells for the RSPs you write. This guide contains four of the sample programs. These samples include explanatory material detailing what the RSP does. Review the sample or samples that fit your RSP needs before continuing with the next section, “Making design decisions.”

- MODELRSP shows you how to use a DB2 format output pipe and a SQLDA definition. See Appendix B, “MODELRSP DB2 Output Pipe Sample RSP” for a reproduction of the sample.
- RSP3C shows you how to use STD format input and output pipes to transmit (send or receive) data. See Appendix C, “RSP3C STD Input and Output Pipe Sample RSP” for a reproduction of the sample.
- RSP4C shows an example of how to transmit keyword variables. See Appendix D, “RSP4C Keyword Variable Sample RSP” for a reproduction of the sample.

- RSP8C shows an example of how to transmit variable text. See Appendix E, “RSP8C Variable Text Sample RSP” for a reproduction of the sample.

Note See Table 3-1 on page 36 for a complete list of the samples provided on the Open ServerConnect API tape.

Making design decisions

Now that you reviewed the RSP commands and a sample RSP, you are ready to make decisions regarding the design of your RSP. Before writing an RSP, you need to make the following design decisions:

- What functions will the RSP perform?
- What functions will the client application perform? Will the client application expect data structure information with results from the RSP?
- Which databases (if any) will the RSP access?
- Will the RSP access temporary storage or transient data queues?
- What type of data (character or binary) will be transmitted?
- Which data pipe format should the RSP use?
- Will the RSP link to other programs or functions?
- What kind of error handling does the RSP require?
- Will the RSP be using input pipes, output pipes, keyword variables, or variable text?

Each of these decisions is discussed in the following subsections.

Note RSPs operate in your environment like any other CICS command-level program. An RSP can access any CICS program or function that you can access with other programs in that environment.

Choosing RSP functions

According to your users' requirements, decide what functions the RSP will perform. For example, your RSP might:

- Access DB2 data, statically or dynamically

  Note With RSPs that contain static SQL, the client application does not need authorization on the DB2 objects accessed by the RSP; authorization to execute the application plan or package of the RSP is all that is required.

- Transfer DB2 data to Adaptive Server Enterprise, or any other supported data source, through DirectConnect
- Access other relational data sources (for example, ADABAS), statically or dynamically
- Access non-relational data (for example, VSAM, IDMS, and IMS)
- Invoke other CICS programs
- Schedule other CICS tasks for execution

Choosing client application functions

You need to understand what functions the client application that calls the RSP is going to perform. Coordinate with the client application programmer to determine the data (that is, keyword variables, variable text, or data) being sent to the RSP and the kind of formatting the client application is capable of performing on the results.

For example, if your RSP provides data structure information with the data it is sending, the client application does less decoding of results. If the RSP sends unformatted data, the client must include more logic to decode the results.

Accessing databases

Your RSP can access any database you have in your CICS environment; for example:

- *DB2*
- *BDAM*

- *IMS*
- *VSAM*
- *ADABAS*
- *IDMS*

For more information on the setup necessary to access DB2 through an RSP, see Chapter 4, "Compiling an RSP."

Using temporary storage/transient data queues

You access temporary storage or transient data queues with RSPs the same way you access them with any other program in CICS. Refer to your CICS documentation for information on accessing temporary storage or transient data queues.

Understanding data transmission formats

You need to determine what type of data to transmit to and from the RSP. The type of data your RSP handles determines, in part, the format of the data pipes you define to send and receive data. For example, if the RSP sends and receives only binary, you define data pipes in the BIN format. For more information on data pipe formats, see Appendix B, "MODELRSP DB2 Output Pipe Sample RSP."

When you send multiple rows of columns, no matter which data pipe you specify, all data transmitted between the RSP and DirectConnect is sent in TDS record format. TRS and DirectConnect translate the TDS records they receive into a standard CT-Library format that the client application can handle. The TDS format is proprietary.

Using data pipes

RSPs use data pipes to receive data from or send results to the client application. There are two types of data pipes: input and output. Use the RSP commands described in Appendix B, "MODELRSP DB2 Output Pipe Sample RSP" to define the type of pipe (input or output) and the format of the data being transmitted. The data pipe management commands are OPENPIPE, GETPIPE, PUTPIPE, and CLOSPIPE.

This section explains input and output data pipes.

Note An input pipe and an output pipe can both be open simultaneously.

Input pipes

You can only use input pipes when an RSP is invoked through the Access Service Library or gateway-less; you cannot use input pipes when an RSP is invoked through TRS. The RSP uses input pipes to read rows of data from the client application.

Note Input pipes must be defined as standard (STD) or binary (BIN) format.

The following code example shows how an RSP uses the SPAREA fields to define an input pipe, then opens, reads from, and closes the input pipe:

```
MOVE 'INPUT' TO SPMODE          — defines an input pipe
 MOVE 'STD' TO SPFORMAT         — defines input pipe as STD format
 MOVE nnnn TO SPMAXLEN          — set maximum size of data record
 SET ADDRESS SPINTO TO dataarea — sets input pointer to record
 CALL 'OPENPIPE' USING SPAREA   — opens the pipe
 CALL 'GETPIPE' USING SPAREA    — reads from the pipe where
  PROCESS INPUT DATA            your code processes data
CALL 'CLOSPIPE' USING SPAREA    — closes the pipe
 CALL ‘MESSAGE’ USING SPAREA    — writes messages
 CALL 'STATUS' USING SPAREA     — sets the return code and returns
                                messages & data
```

A STD or BIN format pipe requires that the SPMAXLEN field provides the maximum size (in bytes) of the data record written to or read from the data pipe.

When defining an input pipe, you need to specify the format of the data to be transmitted through the pipe. An input pipe uses only STD and BIN formats, which do not require data structure information.

STD

(Standard) The simplest type of data pipe to use is the STD format. With a standard data pipe, records are transmitted as a single character string between the client and the RSP. The data is transmitted as variable-length character (VARCHAR) records. Use STD only with input pipes.

BIN

(Binary) With the BIN format, data is transmitted as a binary string. If you transmit records of binary data and you do not want ASCII-EBCDIC or EBCDIC-ASCII conversion done, specify a data pipe in the BIN format. Use BIN only with input pipes.

Note You can transmit any data, including DB2 data, using a STD or BIN data pipe.

For more information about input pipes, see “Using input pipes” on page 29 and “Using concurrent input and output pipes” on page 30.

Output pipes

The RSP uses output pipes to return multiple rows of data to the client application. The following code example shows how an RSP uses the SPAREA fields to define an output pipe, then opens, writes to, and closes the output pipe:

```
MOVE 'OUTPUT' TO SPMODE                  — defines the output pipe
 MOVE 'DB2' TO SPFORMAT                  — defines output pipe as DB2 format
 SET ADDRESS OF SPSQLDA TO SQLDA         — sets a pointer to the SQLDA
 CALL 'OPENPIPE' USING SPAREA            — opens the pipe
 PROGRAM GETS DATA                       where your code processes data
CALL 'PUTPIPE' USING SPAREA              — writes the record
 CALL 'CLOSPIPE' USING SPAREA            — closes the pipe
 CALL ‘MESSAGE’ USING SPAREA             — writes messages
 CALL 'STATUS' USING SPAREA              — sends the return code and returns
                                          messages and data
```

For a DB2 format pipe, the SQLDA describes the location and length of the data columns. However, a STD or BIN format pipe requires that the *SPRECLEN* field contains the length of the data record. It cannot exceed the SPMAXLEN that was specified when the pipe was opened.

An output pipe uses the DB2, STD or Binary format. The DB2 format requires data structure information.

DB2

With the DB2 format, include a SQLDA definition in your RSP when you return data to the client application. You can use these formats to transmit any type of data, not just data from DB2.

The SQLDA is a standard data structure used to define a multi-column result passed to Open ServerConnect. It describes the content of the transmitted data records and, as such, it handles much of the data definition logic that the client application would otherwise have to provide. All files are exchanged between the RSP and MainframeConnect using the SQLDA.

As the RSP programmer, you must define the SQLDA for the data you send to the client and provide a pointer to the SQLDA when you open a data pipe for output. The data structure information passes to Open ServerConnect when the pipe opens. DirectConnect sends this information, in CT-Library format, to the client application.

Note A SQLDA definition is required for all data pipes in DB2 format.

For DB2 output pipes, the RSP must create a SQLDA definition and pass its address to Open ServerConnect through the *SPSQLDA* field in the SPAREA.

For sample COBOL-language and C-language SQLDA declarations for DB2 datatypes and more information about the SQLDA, see Appendix G, “The SQLDA” For an extensive discussion of the SQLDA, see the IBM reference manual for DB2 SQL.

For information about STD and BIN output pipes, see “Using output pipes” on page 30 and “Using concurrent input and output pipes” on page 30.

Linking to other programs

When you link to, or call, another program from an RSP, you must use a command format that allows the program to return to the RSP if you want the called program to share the same pipes. If the program does not return control to the RSP (for example, with an XCTL), CICS makes a copy of the SPAREA for the called program instead of pointing to the original SPAREA, the results of which are unpredictable.

To avoid this, use one of the following commands to link to another program:

CICS LINK

programname

CALL *programname*

Handling errors

You must write your RSP to handle the errors it receives from Open ServerConnect, MainframeConnect, and, optionally, from DB2 or any other database it accesses.

Errors are recorded in the SPRC field of the SPAREA. Your RSP code must check the SPRC field for errors after issuing any RSP command.

See Mainframe Connect Client Option and Server Option *Messages and Codes* for information on Open ServerConnect error messages and actions. See Mainframe Connect DB2 UDB Option for CICS *Installation and Administration Guide* for information on MainframeConnect error messages and actions. Also see Chapter 6, “Troubleshooting” for more information on MainframeConnect errors.

Considering environmental issues

This section discusses the environmental issues you should consider when you design an RSP. Specifically, it discusses how data is transferred to Adaptive Server Enterprise and how DirectConnect configuration property settings affect RSP processing.

How data is transferred to Adaptive Server Enterprise

You can write an RSP to transfer data, as part of a TRANSFER function, from a data source other than DB2 (for example, VSAM) to Adaptive Server Enterprise (or another database). However, the RSP must define a SQLDA for the data so that it is formatted like DB2, and it must use a data pipe in DB2 format to send the data to Adaptive Server Enterprise.

How configuration property settings affect RSP processing

This section describes the DirectConnect and MainframeConnect configuration property settings that affect how an RSP processes.

Access service library

If client applications invoke an RSP through the Access Service Library, you need to be aware of how some of the DirectConnect configuration properties affect both client application and RSP processing. This section explains the following information:

- Datatype conversion
- Preventing inconsistencies in SQL transformation
- Managing COMMIT/ROLLBACK

Datatype conversion

Adaptive Server Enterprise applications are designed to manipulate data in Adaptive Server Enterprise datatypes. When these applications execute an RSP to retrieve host data, DirectConnect converts the result rows into the corresponding Adaptive Server Enterprise datatypes.

Preventing inconsistencies in SQL transformation

Adaptive Server Enterprise uses the Transact-SQL™ query language, while DB2 uses IBM's version of SQL. Consequently, SQL statements written for Adaptive Server Enterprise generally do not perform as expected when executed against DB2. To prevent SQL inconsistencies, each DirectConnect Access Service is configured either for native SQL or for Transact-SQL transformation.

Note DirectConnect Access Service is a specific set of configuration properties working with the Access Service Library. The Access Service Library is the program component that works with MainframeConnect to provide access to DB2 data.

The corresponding DirectConnect Access Service transformation modes are PASSTHROUGH for native DB2 SQL and SYBASE for Transact-SQL.

Note TSQL transformation modes (TSQL0, TSQL1 and TSQL2) are supported to provide backward compatibility.

If you write a client application to invoke an RSP, you must be aware of how the SQL transformation level is configured for the Access Service because it determines the format of the RSP invocation command you use. See Figure 5-3 on page 50 for more information.

Note TRS always uses PASSTHROUGH.

Managing COMMIT /ROLLBACK

When you write an RSP, be aware of how DirectConnect configuration property settings affect COMMIT/ROLLBACK management under normal and error conditions. The following table shows the interaction of the configuration property settings under *normal* processing conditions.

Table 2-2: Configuration properties and COMMIT/ROLLBACK

Transaction mode DirectConnect configuration property setting	Outcome
SHORT	MainframeConnect issues COMMIT/ROLLBACK after each batch
LONG	Client application or RSP issues COMMIT/ROLLBACK

Therefore, if TRS invokes an RSP, the transaction is committed (unless the transaction failed) because TRS always runs in SHORT.

The client application uses standard SQL statements to issue COMMITs and ROLLBACKs; the RSP uses the special RSP COMMIT and ROLLBACK commands.

If the RSP invokes through Access Service Library, COMMIT and ROLLBACK processing under *error* conditions is also affected by the DirectConnect Stop Condition configuration property.

This property can be set as follows:

- None—If an error occurs, the RSP continues processing despite error status messages.
- Error—If an error occurs, the RSP receives a STATUS message from MainframeConnect and RSP processing stops.
- Err/Warn—If either an error or a DB2 warning message occurs, RSP processing stops (for Database Gateway release 2.03 only).

Note The client application can override the DirectConnect StopCondition configuration property with the following set statement: set StopCondition {error|none|warning}.

MainframeConnect

If your site uses exits, review the MainframeConnect Request Exit and Parse Exit user configuration properties in the Mainframe Connect DB2 UDB Option for CICS *Installation and Administration Guide*. If either of the exits transform requests, you need to be aware of that transformation.

Understanding how to invoke an RSP

The client can invoke an RSP with two kinds of variables: keyword variables or variable text. The client can also send data to the RSP using a STD input pipe. How the RSP is invoked affects how you design it. Refer to “Output pipes” on page 21.

Invoking with keyword variables and variable text

If your RSP passes keyword variables or variable text, your code accesses the following fields in the SPAREA:

Table 2-3: SPAREA variable fields

SPAREA Field	Use
SPVARTXT	Specifies the address of the variable text that the client application sent to the RSP
SPVARLEN	Specifies the length of the variable text the client application sent to the RSP
SPVARTAB	Specifies the address of the variable substitution table keyword variables that the client application sent to the RSP

See Appendix F, “The SPAREA” for more information.

Processing with keyword variables

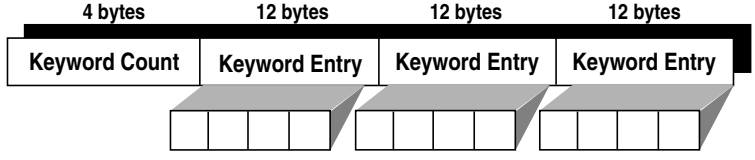
If the client application is sending keyword variables, MainframeConnect (with the Access Service Library) or Open ServerConnect (with TRS):

- Parses the arguments
- Builds a table of keywords and associated values (the keyword variable substitution table)
- Places the address of this table in *SPVARTAB*

If the arguments are not in keyword format, MainframeConnect or Open ServerConnect sets the SPVARTAB to '0'.

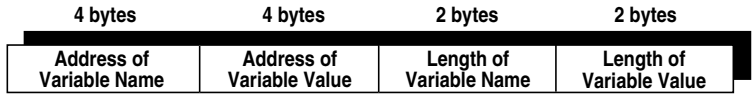
The keyword variable substitution table contains a full word count of the number of keywords that were specified, followed by one keyword entry for each keyword specified. The following figure illustrates the variable substitution table.

Figure 2-1: Keyword variable substitution table



The following figure illustrates the keyword entry format.

Figure 2-2: Keyword entry format



The fields in the keyword entry are in integer format; addresses are 4 bytes long and lengths are 2 bytes long. For example, if the client application passed the following single variable:

```
&DATE=1991-12-04
```

the variable substitution table built by MainframeConnect or Open ServerConnect might appear as follows:

```
10000253D000254F 5      10
```

- where *1* is the keyword count indicating the number of keyword entries; in this case, the &DATE is the only keyword.
- where *0000253D* is the address of the variable name in the SPAREA.
- where *0000254F* is the address of the variable value in the SPAREA.
- where *5* is the length of the variable name; in this case, *&DATE*.
- where *10* is the length of the variable value; in this case, *1991-12-04*.

See Appendix D, “RSP4C Keyword Variable Sample RSP” and Appendix E, “RSP8C Variable Text Sample RSP” for sample RSPs that handle variables.

Processing with variable text

If the client application sends variable text, Open ServerConnect (if TRS is used) or MainframeConnect (if Access Service Library is used) places:

- The address of the variable text in SPVARTXT
- The length of the variable text in SPVARLEN

If the client application does not pass any arguments, Open ServerConnect sets SPVARTXT and SPVARLEN to 0.

See Chapter 5, “Testing and invoking an RSP” for details on sending variables and data from the client application.

Invoking with data pipes

The data pipe is the mechanism by which an RSP sends results to or receives data records from the client application. Both an input pipe and an output pipe can be open at the same time.

You can use a combination of different data pipe formats for input and output. For example, you can define input pipes as STD format and output pipes as DB2 format.

This section describes what you need to consider when using input and output pipes with fixed- and variable-length records and binary data.

Transmitting fixed-length or variable-length records

STD and BIN format pipes can transmit either fixed- or variable-length records. They are the only data pipe formats that use the SPAREA SPMAXLEN and SPRECLEN properties. SPMAXLEN sets the maximum length for data records to be passed through a data pipe; SPRECLEN specifies the actual length of a particular data record.

Using input pipes

When you define an input pipe to handle fixed-length records, you set SPMAXLEN. The RSP needs to read SPMAXLEN only once. SPRECLEN is not required and is set by MainframeConnect.

For every record sent through an input pipe, MainframeConnect places the record length in SPRECLEN, overwriting the existing SPRECLEN value. You must check this value (record length) for each record after every GETPIPE.

The following table explains how to set input pipes for fixed- or variable-length records.

Table 2-4: Setting input pipes

Fixed-length data	Set SPMAXLEN on the OPENPIPE command to the length of a single data record.
Variable-length data	Set SPMAXLEN; then after each GETPIPE, check SPRECLEN and process the incoming record accordingly. Check SPRECLEN only if it is possible that the client application passes variable-length records.

Using output pipes

For every record sent through an output pipe—that is, before every PUTPIPE—the RSP must place the record length in SPRECLEN. The following table explains how to set output pipes for fixed- or variable-length records.

Table 2-5: Setting output pipes

Fixed-length data	Set SPMAXLEN with the OPENPIPE command.
Variable-length data	Set SPMAXLEN with the OPENPIPE command, then set SPRECLEN with every PUTPIPE.

Using concurrent input and output pipes

If both an input pipe and an output pipe are open simultaneously, the RSP needs to know whether the value in SPMAXLEN reflects the input or output pipe. In addition, depending on whether the data is fixed- or variable-length, the RSP may need to reset or restore and reread the SPRECLEN value for every output data record. The following table summarizes how you set fixed- and variable-length data for concurrent input and output pipes.

Table 2-6: Setting concurrent input and output pipes

Input and output pipes both fixed-length data	If both data records are the same length: 1 Set SPMAXLEN with each OPENPIPE command. 2 Check SPRECLEN only if it is possible that the client application passes variable-length records. If this occurs, reset the SPRECLEN value for subsequent PUTPIPE commands. If the data records are different lengths: 1 Set SPMAXLEN with each OPENPIPE command. Then set SPRECLEN with each PUTPIPE command. 2 Check SPRECLEN only if it is possible that the client application passes variable-length records. If this occurs, check the SPRECLEN value for that GETPIPE command, then restore it for subsequent PUTPIPE or GETPIPE commands.
Input and output pipes both variable-length data	1 Set SPMAXLEN with each OPENPIPE command. 2 Check SPRECLEN before each GETPIPE and place the value in the GETPIPE command. 3 Reset SPRECLEN with each PUTPIPE.
Input pipe fixed-length; Output pipe variable- length	Handle as if they were both fixed-length, and of the length set in the output pipe SPMAXLEN.
Input pipe variable- length; Output pipe fixed-length	Handle as if they were both variable-length.

Transmitting binary data

When an RSP uses a DB2 format data pipe, EBCDIC-ASCII or ASCII-EBCDIC conversion does not occur for the columns defined as binary. When you use DB2 format data, each binary column is indicated by setting the corresponding SQLDATA field to X'0000FFFF' at OPENPIPE. You can define only CHAR, VARCHAR, and LVARCHAR columns as binary.

The RSP must set the SQLDATA field appropriately. To indicate whether a column contains binary or normal data, you place the appropriate value in the corresponding SQLDATA field before issuing the OPENPIPE command:

```
X'xxxxxxxx' (for normal data)
 X'0000FFFF' (for binary data)
```

where:

- *xxxxxxxx* is a pointer to the actual data.
- 0000FFFF is the DRDA/DB2 V2R3 “for bit data” indicator.

If any columns were defined as binary, the corresponding SQLDATA fields must be reset to point to the actual column data after the OPENPIPE is issued.

See Appendix G, “The SQLDA” for more information on the SQLDA.

Specifying error handling

When Open ServerConnect executes a command, it uses the SPAREA SPRC field to send a return code that indicates the success or failure of the command.

- If the command succeeds, the SPRC field is set to '000'.
- If an error occurs:
  a The SPRC field is set to a 3-character Open ServerConnect error code. Mainframe Connect Client Option and Server Option *Messages and Codes* contains the Open ServerConnect error codes related to RSPs.
  b Open ServerConnect issues a STATUS command.
  c The RSP is not allowed to issue any more commands. The RSP should perform any termination processing and then return control to Open ServerConnect.

The following COBOL II statements show an example of return code checking after issuing an OPENPIPE command:

```
CALL 'OPENPIPE' USING SPAREA
 IF SPRC NOT EQUAL '000' THEN GOTO PERFORM-TERMINATE.
```

In addition to '000', the SPRC field can contain other codes. For example: ’EOF’, ‘ACE’, and ‘CAN’. See the following table for an explanation of those codes and the SPAREA fields used to communicate status and messages between Open ServerConnect and the RSP.

Table 2-7: SPAREA error handling fields

SPAREA Field	Use
SPRC	RSP API indicates the success or failure of an RSP command in this field. Possible values are: • '000' indicates successful completion. • *'xxx'* indicates a Open ServerConnect error message. • 'EOF' indicates an End of File on input data. • 'ACE' indicates an APPC communication error (when the MainframeConnect configuration property Temporary Storage Type is set to None). • 'CAN' indicates the client issued a DBCANCEL command.
SPSTATUS	RSP API communicates the status of processing in the remote database to the RSP. The RSP also uses the SPSTATUS field to communicate status on its own processing to the client application. Possible values are: • 'OK' indicates success. • 'E' indicates an error. • 'W' indicates a warning.
SPMSG	RSP communicates messages back to the client using this field.
SPCODE	An error code that is sent in a message to the client application appears in this field.

For a complete list of MainframeConnect error messages, see Mainframe Connect Client Option and Server Option *Messages and Codes*.

CHAPTER 3

Writing an RSP

This chapter provides information to help you write an RSP and covers the following topics:

- Overview
- Choosing a sample RSP
- Renaming the sample
- Testing the sample
- Writing the RSP

Overview

Sybase provides sample RSPs for you to use as shells for the RSPs you write. When you write an RSP, select a sample, rename and test the sample, and then alter it to fit your needs.

Choosing a sample RSP

Sybase recommends that you select a sample RSP in the programming language you are using as a shell for your application. The sample RSPs are provided on the Open ServerConnect API tape.

The following table lists the sample programs and definitions available to you:

Table 3-1: Samples on the Open ServerConnect API tape

Sample	*Description*
MODELRSP	Shows how to use a DB2 format output pipe and a SQLDA definition. MODELRSP is reproduced in Appendix B, "MODELRSP DB2 Output Pipe Sample RSP."
RSP3C	Shows how to use STD format input and output pipes to transmit data. RSP3C is reproduced in Appendix C, "RSP3C STD Input and Output Pipe Sample RSP."
RSP4C	Shows an example of transmitting keyword variables. RSP4C is reproduced in Appendix D, "RSP4C Keyword Variable Sample RSP."
RSP8C	Shows an example of transmitting variable text. RSP8C is reproduced in Appendix E, "RSP8C Variable Text Sample RSP."
SAMP01A	Assembler sample program RSP 1. Shows how to use a text property to select data in DB2 and write the results to a CICS temporary storage queue.
SAMP01C	COBOL II sample program RSP 1. (See SAMP01A for description of what it does.)
SAMP02A	Assembler sample program RSP 2. Shows how to select the contents of an entire DB2 table and write the results to STD-format output pipes.
SAMP02C	COBOL II sample program RSP 2. (See SAMP02A for description of what it does.)
SAMP03A	Assembler sample program RSP 3. Shows how to use a keyword property to select data from DB2 and write the results to DB2-format output pipes.
SAMP03C	COBOL II sample program RSP 3. (See SAMP03A for description of what it does.)
SAMP04A	Assembler sample program RSP 4, which demonstrates VSAM access. Shows how to use a text property as a partial key to perform a partial-key "browse" on a VSAM *KSDS* dataset and write the results to DB2-format output pipes.
SAMP04C	COBOL II sample program RSP 4. (See SAMP04A for description of what it does.)
EMPDATA	Test data for sample program SAMP04.
EMPFILE	VSAM define for sample program SAMP04.
EMPREPRO	JCL to populate sample VSAM file.
EMPTAB	Create table for sample SAMP04.
SPAREAP	PL/I RSP communication area.
SPAREAX	C RSP communication area.
SQLDAX	C sample SQLDA.

PARTSTAB	Create SQL statement table for sample RSPs.

Renaming the sample

After selecting a sample RSP to use as a shell, rename the sample using the naming conventions of standard mainframe programs at your site for the RSP name.

Testing the sample

Before you begin to write your RSP, test the sample you are using as a shell. The samples use a table called *PCSQL.SAMPLE_PARTS*. The CREATE TABLE statement for this table is member *PARTSTAB* in the *SYBASE.ORSP310B.CICS.SOURCE* library.

If you want to compile these examples and test them, Sample 1 (SAMP01A or SAMP01C) requires you to provide a 5-byte character value for *PARTNO*. This variable is not in keyword format, so the statement that executes this stored procedure would appear as:

```
USE PROCEDURE SAMP01x 'xxxxx'
```

Sample 3 (SAMP03A or SAMP03C) requires you to provide an ISO-format (yyyy-mm-dd) date value in keyword format for *&DATE*, as follows:

```
USE PROCEDURE SAMP03x &DATE='yyyy-mm-dd'
```

If you need detailed instructions on testing the sample, go to Chapter 5, “Testing and invoking an RSP.”

Writing the RSP

By now you should have:

- Reviewed the RSP commands
- Reviewed one of the four sample RSPs provided in the appendixes
- Reviewed Chapter 2, “Designing an RSP”
- Gathered requirements for and designed your RSP, determining:

- The processing to be done by both the client application and the RSP
- The type of data (character or binary) to transmit
- The types of data pipes (input or output) to use
- The format of data to transmit through those data pipes (STD or DB2)
- Whether you need to use a SQLDA definition (if you are using DB2 format)

You may find it helpful to use existing data definitions or data access code from other programs. Some of the programming tasks involved in writing RSPs are as follows:

- Defining input and output data pipes
- Using the provided RSP commands, such as MESSAGE and STATUS, whenever appropriate (see Appendix A, “RSP Commands” for details)
- Accessing the SPAREA, which the RSP shares with MainframeConnect
- Specifying keyword and variable handling
- Specifying error handling

CHAPTER 4

Compiling an RSP

This chapter discusses the following topics:

- Overview
- Compiling an RSP without DB2
- Compiling an RSP with DB2
- Understanding the linkage

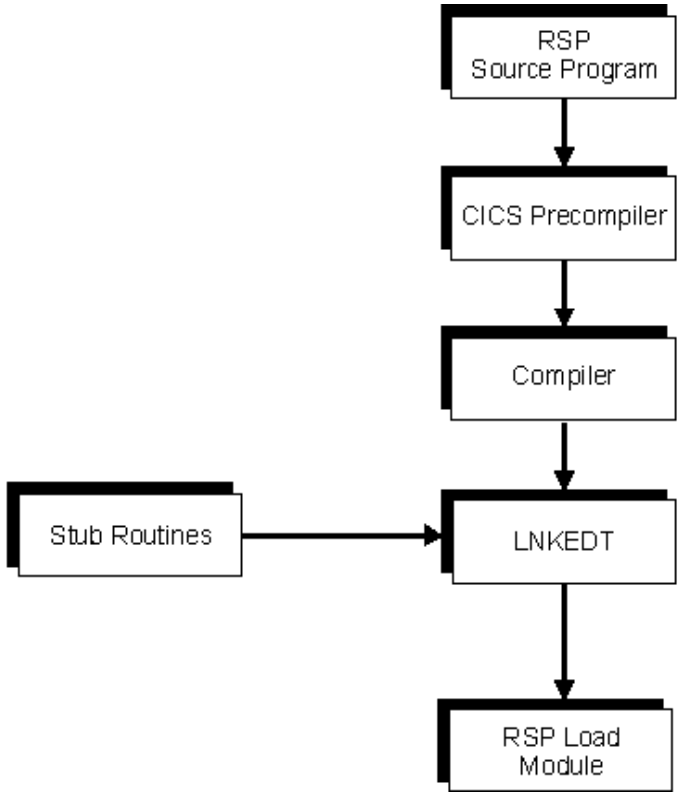
Overview

This chapter explains how to compile an RSP with and without DB2 and includes an explanation of linking.

Compiling an RSP without DB2

Compile and link-edit the RSP in the standard manner for CICS command-level programs. Use the following figure as a guide when performing steps to compile an RSP without DB2.

Figure 4-1: Compiling an RSP without DB2



As Figure 4-1 shows, you perform the following tasks to compile an RSP without DB2:

1 Run the RSP source program through the CICS precompiler.

2 Compile the RSP source program.

3 Link-edit the RSP source program with the stub routines.

The RSP load module is created.

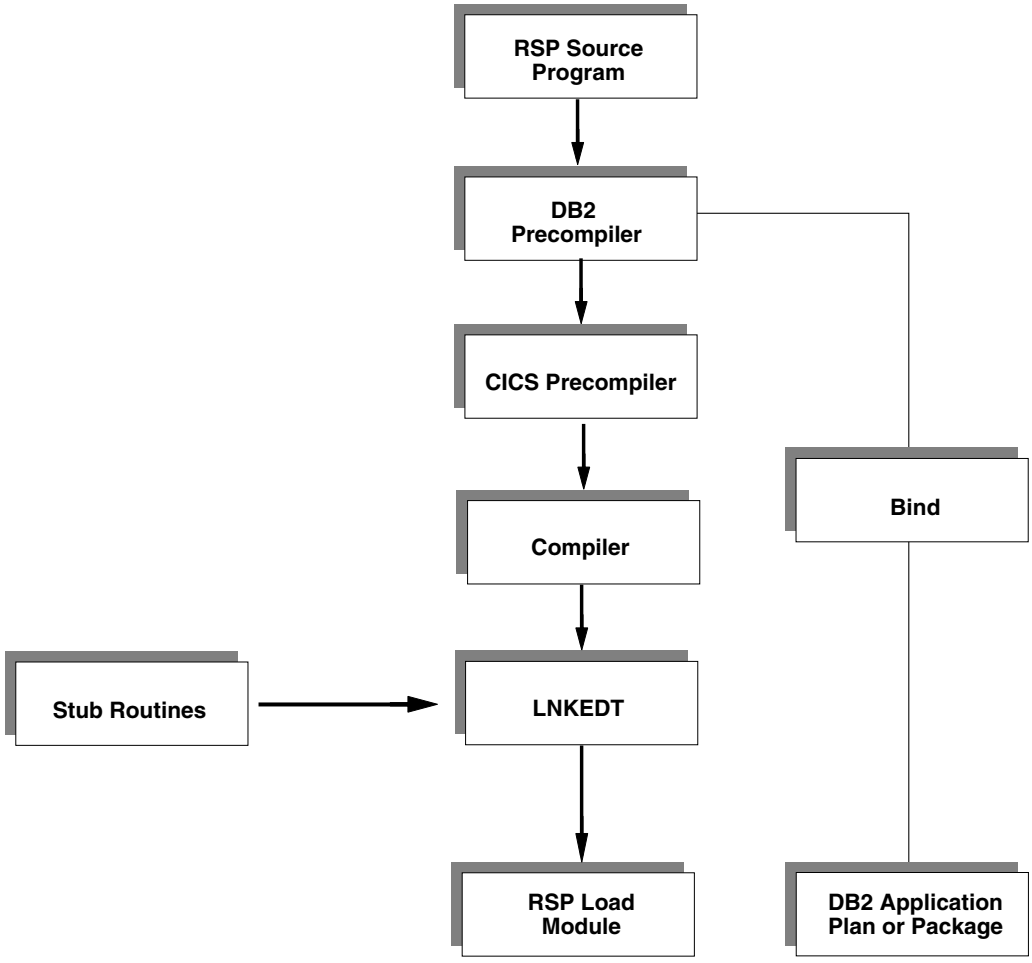
For more information on linking, see “Understanding the linkage” on page 44.

Compiling an RSP with DB2

Compile and link-edit the RSP in the standard manner for CICS command-level programs. If the RSP accesses DB2, be sure the RSP is processed by the DB2 precompiler program before running it through the CICS precompiler. In addition, you need to bind the resulting application plan. Be sure that your systems administrator grants users EXECUTE authority on the RSP plan and package. See Mainframe Connect DB2 UDB Option for CICS *Installation and Administration Guide* for details.

Use the following figure as a guide when performing steps to compile an RSP with DB2.

Figure 4-2: Compiling an RSP with DB2



As Figure 4-2 shows, you perform the following tasks to compile an RSP with DB2:

1 Run the RSP source program through the DB2 precompiler.

2 Run the RSP source program through the CICS precompiler.

3 Compile and link-edit the RSP source program with the stub routines.

The RSP load module is created.

4 Bind the database request module (DBRM) created in the DB2 precompile process to DB2 as a plan or package.

For more information on linking, see “Understanding the linkage” on page 44.

Using DB2 plans (TRS Only)

You can have a separate plan for each RSP. If you do, you need an entry in the CICS RCT table for each RSP transaction that points to each RSP plan name.

Using DB2 packages (TRS or MainframeConnect) or gateway-less

DB2 packages allow you to use one plan for all of the RSPs that access DB2, provided that MainframeConnect, if installed, and all the RSP DBRMs are bound in packages included in that plan. After creating the DB2 collection and plan, you can bind RSP packages in the collection instead of rebinding the plan. This eliminates the need for dynamic plan allocation when MainframeConnect is installed. All the RSP entries in the CICS RCT table can point to the same plan name.

If you are using DB2 packages, ask your DB2 systems administrator for the reference guide for DB2 commands and utilities for information on preparing to use DB2 packages.

Creating a DB2 package

To create a DB2 package, follow these steps:

1 Create the collection using the following command:

```
GRANT CREATE ON COLLECTION SYAMD2 TO PUBLIC
```

2 Bind the plan to include the collection and grant access to the packages using the following command:

```
BIND PLAN(AMD2PLAN) ACTION(REPLACE) PKLIST(*.SYAMD2.*) +
ISOLATION(CS) VALIDATE(BIND)
GRANT RUN ON PLAN
AMD2PLAN TO PUBLIC
```

3 Bind the packages in the collection using the following command:

```
BIND PACKAGE(SYAMD2) ACT(REPLACE) +
LIBRARY('SYBASE.AMD2105.CICSDB2.DBRM') MEMBER(RSPA) +
ISOLATION(CS) VALIDATE(BIND)
GRANT EXECUTE ON PACKAGE SYAMD2.RSPA TO PUBLIC
BIND PACKAGE(SYAMD2) ACT(REPLACE) +
LIBRARY('SYBASE.AMD2105.CICSDB2.DBRM') MEMBER(RSPB) +
ISOLATION(CS) VALIDATE(BIND)
GRANT EXECUTE ON PACKAGE SYAMD2.RSPB TO PUBLIC
```

Understanding the linkage

During the link-edit step, stub routines are included in the resulting load module for the RSP. The stub routines provide the linkage between the RSP and Open ServerConnect.

Note Each time you link-edit, you must also perform a CICS NEWCOPY.

Linking RSPs

MVS requires that RSPs be linked above the 16MB line in 31-bit addressing mode. To do this, add a line to the RSP source program similar to the following JCL:

```
//LNKEDT  EXEC PGM=IEWL,PARM='parms AMODE(31)
          RMODE(ANY)'
```

The concatenation sequence for SYSLIB in the link edit step must include a DD statement for the stub library, either in load format or object format.

Linking load modules

When you link load modules, add a line similar to the following to the SYSLIB DD concatenation in the JCL:

```
//SYSLIB  DD DSN=SYBASE.ORSP310B.CICS.LOADLIB,
             DISP=SHR
```

Linking object code

When you link object code, add a line similar to the following to the SYSLIB DD concatenation in the JCL:

```
//SYSLIB  DD DSN=SYBASE.ORSP310B.CICS.OBJLIB,
              DISP=SHR
```

The *SYBASE.ORSP310B.CICS.xxxxx* value varies with the Open ServerConnect version you are using. See the Mainframe Connect Server Option *Installation and Administration Guide* (platform-specific) for more information.

Note If you are using COBOL II, CICS requires that you link-edit the stub routine DFHECI at the top of the RSP.

CHAPTER 5

Testing and invoking an RSP

This chapter discusses the following topics:

- Overview
- Before you test or invoke an RSP
- Testing an RSP using an ASPT transaction
- Running the RSP test program

Overview

For installations that include MainframeConnect, the Transaction ASPT (RSP Test Screen) utility allows you to view the first 15 rows of results from the RSP. In addition, you can test the RSP fully by invoking it. This chapter explains how to do both.

Before you test or invoke an RSP

Each RSP must have a CICS PPT entry. (Generally, the systems administrator or system programmer makes CICS entries.)

In addition, if the RSP runs through TRS and accesses DB2, a transaction definition in CICS is required for each RSP and an RCT entry is required for that transaction.

Testing an RSP using an ASPT transaction

The ASPT Transaction allows you to test RSPs using STD input pipe data (keyword, variable text). Although you can write RSPs to use BIN input pipes, for testing with ASPT, you must use STD format.

Note Test the RSP in the standard manner for CICS command-level programs.

Testing an RSP involves creating a temporary storage queue and running ASPT.

Creating a temporary storage queue

To provide input pipe data to RSP Testor, create a temporary storage queue and populate it with data of the same type and format that will be sent to the RSP in normal use. You must name the temporary storage queue with the same name as the RSP being tested.

Note Because the RSP Testor screen is case sensitive, you must enter the RSP name in capital letters so the temporary storage queue that holds your input records can be located. If you receive an `EOF ALREADY ENCOUNTERED` message, be sure you entered the RSP name correctly.

Use program function keys to work with the results. The following table describes the program function key operations.

Table 5-1: Function key operations

This key:	Performs this function:
F3	Terminates the RSP test
F5	Displays the arguments that were specified for the RSP test. You can specify new arguments if you want.
F6	Displays the messages or data produced by the RSP

The CICS CECI transaction is a convenient tool for creating and populating the temporary storage queue with STD-format data. The following example uses the CECI command to create and load a temporary storage queue for input records:

```
CECI WRITEQ TS QUEUE('RSPNAME') FROM('THIS IS A DATA RECORD')A
```

Running the RSP test program

To test an RSP using the RSP test program, perform the following steps:

1 Sign on to CICS and enter the command for RSP Test program:

```
ASPT
```

The Stored Procedures Test window appears as shown in the following figure:

Figure 5-1: Stored Procedure Test window

```
S T O R E D   P R O C E D U R E   T E S T
Stored Procedure Name ===>

Specify Variables Below:
```

2 At the following prompt,

```
Stored Procedure Name
```

specify the name of the RSP you are testing. If the RSP expects variables, specify the values in the format the RSP expects.

The completed information in the Stored Procedure Test window is shown in the following figure.

Figure 5-2: Completed Stored Procedure Test window

```
S T O R E D   P R O C E D U R E   T E S T
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□-
Stored Procedure Name ===>> SAMP02C
Specify Variables Below:

&PARTNO=100 &COLOR='BLUE'
```

3 Press Enter to perform the test.

When the RSP completes processing, the results from the test appear on the screen. If the RSP produced any output (messages or data), the first 15 lines of the output also appear.

The following figure shows the test results for the sample program SAMP02C RSP. The output consists of four data records and messages.

Figure 5-3: Stored Procedure Test results window

```
S T O R E D   P R O C E D U R E   T E S T
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
TEST COMPLETE  STATUS: OK  ERRCODE:    ROW COUNT: 4
Stored Procedure Name ===>> SAMP02C
Data Records Shown Below:
0003800300PART NUMBER 300               Z15
0003800300PART NUMBER 300               Z15
0003800200PART NUMBER 200               A15
0003800100PART NUMBER 100               A14
```

Invoking an RSP

Both the client application programmer and the RSP programmer need to be aware of how client applications interact with RSPs. This section describes how to invoke RSPs, how to migrate from previous modes, and how to send data to the RSP.

How the RSP will be invoked (through Access Service Library, TRS, or both) or Gatewayless determines the command you use to invoke it. When a client application invokes an RSP, arguments are passed to the RSP on the USE PROCEDURE, EXECUTE, or EXEC statement. The RSP accesses these values through the SPAREA. When you write a client application to invoke an RSP, the format of the invocation command you use depends on:

- The SQL transformation (TSQL) configuration property setting on the DirectConnect Access Service

  If you write a client application to invoke an RSP, ask your LAN administrator how the DirectConnect Access Service TSQL configuration property is set at your site.

  - A setting of PASSTHROUGH mode allows you to issue statements in the target's SQL dialect.
  - A setting of SYBASE mode transforms most syntax of the received SQL text into the SQL syntax that is supported by the target DBMS.

- The type of data (if any) you send with the RSP invocation request

  The data you transmit can be in binary format or ASCII text.

Note DirectConnect and Open ServerConnect support MDI Database Gateway™ TSQL modes of TSQL0, TSQL1, and TSQL2 for backward compatibility only. TSQL0 corresponds to PASSTHROUGH mode, and TSQL2 corresponds to SYBASE mode. For these modes, your SQL should not require any modification. TSQL1 and TSQL2 continue to work as they do in MDI Database Gateway for DB2, Version 2.05, but Sybase is planning to phase them out. These modes will not be defined or documented beyond what was provided for that version. See "Migrating from TSQL0, TSQL1, and TSQL2 modes" on page 55 for more information.

When invoking an RSP, the client application can specify keyword variables, variable text, or input pipes to pass to the RSP. In turn, the RSP uses pointers in the SPAREA to access the values. Keyword variables have the typical MVS format of &VARNAME=value. The client application passes values according to the DirectConnect TSQL setting for SQL transformation.

Invoking RSPs through Access Service Library

This section explains how to use the PASSTHROUGH and SYBASE transformation mode commands to invoke RSPs through Access Service Library. It also explains how to pass keyword variables and variable text, and how to handle quotes in variables.

Using the PASSTHROUGH mode commands

If the DirectConnect TSQL configuration property is set to PASSTHROUGH, use this command syntax to invoke RSPs:

```
USE PROCEDURE procedurename
```

If you pass variables to the RSP, you must also supply the appropriate arguments in the invoking statement, and the form of the arguments must match the SQL transformation level. See your Transact-SQL manual for more information on variables and arguments.

Passing keyword variables

Use this command syntax to pass keyword variable values to the RSP:

```
USE PROCEDURE procedurename &VARNAME1=value1
&VARNAME2=value2 ... &VARNAMEn=valuen
```

Passing variable text

The client application passes variable text to the RSP as a single text string; the RSP is responsible for interpreting the string.

If the DirectConnect TSQL configuration property is set to PASSTHROUGH, use this command syntax to pass variable text to the RSP:

```
USE PROCEDURE procedurename valuestring
```

Note There is a 32K limit for variable text string size for DB2 Access Service. This limit is not valid for TRS Access Service.

Using the SYBASE mode command

If the DirectConnect TSQL configuration property is set to SYBASE, use this command syntax to invoke RSPs:

```
EXECUTE procedurename
```

If you pass variables to the RSP, you must also supply the appropriate arguments in the invoking statement, and the form of the arguments must match the SQL transformation level. See your Transact-SQL manual for more information on variables and arguments.

Passing keyword variables

Use this command syntax to pass keyword variable values to the RSP:

```
EXECUTE procedurename @VARNAME1=value1,
@VARNAME2=value2 ... , @VARNAMEn=valuen
```

With TSQL set to SYBASE, you must comply with Transact-SQL syntax for variables. In particular, be sure to prefix your variable names with the at sign (@) instead of the ampersand (&) and to separate the variables with commas.

Passing variable text

The client application passes variable text to the RSP as a single text string; the RSP is responsible for interpreting the string. When using variable text, you can include an unlimited number of variables in the string.

Note There is a 32K limit for variable text string size.

- If TSQL is set to PASSTHROUGH, use this command syntax to pass variable text to the RSP:

  ```
  USE PROCEDURE procedurename valuestring
  ```

- If TSQL is set to SYBASE, use this command syntax to pass variable text to the RSP:

  ```
  EXECUTE procedurename valuestring
  ```

Handling quotes in variables

In some cases, the values the client application sends to the RSP contain quotation mark characters, either single or double. Because these characters are frequently used as string delimiters, DirectConnect can misinterpret strings containing quotes. Therefore, it may transform the values in ways that the RSP does not expect, for example by replacing the carriage return-linefeed sequence (CR/LF) with spaces.

To provide maximum control over quote handling in USE statements, Sybase implemented the following rules:

Note These rules apply only if your setting is TSQL1 or PASSTHROUGH.

- The first non-white-space character following the procedure or request name is tested by MainframeConnect for the possibility that it is a special delimiter. Special delimiters can be used to enclose the entire set of argument strings sent to the request or RSP. If the argument string is enclosed by such delimiters, then the characters between the delimiters (including the delimiters themselves) are not modified in any way. In other words, quote processing, uppercasing and so on, is not performed by MainframeConnect.
- DirectConnect recognizes a character as a delimiter if it is a member of the following set of characters:

```
! % ( ) * / : << >> ? \ ' { } | ~
```

Note The same delimiter character must be used at both ends of the string: for example, *(xxxxxxxx(* or *{xxxxxxxx{* (not *(xxxxxxxx)*).

If the first non-white-space character is not a delimiter, then MainframeConnect handles quotes according to the following standard TSQL1 rules:

- It passes doubled occurrences of either quote character—that is, '' or ""—without modification.
- It assumes the first single occurrence of either quote character is a delimiter beginning a quoted string, and it assumes the next single occurrence of the same character ends the quoted string.
- It compares the delimiter to the setting in the DirectConnect configuration (*.cfg*) file, and converts the delimiter if required; that is, double quotes may be converted to single quotes.

- It passes occurrences of the other quote characters (that is, double quotes occurring in a string delimited by single quotes or single quotes occurring in a string delimited by double quotes) without modification.

Invoking RSPs through TRS

If you invoke the RSP through TRS, use this command syntax:

```
EXEC rpcname
```

Passing keyword variables

Use this command syntax to pass keyword variable values to the RSP:

```
EXEC rpcname @VARNAME1='value1', @VARNAME2='value2' ...
, @VARNAMEn='valuen'
```

Passing variable text

The client application passes variable text to the RSP as a single text string; the RSP is responsible for interpreting the string. When using variable text, the number of variables you can include in the string is unlimited.

Note There is a 32K limit for variable text string size.

If TSQL is set to SYBASE, use this command syntax to pass variable text to the RSP:

```
EXEC rpcname 'value'
```

Migrating from TSQL0, TSQL1, and TSQL2 modes

TSQL0 corresponds to PASSTHROUGH mode, and TSQL2 corresponds to SYBASE mode. For these modes, your SQL should not require any modification.

If you used TSQL1 mode for earlier releases, review your SQL.

If you migrate to a setting of PASSTHROUGH mode, your code will probably fail because the TSQL1 partial conversion does not occur. If you migrate to a new setting of SYBASE mode, your code should work because DirectConnect passes any SQL statement that the parser cannot identify on to the server without changes.

Sending data to the RSP

You can use STD input pipes to send data to an RSP only if your DirectConnect TSQL setting is PASSTHROUGH (or TSQL0 or TSQL1 for backward compatibility only). You can send ASCII data through parameters and pipes; however, binary data can only be sent through pipes.

Note If your DirectConnect setting is SYBASE (or TSQL2, for backward compatibility only), you must pass data as parameters.

When invoking an RSP, the client application can send ASCII formatted data or binary data. If it sends binary data, see "Sending binary data" on page 57.

Sending ASCII-formatted data

To send ASCII data to an RSP, you use this command syntax:

```
USE PROCEDURE WITH DATA rspname [keywords or variable text];


ASCII data records
```

The following list describes the previous syntax:

- The WITH DATA clause appends input records.
- A carriage return or line feed separates data records.
- A semicolon and carriage return/linefeed must separate the USE PROCEDURE clause from the data.
- When another statement follows the data records, the data records must end with a semicolon on a line by itself.

This is an example of ASCII-formatted data:

```
521-44-3201  JOHN SMITH             1991-04-16 00004 012.25
 521-56-4368  JERRY GREEN            1987-11-02 00001 018.75
 522-63-7188  SALLY JONES            1988-09-21 00002 015.00
```

```
 521-44-3201  BILL SMITH             1981-12-16 00004 012.25
 521-56-4368  GEORGE BROWN           1986-05-24 00001 018.75
 522-63-7188  KATHY JOHNSON         1987-09-19 00002 015.00
```

Sending binary data

The client application can send RSPs binary input data using a BIN-format input pipe. The client application specifies the USE PROCEDURE statement using the WITH BINARY DATA option in this command syntax:

To send binary data to an RSP, use this command syntax:

```
USE PROCEDURE WITH BINARY DATA rspname [keywords or variable text];


....binary data....
```

The following describes the syntax:

- The WITH BINARY DATA clause appends the input file as binary data.
- rspname represents the name of the RSP.
- A semicolon and carriage return/linefeed must separate the USE PROCEDURE clause from the data.

  The RSP assumes all data between the semicolon and the end of the buffer is binary. Because there is no internal formatting in the binary file, the RSP must be able to interpret the data appropriately.
- With a BIN-format data pipe, ASCII-EBCDIC conversion does not occur.

Understanding input data requirements

All data, except binary, the client sends as input to the RSP must meet the following requirements:

- All characters must be printable ASCII characters (20–7F hexadecimal).
- Records must be delimited by either linefeed or carriage return/linefeed.

In PASSTHROUGH mode, input pipe data passes unchanged to the RSP, except that control characters are deleted and ASCII is converted to EBCDIC. All line feeds in the input data serve to separate data records, and their positions control what the RSP receives as a single record.

CHAPTER 6

Troubleshooting

This chapter describes the following topics:

- Overview
- MainframeConnect errors related to RSPs
- Troubleshooting errors

Overview

This chapter describes how to use the output records of an RSP to troubleshoot problems in the RSP.

MainframeConnect errors related to RSPs

Your RSP receives error messages, if there are any, in the SPRC field of the SPAREA.

MainframeConnect invokes the RSP through the CICS LINK command, which causes the CICS program table to be searched for the RSP name.

If CICS does not find the RSP name, one of three messages returns:

- If DB2 does not exist in this CICS region, then MainframeConnect returns a RSP or REQUEST not found message or a CICS *Abend AEY9*.
- If DB2 does exist in this CICS region but the host request table does not exist, then MainframeConnect returns a RSP or REQUEST not found message.
- If DB2 and the host request table both exist but the RSP name is not in the table, then MainframeConnect returns an RSP or REQUEST not found message.

See Mainframe Connect DB2 UDB Option for CICS *Installation and Administration Guide* for the valid message numbers, the message text, the reason the message was issued, and the required action.

Note snaping and cicsping are troubleshooting programs available with MainframeConnect. See Mainframe Connect DB2 UDB Option for CICS *Installation and Administration Guide* for more information.

Troubleshooting errors

This section covers DB2 errors, and what to do if ASRA abends at PUTPIPE and at OPENPIPE.

DB2 errors

If you receive a DB2 -805 error when you execute RSPs that access DB2, ensure that:

- Pooled threads are specified
- The package was bound with the current database request module (DBRM)

If you receive other DB2 error messages, refer to your DB2 documentation.

CICS ASRA abend errors

ASRA is an abend error indicating that CICS found a problem in a program that was running. It is the most common CICS abend.

If a CICS ASRA abend (OC4) occurs at PUTPIPE

There are two common causes of ASRA abends at the PUTPIPE command: a SQLLEN packed decimal error and VARCHAR or LVARCHAR definition error.

A SQLLEN packed decimal error

Defining packed decimals in the SQLDA is a common source of errors. When you define the length of a packed decimal in the SQLLEN field, the length is a decimal translation of hexadecimal '*PPSS*', where:

- *PP* (precision) is the number of total digits in the decimal.
- *SS* (scale) is the number of those digits to the right of the decimal.

An incorrect length causes an ASRA abend at the PUTPIPE command. The following table shows how the problem can occur.

Table 6-1: Coding decimal and hexadecimal values

Code	Picture	Hex value	Decimal value
PIC S9(03)V99	nnn.nn	X'0502'	'1282'
PAC S9(11)V99	nnnnnnnnnnn.nn	X'0D02'	'3330'

You can calculate the hex value using the following formula:

```
pp x 256 + ss = length
```

where *pp* is precision and *ss* is scale.

For example:

```
05 SQLLEN PIC S9(4) COMP VALUE +3330.
13 x 256 + 02 = 3330
```

You can avoid decimal translation by redefining the *SQLLEN* field as a PIC(2) with a hexadecimal value:

```
05 SQLLEN-X                  PIC X(2)VALUE X'0D02'.
 05 SQLLEN REDEFINES SQLLEN-X PIC S9(4)COMP.
```

VARCHAR or LVARCHAR definition error

When *VARCHAR* and *LVARCHAR* are defined in the LINKAGE SECTION, they each require a preceding 2-byte field for their length. Not including this length field causes an ASRA abend at the PUTPIPE command.

The code must include a computed field, which passes the amount of space that is required for the text:

```
01 VARCHAR-HOLD.
     05 VARCHAR-LENGTH     PIC S9(4)COMP.
     05 VARCHAR-TEXT       PIC X(200).
```

If the code omits the computed field, the first two characters in the text field are used for the length of the text field:

```
01 VARCHAR-HOLD.
        05 VARCHAR-TEXT        PIC X(200).
```

The hexadecimal value for alphas can be very large. The result is an ASRA abend, or even a CICS crash.

If a CICS ASRA abend occurs at OPENPIPE

Errors in the model SQLDA definition cause an ASRA abend at the OPENPIPE command. MainframeConnect does not check errors for the SQLDA structure, so any typing error causes an abend. Recheck the RSP code, or copy the SQLDA definition from another file.

APPENDIX A

RSP Commands

This appendix discusses the following topics:

- Command examples
- Commands

Command examples

The following examples show commands in assembler, COBOL II, PL/I, and C languages:

Assembler language example

```
MVC SPMODE,=C'INPUT'
 MVC SPFORMAT,=C'STD'
 MVC SPMAXLEN,=F'400'
 CALL OPENPIPE,SPAREA
```

COBOL I I language example

```
MOVE 'INPUT' TO SPMODE.
 MOVE 'STD' TO SPFORMAT.
 MOVE 400 TO SPMAXLEN.
 CALL 'OPENPIPE' USING SPAREA.
```

PL/I language example

```
SPMODE='INPUT';
 SPFORMAT='STD';
 SPMAXLEN=400;
 CALL OPENPIPE(SPAREA);
```

C language example

```
memcpy(spPointer->spmode, "INPUT ",
    sizeof(spPointer->spmode));
 memcpy(spPointer->spformat, "STD",
    sizeof(spPointer->spformat));
 spPointer->spmaxlen = 400;
 openpipe(spPointer);
```

Note All the other examples in the command explanations in this appendix are in COBOL II.

Commands

The following RSP commands are explained in this appendix:

CLOSPIPE

Description

Closes a data pipe.

Syntax

Syntax varies with the programming language.

Examples

COBAL II

1 Closing an input pipe:

```
MOVE 'INPUT' TO SPMODE.
 CALL 'CLOSPIPE' USING SPAREA.
```

2 Closing an output pipe:

```
MOVE 'OUTPUT' TO SPMODE.
 CALL 'CLOSPIPE' USING SPAREA.
```

Usage

Properties

The CLOSPIPE command uses the value from the SPAREA field SPMODE (see "SPMODE" on page 140), which specifies whether the data pipe is opened for input or output.

COMMIT

Description — Commits database processing of the most recent unit of work.

Syntax — Syntax varies with the programming language.

Examples — **COBAL II** The equivalent to SYNCPOINT is:

```
CALL 'COMMIT' USING SPAREA.
```

Usage — The RSP COMMIT command is provided because the standard SQL COMMIT statement cannot be executed in CICS environments. MainframeConnect converts the command to the equivalent CICS SYNCPOINT command.

GETPIPE

Description — Reads data records from an input pipe.

Syntax — Syntax varies with the programming language.

Note STD and BIN pipes are the only valid formats for the GETPIPE command.

Parameters — The GETPIPE command uses values from these SPAREA fields:

- SPINTO (see "SPINTO" on page 140) specifies the address of the RSP storage area to receive the input data. MainframeConnect places the data record into this area.
- SPRECLEN (see "SPRECLEN" on page 141) specifies the length of the data record. Open ServerConnect sets the SPRECLEN for a GETPIPE.

Note GETPIPE is used with Access Service Library only; it is not used with TRS.

Examples — **COBOL II** This example reads data from a STD format input pipe into the *DATAREC* storage area (*DATAREC* is a data area defined in the RSP program):

```
SET ADDRESS OF DATAREAC TO SPINTO.
 CALL 'GETPIPE' USING SPAREA.
```

Usage

- If you write fixed-length records of the same size as SPMAXLEN, the SPRECLEN value is not required.

- However, when you have both an input pipe and an output pipe open, both pipes use this field and each must set the field value before writing or reading the record. See “Transmitting fixed-length or variable-length records” for more information.

MESSAGE

Description

Communicates error and informational messages to the client application.

Syntax

Syntax varies with the programming language.

Examples

COBOL II 1Provide the message text:

```
MOVE 'E' TO SPSTATUS.
 MOVE 'DATA REQUESTED CANNOT BE FOUND' TO SPMSG.
 CALL 'MESSAGE' USING SPAREA.
```

2 Repeat the message previously stored in SPMSG:

```
MOVE 'E' TO SPSTATUS.
 CALL 'MESSAGE' USING SPAREA.
```

Usage

The MESSAGE command uses values from these SPAREA fields:

- SPMSG (see “SPMSG” on page 141) specifies the message text. Message text can be up to 100 bytes long.
- SPSTATUS (see “SPSTATUS” on page 139) specifies processing status. Use one of these codes:
  - OK indicates success.
  - E indicates an error.
  - W indicates a warning.

Your RSP can issue as many MESSAGE commands as you need. The RSP API sends the messages to the client application immediately.

To send messages and status to the client, the RSP places message text in an SPAREA field (SPMSG) and issues the RSP MESSAGE command, which signals to the RSP API that a message is ready to be sent.

Note A call to MESSAGE cannot be made between an OPENPIPE and a PUTPIPE.

OPENPIPE

Description

Opens a data pipe either to send output to or receive input from the client application.

Syntax

Syntax varies with the programming language.

Examples

COBOL II

1 Open a STD output pipe:

```
MOVE 'OUTPUT' TO SPMODE.
 MOVE 'STD' TO SPFORMAT.
 MOVE 450 TO SPMAXLEN.
 CALL 'OPENPIPE' USING SPAREA.
```

2 Open a BIN input pipe:

```
MOVE 'INPUT' TO SPMODE.
 MOVE 'BIN' TO SPFORMAT.
 MOVE 625 TO SPMAXLEN.
 CALL 'OPENPIPE' USING SPAREA.
```

Usage

The OPENPIPE command uses values from these SPAREA fields:

- SPMODE (see “SPMODE” on page 140) specifies whether the data pipe is opened for input or output.
  - INPUT indicates the RSP reads data records sent from the client application.
  - OUTPUT indicates the RSP writes data records to be sent to the client application.
- SPFORMAT (see “SPFORMAT” on page 140) specifies the data pipe format.
  - STD indicates standard format, in which each data record is transmitted to or from the client application as a single-text column record.
  - BIN indicates a single-binary column format, like STD, except that the data is binary. No ASCII-EBCDIC or EBCDIC-ASCII conversion occurs on binary data.

Note Use STD and BIN only for input pipes.

- DB2 indicates data is transmitted from the RSP as a multiple-column record, where the column definitions are contained in an associated SQLDA. The SQLDA is a collection of variables and pointers that provide column information about data being transmitted to the client application. See Appendix G, "The SQLDA" for more information.

Note Use DB2 only for output pipes.

- SPMAXLEN (see "SPMAXLEN" on page 141) specifies the maximum size, in bytes, of the data records written to or read from the data pipe.
- SPSQLDA (see "SPSQLDA" on page 140) specifies the address of a SQLDA that describes the content of the data records. *Use only for output pipes.*
- STD and BIN format pipes must use SPMAXLEN to identify the maximum record length.
- For DB2 format pipes, the RSP must supply the SPSQLDA address. DB2 format pipes must use SPSQLDA.
- Both an input pipe and an output pipe can be open at the same time.
- As part of opening a pipe, you must specify the format of the data the pipe handles. RSPs can handle DB2, BIN, and STD format data. See Chapter 2, "Designing an RSP" for more information on these formats.
- When a data pipe of any format opens for output with the OPENPIPE command, it issues Open Server describe and bind commands. You cannot subsequently change the maximum column length of any columns or types in the SQLDA definition when you issue a PUTPIPE command.

PUTPIPE

Description — Writes data records to an output pipe. Open ServerConnect then reads the records and sends them to the client application.

Syntax — Syntax varies with the programming language.

Examples — **COBOL II** This example writes a 130-byte data record built in a storage area called AREA1 to a STD format input pipe:

```
MOVE 130 TO SPRECLEN.
 SET ADDRESS OF AREA1 TO SPFROM.
```

```
CALL 'PUTPIPE' USING SPAREA.
```

Usage

The PUTPIPE command uses values from these SPAREA fields:

- SPFROM (see "SPFROM" on page 140) specifies the address of the data record.
- SPRECLEN (see "SPRECLEN" on page 141) specifies the length of the data record.
- SPSQLDA (see "SPSQLDA" on page 140) provides the SQLDA address.
- Only STD and BIN format pipes use the SPFROM field. For a DB2 format pipe, the SQLDA describes the location and length of the data columns.
- If you have a single output pipe open, you can set the SPFROM value once for all records. However, when you have both an input pipe and an output pipe open, both pipes use this field and each must set the field value before writing or reading the record.
- For STD and BIN pipes, the SPRECLEN value must not exceed the value that was specified for SPMAXLEN (see "SPMAXLEN" on page 141) when the pipe was opened.
- If you write fixed-length records of the same size as SPMAXLEN, the SPRECLEN value is not required.

ROLLBACK

Description

Rolls back database processing to the last syncpoint (COMMIT).

Syntax

Syntax varies with the programming language.

Examples

COBOL II The equivalent to SYNCPOINT WITH ROLLBACK is:

```
CALL 'ROLLBACK' USING SPAREA.
```

Usage

The RSP ROLLBACK command is provided because the standard SQL ROLLBACK statement cannot be executed in CICS environments. MainframeConnect converts the command to the equivalent CICS SYNCPOINT WITH ROLLBACK command.

RPDONE

Description — Ends processing for an RSP invoked through TRS.

Syntax — Syntax varies with the programming language.

Examples — **COBOL II**

```
CALL 'RPDONE' USING SPAREA.
```

Usage

- This must be the last API call in an RSP invoked through TRS.
- It cleans up RSP memory (the SPAREA) because MainframeConnect is not involved.

RPSETUP

Description — Initiates an RSP invoked through TRS.

Syntax — Syntax varies with the programming language.

Examples — **COBOL II**

```
CALL 'RPSETUP' USING SPAREA.
```

Usage — This must be the first API call in an RSP invoked through TRS. It is used because MainframeConnect is not involved. It allocates and initializes memory for the SPAREA.

STATUS

Description — Communicates to MainframeConnect the success or failure of the processing it performed.

Syntax — Syntax varies with the programming language.

Examples — **COBOL II** This example sets the status to indicate an error condition:

```
MOVE 'E' TO SPSTATUS.
 CALL 'STATUS' USING SPAREA.
```

Usage — The STATUS command uses the SPSTATUS field (see “SPSTATUS” on page 139) to specify processing status. Use one of these codes:

- 'OK' indicates success.
- 'E' indicates an error.
- 'W' indicates a warning.

- STATUS releases results and messages to the client application.
- An RSP must issue at least one STATUS command. If an RSP terminates without issuing a STATUS command, MainframeConnect automatically issues a STATUS message indicating an error occurred.
- For each result set returned to the client application, the RSP must issue a STATUS command after the output pipe closes. Issuing a STATUS command while a data pipe is open automatically closes the pipe.
- An RSP can issue the STATUS command as many times as necessary.

APPENDIX B

MODELRSP DB2 Output Pipe Sample RSP

If you want to write an RSP with DB2-formatted output pipes or multiple column rows, review MODELRSP.

This appendix discusses the following topics:

- Understanding MODELRSP
- The SPAREA in MODELRSP
- The SQLDA in MODELRSP
- Invoking MODELRSP from the client application
- MODELRSP DB2 output pipe sample code

Understanding MODELRSP

MODELRSP is a RSP sample COBOL II program that provides examples of:

- Using a DB2-format output pipe
- Defining a SQLDA with all possible datatypes represented
- Using the SPAREA to communicate with MainframeConnect
- Using the RSP commands to manage a data pipe and communicate status
- Sending data to the client application
- Handling errors

In the MODELRSP example, keyword variables, variable text, or data are not sent as input to the RSP. The sample program is shown in its entirety. The program also contains many in-line comments (denoted with standard asterisks) to explain the flow of processing and clarify points.

For simplicity, the example does not include database access code. Instead, it sends 11 columns of employee data to illustrate 11 types of data you can transmit to the client application.

The SPAREA in MODELRSP

This section describes how MODELRSP uses SPAREA fields and RSP commands, as well as a brief example of the SPAREA from MODELRSP.

How MODELRSP uses SPAREA fields

This section explains how MODELRSP uses the return code, status, and message fields. See Appendix F, "The SPAREA" for detailed information on all SPAREA fields.

SPRC

The SPRC (return code) field communicates the success or failure of an RSP command.

Note Your code should check the SPRC field after issuing any RSP command.

The following MODELRSP code fragment shows how an RSP accesses the SPRC field to get this information:

```
IF SPRC IS NOT EQUAL TO '000'
 MOVE WS-CLOSPIPE         TO ERROR1-CALL
 PERFORM 9800-PIPE-ERROR-MSG THRU 9800-EXIT
 GO TO 9999-RETURN-TO-CALLER.
```

SPSTATUS

The SPSTATUS field communicates processing status in the remote database to the RSP. As shown in the following MODELRSP code fragment, the RSP also uses the SPSTATUS field to communicate status on its own processing to the client application.

```
MOVE 'OK' TO SPSTATUS.
CALL 'STATUS' USING SPAREA.
```

SPMSG

The SPMSG field communicates messages back to the client application. Then the SPAREA issues the RSP MESSAGE command as shown in the following modified MODELRSP code fragment:

```
 MOVE SPRC                    TO ERROR1-SPRC.
```

```
 MOVE ERROR1-MSG              TO SPMSG.
 MOVE 'E'                     TO SPSTATUS.
CALL 'MESSAGE' USING SPAREA.
```

In this case, the client application receives the error message in SPMSG.

You can issue the MESSAGE command with message text of up to 100 bytes with USING SPAREA:

```
MOVE 'OK'                       TO SPSTATUS.
MOVE 'THIS IS THE OK MESSAGE'    TO SPMSG.
CALL 'MESSAGE' USING SPAREA.
```

Refer to Appendix A, “RSP Commands” for detail about the MESSAGE command.

Using RSP commands with the SPAREA

The MODELRSP program uses these RSP commands: OPENPIPE, PUTPIPE, CLOSPIPE, STATUS, and MESSAGE. In all the supported programming languages, the RSP commands are invoked with a standard CALL statement.

In COBOL II, the RSP command can be enclosed in single quotes; in the other supported languages, quotes are not necessary. The following COBOL II statements show how your RSP code must use the RSP commands.

Note Single quotes in a COBOL CALL statement indicate a “static call.”

```
CALL 'OPENPIPE'     USING SPAREA.
CALL 'PUTPIPE'      USING SPAREA
CALL 'CLOSPIPE'     USING SPAREA.
CALL 'STATUS'       USING SPAREA.
CALL 'MESSAGE'      USING SPAREA.
```

The previous sample shows:

- Data pipe mode and format values are moved to the corresponding SPAREA fields. Then the command is issued

  ```
  CALL ‘OPENPIPE’ USING SPAREA.
  ```

- Each PUTPIPE generates one result row. Therefore, your code must issue the PUTPIPE command for every row of data you send.

- A STATUS command always follows the CLOSPIPE command. This ensures the processing status is communicated to the client application and clears out the data pipe and all messages.

For more information on the RSP commands, their formats and results, see Appendix A, “RSP Commands.”

SPAREA example

In the following example, the LWKCOMMAREA is the RSP API communication area. SPAREAC (the sample COBOL II copy book provided on the Open ServerConnect base tape) is included in the linkage section with a COPY statement.

```
01  LWKCOMMAREA.
    COPY SPAREAC.
```

Further on in the program, the SPAREA fields pass information about the type of data pipe the RSP uses and the pointers to the SQLDA.

```
MOVE 'OUTPUT'              TO SPMODE.
 MOVE 'DB2'                 TO SPFORMAT.
 SET SPSQLDA                TO ADDRESS OF SQLDA.
CALL 'OPENPIPE'            USING SPAREA.
```

The following three SPAREA fields are used by the RSP to communicate to the Open ServerConnect RSP API:

- SPMODE specifies the mode (input or output) of the data pipe.
- SPFORMAT specifies the format (DB2, STD, or BIN) of the data to be transmitted through the pipe.
- SPSQLDA specifies the pointer to the SQLDA.

See “SPAREA field descriptions” on page 139 for more information on all the SPAREA fields.

The SQLDA in MODELRSP

MODELRSP shows you how to create a SQLDA definition to send along with data to the client application using a DB2 output pipe. (The SQLDA definition in the RSP provides the data structure information sent along with the data to the client.)

If you have not worked with a SQLDA definition, review Appendix G, "The SQLDA."

Note If the client application you are using expects data structure information to be transmitted with the data, use the DB2 format even if the data source is not DB2. For client application software, such as PowerBuilder, check data structure requirements in the vendor documentation.

Relating the standard SQLDA fields to the example from MODELRSP that follows, you can see the first SQLVAR definition is named MS-COL01. It is a fixed-character datatype that can contain nulls (value 453) and is defined for the first column of EMPLOYEE-DATA (FIXED-CHAR) that the sample RSP is sending to the client. MODELRSP includes one SQLVAR definition for each of the 11 columns of data it sends.

```
*****************************************************************
 *  DESCRIPTION OF THE MODEL SQLDA                               *
 *****************************************************************
 01 MODEL-SQLDA.
       03  MS-SQLDAID                 PIC X(08)  VALUE 'SQLDA   '.
       03  MS-SQLDABC                PIC S9(8)  COMP VALUE 500.
       03  MS-SQLN                   PIC S9(4)  COMP VALUE 11.
       03  MS-SQLD                   PIC S9(4)  COMP VALUE 11.
       03  MS-COL01.
 *      - 1ST   COLUMN DATATYPE = FIXED CHAR (LENGTH 1 - 256)
          05  MS-COL01-SQLTYPE       PIC S9(4)  COMP VALUE 453.
          05  MS-COL01-SQLLEN        PIC S9(4)  COMP VALUE 5.
          05  MS-COL01-SQLDATA       USAGE IS POINTER.
          05  MS-COL01-SQLIND        USAGE IS POINTER VALUE NULL.
          05  MS-COL01-SQLNAMEL      PIC S9(4)  COMP VALUE 10.
          05  MS-COL01-SQLNAME       PIC X(30) VALUE 'FIXED_CHAR'.
                      :
                      :
```

Invoking MODELRSP from the client application

The client application invokes MODELRSP using the command that corresponds to the SQL transformation setting (TSQL) on DirectConnect:

PASSTHROUGH TSQL setting

```
USE PROCEDURE MODELRSRSP
```

SYBASE TSQL setting

```
EXECUTE MODELRSP
```

MODELRSP DB2 output pipe sample code

```
IDENTIFICATION DIVISION.
 PROGRAM-ID.   MODELRSP.
 AUTHOR.       SYBASE ICD.
 DATE-WRITTEN. SEPTEMBER 15, 1993.
*****************************************************************
*  MODELRSP - SAMPLE TO ILLUSTRATE SQLDA USAGE.                 *
*                                                               *
*  THIS SAMPLE STORED PROCEDURE HAS A LOT OF INTERNAL           *
*  DOCUMENTATION TO HELP EXPLAIN AND ILLUSTRATE THE PROPER      *
*  USAGE OF THE SQLDA FOR A DB2 OUTPUT PIPE.  A ROW IS SET UP   *
*  FOR ALL DATATYPES AND ALL WILL BE SET TO ALLOW NULLS.        *
*                                                               *
*****************************************************************

 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
01 FILLER                        PIC X(27)  VALUE
                                'WORKING-STORAGE STARTS HERE'.

 01  COMMAREA-POINTER               USAGE IS POINTER.
 01  SQLDA-POINTER                  USAGE IS POINTER.
 01  EMPLOYEE-DATA-POINTER          USAGE IS POINTER.
 01  INDICATOR-VAR-POINTER          USAGE IS POINTER.
```

```
  01  SQLDA-SIZE                      PIC S9(8)  COMP.

  01 WS-LITERALS.
      05 WS-STATUS                 PIC X(06)  VALUE 'STATUS'.
      05 WS-MESSAGE                PIC X(07)  VALUE 'MESSAGE'.
      05 WS-COMMIT                 PIC X(06)  VALUE 'COMMIT'.
      05 WS-ROLLBACK               PIC X(08)  VALUE 'ROLLBACK'.
      05 WS-OPENPIPE               PIC X(08)  VALUE 'OPENPIPE'.
      05 WS-PUTPIPE                PIC X(07)  VALUE 'PUTPIPE'.
      05 WS-GETPIPE                PIC X(07)  VALUE 'GETPIPE'.
      05 WS-CLOSPIPE               PIC X(08)  VALUE 'CLOSPIPE'.

  01 MESSAGES.
     05  ERROR1-MSG.
         07  ERROR1-TEXT1          PIC X(19)  VALUE
             'ERROR WITH CALL TO '.
         07  ERROR1-CALL           PIC X(10)  VALUE SPACES.
         07  ERROR1-TEXT2          PIC X(14)  VALUE
             ' - SPRC CODE: '.
         07  ERROR1-SPRC           PIC X(03)  VALUE SPACES.
     05  ERROR2-MSG.
         07  ERROR2-TEXT2          PIC X(46)  VALUE SPACES.
     05  WS-LONG-VARCHAR-TEXT.
         07  FILLER                PIC X(50)  VALUE
         'THIS IS A LINE OF VERY LONG TEXT TO DEMONSTRATE TH'.
         07  FILLER                PIC X(50)  VALUE
         'AT A LONG VARCHAR DATATYPE CAN BE SENT DOWN A DB2 '.
         07  FILLER                PIC X(50)  VALUE
         'OUTPUT PIPE WITH NO PROBLEMS, WORRIES, OR CONSTERN'.
         07  FILLER                PIC X(50)  VALUE
         'ATION, AS LONG AS ONE REMEMBERS THAT LARGE AMOUNTS'.
         07  FILLER                PIC X(50)  VALUE
         ' OF DATA WILL ALWAYS HAVE AN ELEMENT OF UNEXPECTED'.
         07  FILLER                PIC X(50)  VALUE
         'NESS.  EVEN SO, USE SYBASE FOR ALL YOUR SOLUTIONS.'.

*****************************************************************
 *  DESCRIPTION OF THE MODEL SQLDA                               *
 *****************************************************************
 *
 *    SQLTYPES USED IN SQLDA:
 *    VALUE     DATA TYPE            NULLS ALLOWED
 *    =======   ==================   =============
 *    384/385   DATE                 NO/YES
 *    388/389   TIME                 NO/YES
 *    392/393   TIMESTAMP            NO/YES
```

```
 *    448/449   CHAR VARIABLE LENG   NO/YES
 *    452/453   CHAR FIXED LENGTH    NO/YES
 *    456/457   CHAR LONG VARIABLE   NO/YES
 *    480/481   FLOATING-POINT       NO/YES
 *    484/485   DECIMAL              NO/YES
 *    496/497   LARGE INTEGER        NO/YES
 *    500/501   SMALL INTEGER        NO/YES
 ******************************************************
 *  NOTE: ALL DATATYPES IN THIS EXAMPLE ARE DEFINED AS NULLABLE
 ******************************************************
 *-------------------------------------------------------------*
 * MODEL-SQLDA IS USED TO HOLD THE COLUMN DESCRIPTIONS IN     *
 * WORKING STORAGE.  THIS IS DONE THIS WAY BECAUSE YOU CANNOT *
 * USE VALUE CLAUSES IN A COBOL LINKAGE SECTION....           *
 *-------------------------------------------------------------*
  01 MODEL-SQLDA.
*     - EYE CATCHER - MUST ALWAYS SAY 'SQLDA   '.
      03  MS-SQLAID                 PIC X(08)  VALUE 'SQLDA   '.
 *     - SIZE OF SQLDA = 16 + (44 * SQLN VALUE)
      03  MS-SQLDABC                PIC S9(8)  COMP VALUE 500.
 *     - NUMBER OF SQLVAR OCCURENCES
 *     - MUST MATCH VALUE OF MS-SQLD
      03  MS-SQLN                   PIC S9(4)  COMP VALUE 11.
 *     - NUMBER OF SQLVAR OCCURENCES ACTUALLY USED
 *     - MUST MATCH VALUE OF MS-SQLN
      03  MS-SQLD                   PIC S9(4)  COMP VALUE 11.
      03  MS-COL01.
 *      - 1ST   COLUMN DATATYPE = FIXED CHAR (LENGTH 1 - 256)
          05  MS-COL01-SQLTYPE       PIC S9(4)  COMP VALUE 453.
          05  MS-COL01-SQLLEN        PIC S9(4)  COMP VALUE 5.
 *          -  SQLDATA WILL BE SET TO ADDRESS OF DATA FIELD
          05  MS-COL01-SQLDATA       USAGE IS POINTER.
 *          -  SQLIND WILL BE SET TO ADDRESS OF A S9(4) COMP FIELD
 *          -  WHEN COMP FIELD'S VALUE IS LESS THAN ZERO THEN
 *          -  COLUMN IS NULL - ONLY USED WHEN COLUMN IS NULLABLE
          05  MS-COL01-SQLIND        USAGE IS POINTER VALUE NULL.
 *          -  SQLNAMEL IS THE LENGTH OF THE COLUMN NAME
          05  MS-COL01-SQLNAMEL      PIC S9(4)  COMP VALUE 10.
 *          -  SQLNAME IS ALWAYS 30 IN LENGTH
          05  MS-COL01-SQLNAME   PIC X(30) VALUE 'FIXED_CHAR'.
      03  MS-COL02.
 *      - 2ND   COLUMN DATATYPE = DATE (LENGTH ALWAYS 10)
          05  MS-COL02-SQLTYPE       PIC S9(4) COMP VALUE 385.
          05  MS-COL02-SQLLEN        PIC S9(4) COMP VALUE 10.
          05  MS-COL02-SQLDATA       USAGE IS POINTER.
          05  MS-COL02-SQLIND        USAGE IS POINTER VALUE NULL.
```

```
          05  MS-COL02-SQLNAMEL      PIC S9(4) COMP VALUE 4.
          05  MS-COL02-SQLNAME       PIC X(30) VALUE 'DATE'.
      03  MS-COL03.
*       - 3RD   COLUMN DATATYPE = VARIABLE LENGTH CHAR (1-256)
          05  MS-COL03-SQLTYPE       PIC S9(4) COMP VALUE 449.
          05  MS-COL03-SQLLEN        PIC S9(4) COMP VALUE 30.
          05  MS-COL03-SQLDATA       USAGE IS POINTER
          05  MS-COL03-SQLIND        USAGE IS POINTER VALUE NULL.
          05  MS-COL03-SQLNAMEL      PIC S9(4) COMP VALUE 7.
          05  MS-COL03-SQLNAME       PIC X(30) VALUE 'VARCHAR'.
      03  MS-COL04.
*       - 4TH COL - DATATYPE = SMALL INTEGER (LENGTH ALWAYS 2)
*       - CORRESPONDING PIC S9(4) COMP - UP TO 5 DIGITS.
          05  MS-COL04-SQLTYPE       PIC S9(4) COMP VALUE 501.
          05  MS-COL04-SQLLEN        PIC S9(4) COMP VALUE 2.
          05  MS-COL04-SQLDATA       USAGE IS POINTER.
          05  MS-COL04-SQLIND        USAGE IS POINTER VALUE NULL.
          05  MS-COL04-SQLNAMEL      PIC S9(4) COMP VALUE 9.
          05  MS-COL04-SQLNAME       PIC X(30) VALUE 'SMALL_INT'.
      03  MS-COL05.
*       - 5TH COL - DATATYPE = PACKED DECIMAL
          05  MS-COL05-SQLTYPE       PIC S9(4) COMP VALUE 485.
*-------------------------------------------------------------*
*       - NOTE: FOR PACKED DECIMAL DATATYPES ONLY!!!!!        *
*       - LENGTH IS DECIMAL TRANSLATION OF HEX "PPSS"         *
*         (PRECISION AND SCALE)                               *
*       - WHERE "PP" = NUMBER OF TOTAL DIGITS                 *
*       -   AND "SS" = NUMBER OF DIGITS TO RIGHT OF DECIMAL   *
*       -  S9(3)V99 COMP-3 WOULD BE X'0502' OR IN DEC '1282'  *
*       -  S9(11)V99 COMP-3 WOULD BE X'0D02' OR IN DEC '3330' *
*       -  SQLLEN = (PP * 256) + SS                           *
*       -  1282=5*256+2==> FOR S9(3)V99                       *
*-------------------------------------------------------------*
          05  MS-COL05-SQLLEN        PIC S9(4) COMP VALUE +1282.
          05  MS-COL05-SQLDATA       USAGE IS POINTER.
          05  MS-COL05-SQLIND        USAGE IS POINTER VALUE NULL.
          05  MS-COL05-SQLNAMEL      PIC S9(4) COMP VALUE 10.
          05  MS-COL05-SQLNAME       PIC X(30) VALUE 'PACKED_DEC'.
      03  MS-COL06.
*       - 6TH COL - DATATYPE = TIME (LENGTH ALWAYS 8) 'HH.MM.SS'
          05  MS-COL06-SQLTYPE       PIC S9(4) COMP VALUE 389.
          05  MS-COL06-SQLLEN        PIC S9(4) COMP VALUE 8.
          05  MS-COL06-SQLDATA       USAGE IS POINTER.
          05  MS-COL06-SQLIND        USAGE IS POINTER VALUE NULL.
          05  MS-COL06-SQLNAMEL      PIC S9(4) COMP VALUE 4.
          05  MS-COL06-SQLNAME       PIC X(30) VALUE 'TIME'.
```

```
      03  MS-COL07.
*       - 7TH COL - DATATYPE = TIMESTAMP (LENGTH 19 OR 26)
*       - PIC X(19) VALUE 'YYYY-MM-DD:HH:MM:SS'
*       - PIC X(26) VALUE 'YYYY-MM-DD:HH:MM:SS:NNNNNN'
          05  MS-COL07-SQLTYPE       PIC S9(4) COMP VALUE 393.
          05  MS-COL07-SQLLEN        PIC S9(4) COMP VALUE 26.
          05  MS-COL07-SQLDATA       USAGE IS POINTER.
          05  MS-COL07-SQLIND        USAGE IS POINTER VALUE NULL.
          05  MS-COL07-SQLNAMEL      PIC S9(4) COMP VALUE 9.
          05  MS-COL07-SQLNAME       PIC X(30) VALUE 'TIMESTAMP'.
      03  MS-COL08.
*       - 8TH COL - DATATYPE = FLOAT (COMP-1 LENGTH ALWAYS 4)
*       -     SINGLE PRECISION FLOAT (COMP-1 LENGTH ALWAYS 4)
          05  MS-COL08-SQLTYPE       PIC S9(4) COMP VALUE 481.
          05  MS-COL08-SQLLEN        PIC S9(4) COMP VALUE 4.
          05  MS-COL08-SQLDATA       USAGE IS POINTER.
          05  MS-COL08-SQLIND        USAGE IS POINTER VALUE NULL.
          05  MS-COL08-SQLNAMEL      PIC S9(4) COMP VALUE 10.
          05  MS-COL08-SQLNAME       PIC X(30) VALUE 'FLOATING_P'.
      03  MS-COL09.
*       - 9TH COL - DATATYPE = FLOAT (COMP-2 LENGTH ALWAYS 8)
*       -     DOUBLE PRECISION FLOAT (COMP-2 LENGTH ALWAYS 8)
          05  MS-COL09-SQLTYPE       PIC S9(4) COMP VALUE 481.
          05  MS-COL09-SQLLEN        PIC S9(4) COMP VALUE 8.
          05  MS-COL09-SQLDATA       USAGE IS POINTER.
          05  MS-COL09-SQLIND        USAGE IS POINTER VALUE NULL.
          05  MS-COL09-SQLNAMEL      PIC S9(4) COMP VALUE 10.
          05  MS-COL09-SQLNAME       PIC X(30) VALUE 'DBL_FLOATP'.
      03  MS-COL10.
*       -10TH COL - DATATYPE = LARGE INTEGER (LENGTH ALWAYS 4)
*       - CORRESPONDING PIC S9(8) COMP - UP TO 10 DIGITS.
          05  MS-COL10-SQLTYPE       PIC S9(4) COMP VALUE 497.
          05  MS-COL10-SQLLEN        PIC S9(4) COMP VALUE 4.
          05  MS-COL10-SQLDATA       USAGE IS POINTER.
          05  MS-COL10-SQLIND        USAGE IS POINTER VALUE NULL.
          05  MS-COL10-SQLNAMEL      PIC S9(4) COMP VALUE 7.
          05  MS-COL10-SQLNAME       PIC X(30) VALUE 'INTEGER'.
     03  MS-COL11.
*       - 11TH  COL DATATYPE = LONG VARIABLE LENGTH CHAR (1-32K)
          05  MS-COL11-SQLTYPE       PIC S9(4) COMP VALUE 457.
          05  MS-COL11-SQLLEN        PIC S9(4) COMP VALUE 300.
          05  MS-COL11-SQLDATA       USAGE IS POINTER.
          05  MS-COL11-SQLIND        USAGE IS POINTER VALUE NULL.
          05  MS-COL11-SQLNAMEL      PIC S9(4) COMP VALUE 8.
          05  MS-COL11-SQLNAME       PIC X(30) VALUE 'LVARCHAR'.
```

```
*  THIS  SWITCH IS USED FOR TESTING IF RPC CALL
 77  RSPRPC-SWITCH   PIC S9(4) COMP VALUE 0.
     88 RPC-CALL                   VALUE 0.

LINKAGE SECTION.
*************************************************************
* THE LINKAGE SECTION DEFINES MASKS FOR DATA AREAS
* THAT ARE EITHER PASSED TO THE PROGRAM IN THE CASE OF THE
* COMMAREA OR CREATED BY THE PROGRAM IN THE CASE OF THE SQLDA
* AND DATA FIELDS.
*
* UNLIKE WORKING-STORAGE, STORAGE ASSOCIATED WITHIN THE LINKAGE
* SECTION IS AVAILABLE TO OTHER PROGRAMS BY PASSING ADDRESSES
* AND USING MASKS.
*
* IT IS IMPORTANT TO NOTE, THAT EVEN THOUGH THE DEFINES IN
* THE LINKAGE SECTION LOOK EXACTLY LIKE THOSE IN WORKING
* STORAGE, NO SPACE IS ASSOCIATED WITH THESE DEFINES IN LINKAGE
* UNTIL IT IS "GETMAINED".
*************************************************************

 01  DFHCOMMAREA.
     05  NOT-USED                 PIC X(1).

*****************************************************************
*  THIS IS THE ACTUAL SPAREA POINTER AND DEFINITION             *
*****************************************************************
 01  LWKCOMMAREA.
     COPY SPAREAC.

*****************************************************************
*  NULL INDICATOR VARIABLES - SET TO -1 IF NULL; 0 IF NOT NULL.  *
*  ONLY REQUIRED FOR COLUMNS DEFINED AS ALLOWING NULLS!          *
*****************************************************************
 01  INDICATOR-VARIABLES.
     10 FIXED-CHAR-IND       PIC S9(4) COMP.
     10 DATE-OUT-IND         PIC S9(4) COMP.
     10 VAR-CHAR-IND         PIC S9(4) COMP.
     10 SMALL-INT-IND        PIC S9(4) COMP.
     10 PACKED-DEC-IND       PIC S9(4) COMP.
     10 TIME-OUT-IND         PIC S9(4) COMP.
     10 TIMESTAMP-IND        PIC S9(4) COMP.
     10 FLOAT-SGL-IND        PIC S9(4) COMP.
     10 FLOAT-DBL-IND        PIC S9(4) COMP.
     10 LARGE-INT-IND        PIC S9(4) COMP.
     10 LARGE-VCHAR-IND      PIC S9(4) COMP.
```

```
*****************************************************************
 *  DESCRIPTION OF THE EMPLOYEE DATA                             *
 *****************************************************************
 *  NOTE THAT VARCHAR AND LONG-VARCHAR FIELDS ARE PRECEDED BY    *
 *  A TWO-BYTE COMP LENGTH FIELD.  SQLDA KNOWS NOT TO INCLUDE THE *
 *  EXTRA TWO BYTES IN THE LENGTH OF THE DATA.  WANT TO SEE YOUR  *
 *  REGION COME DOWN?  TRY LEAVING THE LENGTH FIELD OUT...       *
 *  THE FIRST TWO BYTES OF YOUR DATA WILL BE USED TO CALC THE    *
 *  LENGTH OF YOUR DATA AND CICS WILL START TO EAT ITSELF...     *
 *****************************************************************
  01  EMPLOYEE-DATA.
      10 FIXED-CHAR           PIC X(05).
      10 DATE-OUT             PIC X(10).
      10 VAR-CHAR.
         15 VCHAR-LENGTH      PIC S9(4) COMP.
         15 VCHAR-DATA        PIC X(30).
      10 SMALL-INT            PIC S9(4) USAGE COMP.
      10 PACKED-DEC           PIC S999V99 USAGE COMP-3.
      10 TIME-OUT             PIC X(08).
      10 TIMESTAMP            PIC X(26).
      10 FLOAT-SGL            COMP-1.
      10 FLOAT-DBL            COMP-2.
      10 LARGE-INT            PIC S9(8) USAGE COMP.
      10 LARGE-VAR-CHAR.
         15 L-VCHAR-LENGTH    PIC S9(4) COMP.
         15 L-VCHAR-DATA      PIC X(300).


*---------------------------------------------------------------*
 *  SQLDA - THIS IS USED AS A PLACE HOLDER IN THE COMMUNICATION  *
 *          AREA FOR THE COLUMN VALUES DESCRIBED IN THE MODEL-   *
 *          SQLDA. THIS IS DONE BECAUSE SYBASE USES POINTERS TO  *
 *          PASS DATA AND ADDRESS IN COBOL CAN ONLY BE SET IN THE *
 *          LINKAGE SECTION.........                             *
 *---------------------------------------------------------------*
  01  SQLDA.
      03  SQLDAID                  PIC X(8).
      03  SQLDABC                  PIC S9(8) COMP.
      03  SQLN                     PIC S9(4) COMP.
      03  SQLD                     PIC S9(4) COMP.
      03  SQLVARN                  OCCURS 11.
          05  SQLTYPE              PIC S9(4) COMP.
          05  SQLLEN               PIC S9(4) COMP.
          05  SQLDATA              USAGE IS POINTER.
          05  SQLIND               USAGE IS POINTER.
```

```
          05  SQLNAMEL               PIC S9(4) COMP.
          05  SQLNAME                PIC X(30).

*-------------------------------------------------------*
  PROCEDURE DIVISION.
 *-------------------------------------------------------*
      EXEC CICS HANDLE CONDITION
           INVREQ(9999-RETURN-TO-CALLER)
           END-EXEC.

 0000-MAIN-PROCESSING.

      PERFORM 1000-INITIALIZATION         THRU 1000-EXIT.

      PERFORM 5000-PROCESS-DATA           THRU 5000-EXIT.

      PERFORM 9000-WRAP-UP                THRU 9000-EXIT.

      EXEC CICS
           RETURN
      END-EXEC.

      GOBACK.

*-------------------------------------------------------*
  1000-INITIALIZATION.
 *-------------------------------------------------------*

      PERFORM 1050-SPAREA-SETUP           THRU 1050-EXIT.

      PERFORM 1100-TEST-SQLDA             THRU 1100-EXIT.

      PERFORM 1200-GET-STORAGE            THRU 1200-EXIT.

      PERFORM 1300-SET-ADDRESSES          THRU 1300-EXIT.

      PERFORM 1400-OPEN-OUTPUT-PIPE       THRU 1400-EXIT.

  1000-EXIT.
       EXIT.

*-------------------------------------------------------*
  1050-SPAREA-SETUP.
 *-------------------------------------------------------*

 ************************************************
```

```
* IF THIS IS A RPC CALL, CALL RPSETUP TO INITIALIZE SPAREA
* AND OPEN SERVER (TRANSACTION ROUTER SERVICE)
* IF THIS IS A RSP CALL, SPAREA IS PASSED IN THE COMMAREA.
* (DIRECTCONNECT).
* FOR TRACING, MOVE 'Y' TO SPTRCOPT
*****************************************************

     MOVE EIBCALEN TO RSPRPC-SWITCH.

     IF RPC-CALL
        EXEC CICS GETMAIN
              SET      (COMMAREA-POINTER)
              FLENGTH  (LENGTH OF LWKCOMMAREA)
        END-EXEC
        SET ADDRESS OF LWKCOMMAREA TO COMMAREA-POINTER
        CALL 'RPSETUP'               USING SPAREA
     ELSE
        SET ADDRESS OF LWKCOMMAREA TO ADDRESS OF DFHCOMMAREA.

 1050-EXIT.
      EXIT.


 1100-TEST-SQLDA.
*-------------------------------------------------------*

*****************************************************
* CALCULATE THE CORRECT SQLDA SIZE INTO "SQLDA-SIZE"

     MULTIPLY MS-SQLN BY 44            GIVING SQLDA-SIZE.
     ADD +16                           TO SQLDA-SIZE.
     MOVE SQLDA-SIZE                   TO MS-SQLDABC.

*****************************************************
* CHECK TO MAKE SURE THE CALCULATED SIZE EQUALS ACTUAL SIZE
* IF IT DOESN'T THEN A SQLDA FIELD IS MISSING OR ONE
*   OF THE SQLDA FIELDS HAS THE WRONG PICTURE SIZE.

     IF (LENGTH OF MODEL-SQLDA) NOT EQUAL SQLDA-SIZE
         MOVE 'SQLDA/SQLN SIZE IN ERROR'  TO ERROR2-TEXT2
         PERFORM 9810-ERROR-MSG           THRU 9810-EXIT
         GO TO 9999-RETURN-TO-CALLER.

 1100-EXIT.
      EXIT.
```

```
 1200-GET-STORAGE.
*-------------------------------------------------------*

*******************************************************
* ALLOCATE A BLOCK OF STORAGE TO BE USED FOR THE SQLDA
* SET POINTER VARIABLE TO ADDRESS OF ALLOCATED STORAGE
* USE FLENGTH TO ALLOCATE STORAGE ABOVE THE 16M LINE
    EXEC CICS GETMAIN
              SET      (SQLDA-POINTER)
              FLENGTH  (LENGTH OF SQLDA)
              END-EXEC.

*******************************************************
* ASSOCIATE THE LINKAGE SQLDA MASK TO THE ALLOCATED STORAGE
* BY SETTING THE MASK ADDRESS TO THE ADDRESS OF THE STORAGE
     SET ADDRESS OF SQLDA TO SQLDA-POINTER.

*******************************************************
* ALLOCATE A BLOCK OF STORAGE TO BE USED FOR THE DATA
* SET POINTER VARIABLE TO ADDRESS OF ALLOCATED STORAGE
     EXEC CICS GETMAIN
               SET(EMPLOYEE-DATA-POINTER)
               FLENGTH(LENGTH OF EMPLOYEE-DATA)
     END-EXEC.
     SET ADDRESS OF EMPLOYEE-DATA TO EMPLOYEE-DATA-POINTER.

*******************************************************
* ALLOCATE A BLOCK OF STORAGE TO BE USED FOR NULL INDICATORS
* ONLY REQUIRED FOR COLUMNS DEFINED AS ALLOWING NULLS
* SET POINTER VARIABLE TO ADDRESS OF ALLOCATED STORAGE
     EXEC CICS GETMAIN
               SET(INDICATOR-VAR-POINTER)
               FLENGTH(LENGTH OF INDICATOR-VARIABLES)
     END-EXEC.
     SET ADDRESS OF INDICATOR-VARIABLES TO INDICATOR-VAR-POINTER.

 1200-EXIT.
      EXIT.

1300-SET-ADDRESSES.
*-------------------------------------------------------*

*******************************************************
* SET THE POINTER VARIABLES IN THE LINKAGE SECTION SQLDA TO
* THE ADDRESSES OF THE DATA LOCATIONS ALSO IN THE LINKAGE
```

```
 * SECTION IE: THE DATA FIELDS IN EMPLOYEE-DATA
 *
 * THESE ADDRESSES MUST BE ADDRESSES ASSOCIATED WITH VARIABLES
 * DEFINED IN THE LINKAGE SECTION BECAUSE THE OPEN SERVER API
 * PROGRAM MUST BE ABLE TO ACCESS THIS STORAGE.
 *
 * THE MODEL-SQLDA IS MOVED TO THE SQLDA TO INITIALIZE
 * THE COLUMN TYPES AND SIZES.........
 ******************************************************
      MOVE MODEL-SQLDA TO SQLDA.

      SET SQLDATA(1)  TO ADDRESS OF  FIXED-CHAR.
      SET SQLDATA(2)  TO ADDRESS OF  DATE-OUT.
      SET SQLDATA(3)  TO ADDRESS OF  VAR-CHAR.
      SET SQLDATA(4)  TO ADDRESS OF  SMALL-INT.
      SET SQLDATA(5)  TO ADDRESS OF  PACKED-DEC.
      SET SQLDATA(6)  TO ADDRESS OF  TIME-OUT.
      SET SQLDATA(7)  TO ADDRESS OF  TIMESTAMP.
      SET SQLDATA(8)  TO ADDRESS OF  FLOAT-SGL.
      SET SQLDATA(9)  TO ADDRESS OF  FLOAT-DBL.
      SET SQLDATA(10) TO ADDRESS OF  LARGE-INT.
      SET SQLDATA(11) TO ADDRESS OF  LARGE-VAR-CHAR.

 ******************************************************
 *  SET SQLIND TO ADDRESS OF NULL INDICATOR FIELDS
 *  FOR ANY COLUMN DEFINED AS NULLABLE
 ******************************************************

       SET SQLIND(1)   TO ADDRESS OF  FIXED-CHAR-IND.
       SET SQLIND(2)   TO ADDRESS OF  DATE-OUT-IND.
       SET SQLIND(3)   TO ADDRESS OF  VAR-CHAR-IND.
       SET SQLIND(4)   TO ADDRESS OF  SMALL-INT-IND.
       SET SQLIND(5)   TO ADDRESS OF  PACKED-DEC-IND.
       SET SQLIND(6)   TO ADDRESS OF  TIME-OUT-IND.
       SET SQLIND(7)   TO ADDRESS OF  TIMESTAMP-IND.
       SET SQLIND(8)   TO ADDRESS OF  FLOAT-SGL-IND.
       SET SQLIND(9)   TO ADDRESS OF  FLOAT-DBL-IND.
       SET SQLIND(10)  TO ADDRESS OF  LARGE-INT-IND.
       SET SQLIND(11)  TO ADDRESS OF  LARGE-VCHAR-IND.

  1300-EXIT.
       EXIT.

*-------------------------------------------------------*
  1400-OPEN-OUTPUT-PIPE.
 *-------------------------------------------------------*
```

```
 *-------------------------------------------------------*
 * AN OPEN PIPE WILL SET UP THE COLUMN INFORMATION,      *
 * WHICH WILL EVENTUALLY BE SENT TO THE CLIENT.......    *
 *-------------------------------------------------------*
      MOVE 'OUTPUT'               TO SPMODE.
      MOVE 'DB2'                  TO SPFORMAT.
      SET SPSQLDA                 TO ADDRESS OF SQLDA.

      CALL 'OPENPIPE'             USING SPAREA.

      IF SPRC IS NOT EQUAL TO '000'
          MOVE WS-OPENPIPE             TO ERROR1-CALL
          PERFORM 9800-PIPE-ERROR-MSG  THRU 9800-EXIT
          GO TO 9999-RETURN-TO-CALLER.

  1400-EXIT.
       EXIT.

*-------------------------------------------------------*
  5000-PROCESS-DATA.
 *-------------------------------------------------------*

      PERFORM 5300-LOAD-A-ROW      THRU 5300-EXIT.

      PERFORM 5500-SEND-A-ROW      THRU 5500-EXIT.

      PERFORM 5400-LOAD-A-NULL-ROW THRU 5400-EXIT.

      PERFORM 5500-SEND-A-ROW      THRU 5500-EXIT.

  5000-EXIT.
       EXIT.

*-------------------------------------------------------*
  5300-LOAD-A-ROW.
 *-------------------------------------------------------*
 *-------------------------------------------------------*
 * COLUMN DATA IS HARDCODED FOR THIS EXAMPLE.            *
 *-------------------------------------------------------*
      MOVE '00100'                TO FIXED-CHAR.
      MOVE '1993-09-16'           TO DATE-OUT.
      MOVE 30                     TO VCHAR-LENGTH.
      MOVE 'A ROSE BY ANY OTHER..' TO VCHAR-DATA.
      MOVE 123                    TO SMALL-INT.
      MOVE 123.45                 TO PACKED-DEC.
```

```
      MOVE '11.35.25'              TO TIME-OUT.
      MOVE '1993-10-31:10:34:24'   TO TIMESTAMP.
      MOVE 1.00345                 TO FLOAT-SGL.
      MOVE 0.0023544               TO FLOAT-DBL.
      MOVE 1234567                 TO LARGE-INT.
      MOVE 300                     TO L-VCHAR-LENGTH.
      MOVE WS-LONG-VARCHAR-TEXT    TO L-VCHAR-DATA.

 ************************************************
 *  MOVE ZERO TO NULL INDICATOR FIELDS TO INDICATE NOT NULL

      MOVE 0                      TO  FIXED-CHAR-IND.
      MOVE 0                      TO  DATE-OUT-IND.
      MOVE 0                      TO  VAR-CHAR-IND.
      MOVE 0                      TO  SMALL-INT-IND.
      MOVE 0                      TO  PACKED-DEC-IND.
      MOVE 0                      TO  TIME-OUT-IND.
      MOVE 0                      TO  TIMESTAMP-IND.
      MOVE 0                      TO  FLOAT-SGL-IND.
      MOVE 0                      TO  FLOAT-DBL-IND.
      MOVE 0                      TO  LARGE-INT-IND.
      MOVE 0                      TO  LARGE-VCHAR-IND.

  5300-EXIT.
       EXIT.

*-------------------------------------------------------*
  5400-LOAD-A-NULL-ROW.
 *-------------------------------------------------------*

 ************************************************
 *  MOVE -1 TO NULL INDICATOR FIELDS TO INDICATE NULL
 *    LEFTOVER DATA IN DATA FIELDS WILL BE IGNORED

      MOVE -1                     TO  FIXED-CHAR-IND.
      MOVE -1                     TO  DATE-OUT-IND.
      MOVE -1                     TO  VAR-CHAR-IND.
      MOVE -1                     TO  SMALL-INT-IND.
      MOVE -1                     TO  PACKED-DEC-IND.
      MOVE -1                     TO  TIME-OUT-IND.
      MOVE -1                     TO  TIMESTAMP-IND.
      MOVE -1                     TO  FLOAT-SGL-IND.
      MOVE -1                     TO  FLOAT-DBL-IND.
      MOVE -1                     TO  LARGE-INT-IND.
      MOVE -1                     TO  LARGE-VCHAR-IND.
```

```
  5400-EXIT.
       EXIT.

*-------------------------------------------------------*
  5500-SEND-A-ROW.
 *-------------------------------------------------------*

*-------------------------------------------------------*
 * PUTPIPE SENDS A RESULT ROW TO THE OUTPUT BUFFER, WHICH*
 * WILL EVENTUALLY BE SENT DOWN TO THE CLIENT....        *
 *-------------------------------------------------------*

      CALL 'PUTPIPE'               USING SPAREA.

      IF SPRC IS NOT EQUAL TO '000'
          MOVE WS-PUTPIPE          TO ERROR1-CALL
          PERFORM 9800-PIPE-ERROR-MSG THRU 9800-EXIT
          GO TO 9999-RETURN-TO-CALLER.

  5500-EXIT.
       EXIT.

*-------------------------------------------------------*
  9000-WRAP-UP.
 *-------------------------------------------------------*

      PERFORM 9200-CLOSE-PIPE      THRU 9200-EXIT.

      PERFORM 9900-ALL-DONE        THRU 9900-EXIT.

 *    IF THIS IS AN RPC CALL, PERFORM OPEN SERVER CLOSE
      IF RPC-CALL
         PERFORM 9950-RPDONE          THRU 9950-EXIT.

  9000-EXIT.
       EXIT.

*-------------------------------------------------------*
  9200-CLOSE-PIPE.
 *-------------------------------------------------------*
 *-------------------------------------------------------*
 *CLOSEPIPE IS LIKE CLOSING A FILE,  PLACES AN EOF MARKER*
 *-------------------------------------------------------*

      CALL 'CLOSPIPE'              USING SPAREA.
```

```
      IF SPRC IS NOT EQUAL TO '000'
          MOVE WS-CLOSPIPE         TO ERROR1-CALL
          PERFORM 9800-PIPE-ERROR-MSG THRU 9800-EXIT
          GO TO 9999-RETURN-TO-CALLER.

  9200-EXIT.
      EXIT.

*-------------------------------------------------------*
  9800-PIPE-ERROR-MSG.
 *-------------------------------------------------------*

 ********************************************************
 *  IF NO ERRORS, MOVE 'OK' TO SPSTATUS BEFORE CALLING MESSAGE.
 *  IF ERRORS, MOVE 'E' TO SPSTATUS.
 *  EITHER WAY MOVE A MESSAGE UP TO A 100 CHAR INTO SPMSG
  ********************************************************

 *-------------------------------------------------------*
 * MESSAGE WILL WRITE THE 100 BYTE SPMSG TO A MSG BUFFER,*
 * WHICH WILL EVENTUALLY BE WRITTEN TO THE CLIENT...     *
 *-------------------------------------------------------*
      MOVE SPRC                    TO ERROR1-SPRC.
      MOVE ERROR1-MSG              TO SPMSG.
      MOVE 'E'                     TO SPSTATUS.

      CALL 'MESSAGE' USING SPAREA.

 9800-EXIT.
       EXIT.

*-------------------------------------------------------*
  9810-ERROR-MSG.
 *-------------------------------------------------------*

 *-------------------------------------------------------*
 * MESSAGE WILL WRITE THE 100 BYTE SPMSG TO A MSG BUFFER,*
 * WHICH WILL EVENTUALLY BE WRITTEN TO THE CLIENT...     *
 *-------------------------------------------------------*

      MOVE ERROR2-MSG              TO SPMSG.
      MOVE 'E'                     TO SPSTATUS.

      CALL 'MESSAGE' USING SPAREA.

  9810-EXIT.
```

```
       EXIT.

*-------------------------------------------------------*
  9900-ALL-DONE.
 *-------------------------------------------------------*

 ***********************************************************
 *  IF NO ERRORS, MOVE 'OK' TO SPSTATUS BEFORE CALLING STATUS*
 *  IF ERRORS, MOVE 'E' TO SPSTATUS BEFORE CALLING STATUS    *
 *    CAN MOVE UP TO 8 CHARS INTO SPCODE (SPMSG IS IGNORED)  *
 *    BUT EITHER WAY ALWAYS CALL STATUS AFTER CLOSPIPE       *
 *    CALLING STATUS WILL AUTOMATIC CLOSE ANY OPEN PIPES     *
 *                                                           *
 *    CALLING STATUS WILL ALSO FLUSH ANY RESULTS AND/OR      *
 *    MESSAGES FROM THE BUFFERS, TO THE CLIENT               *
 ***********************************************************

      MOVE 'OK'                         TO SPSTATUS.
      CALL 'STATUS' USING SPAREA.

  9900-EXIT.
       EXIT.

*-------------------------------------------------------*
  9950-RPDONE.
 *-------------------------------------------------------*

 *******************************************************
 *    CLOSE OPEN SERVER
 *    IF THIS IS AN RPC CALL, PERFORM OPEN SERVER CLOSE
 *******************************************************
      CALL 'RPDONE' USING SPAREA.

  9950-EXIT.
       EXIT.

*-------------------------------------------------------*
 9999-RETURN-TO-CALLER.
 *-------------------------------------------------------*

 *****************************************************
 *  FOR EMERGENCY BAIL-OUT

      CALL 'RPDONE' USING SPAREA.

      EXEC CICS
```

```
           RETURN
           END-EXEC.

 9999-EXIT.
      EXIT.
```

APPENDIX C

RSP3C STD Input and Output Pipe Sample RSP

If you want to write an RSP to send single-column rows of character strings, review the RSP3C sample RSP. RSP3C illustrates how to use input and output data pipes in STD format to echo data records sent to it from the client application. Recall that with STD format data pipes, the data is transmitted as one VARCHAR column.

This appendix discusses the following topics:

- Using the SPAREA with RSP3C
- Specifying error handling
- Client application processing
- RSP3C STD input and output pipe sample code

Using the SPAREA with RSP3C

The SPAREA is the storage area used to pass information between the RSP and Open ServerConnect.

In the following code fragment, the DFHCOMMAREA is the Open ServerConnect communication area. SPAREAC is the COBOL COPY definition.

```
01  DFHCOMMAREA.
    COPY SPAREAC.
```

SPMAXLEN and SPRECLEN

RSP3C uses the SPAREA to pass information about the type of data pipe to MainframeConnect.

```
MOVE 'INPUT'              TO SPMODE.
 MOVE 'STD'                TO SPFORMAT.
 MOVE 55                   TO SPMAXLEN.
 CALL 'OPENPIPE' USING SPAREA.
```

In this example, the type and format of the pipe are specified using the SPAREA SPMODE and SPFORMAT fields. Because the exact length of the record is not known, a maximum record length is specified with SPMAXLEN.

In the following example, you can see that because you already set the maximum input record size with SPMAXLEN and the OPENPIPE command, you do not need to reset SPRECLEN for each GETPIPE command. MainframeConnect determines the size of the input record and sets SPRECLEN accordingly.

```
SET SPINTO TO ADDRESS OF WS-INPUT-REC
 CALL 'GETPIPE' USING SPAREA
```

In the following example, RSP3C uses SPRECLEN with a PUTPIPE command to pass the length of an output record to MainframeConnect.

```
SET SPFROM TO ADDRESS OF WS-INPUT-REC
 MOVE 55 TO SPRECLEN
 CALL 'PUTPIPE' USING SPAREA
```

The following table describes these SPAREA fields in RSP3C and explains how they are used.

Table C-1: SPAREA fields describing records

SPAREA Field	Use
SPMODE	Specifies the mode of the data pipe. Valid values are 'INPUT' or 'OUTPUT'.
SPFORMAT	Specifies the format of the data to be transmitted through the pipe. Valid values are: • 'DB2' (only for output pipes) • 'STD' • 'BIN'
SPMAXLEN	Specifies the maximum record length of records transmitted through a STD or BIN format pipe. **Note** For DB2 or STD format pipes, you provide maximum record length information in the SQLDA.
SPRECLEN	Specifies the length of a particular record transmitted through a STD or BIN format pipe. For output pipes, the RSP sets this value; for input pipes, MainframeConnect sets this value.

Note You must specify either SPMAXLEN or SPRECLEN, which defines the actual length of a particular data record.

SPINTO and SPFROM

The following sample shows how to use the SPINTO field.

```
SET SPINTO TO ADDRESS OF WS-INPUT-REC
 CALL 'GETPIPE' USING SPAREA
```

Use the SPINTO field to specify the address of the storage location where the RSP places the input data it receives from the client application. The SPINTO field is used with the GETPIPE command, which reads client application data from an input pipe.

In RSP3C, the input and output storage area are defined as follows:

- A GETMAIN is issued to allocate this storage area
- A pointer was set to the area
- The *WS-INPUT-REC* variable is associated with that pointer, as shown:

```
EXEC CICS
 GETMAIN SET(PARTSPOINTER)
    FLENGTH(55)
 END-EXEC.
SET ADDRESS OF WS-INPUT-REC TO PARTSPOINTER.
```

RSP3C uses a corresponding field, SPFROM, to specify the address of storage where the RSP places the data it is returning with the PUTPIPE command. The PUTPIPE command returns data to the client application through an output pipe.

```
SET SPFROM TO ADDRESS OF WS-INPUT-REC
MOVE 55 TO SPRECLEN
CALL 'PUTPIPE' USING SPAREA
```

Again, the storage is defined within the RSP.

Note You must specify SPINTO for input pipes.

Specifying error handling

RSP3C handles status and messages the same way MODELRSP does. It uses three SPAREA fields to communicate status and messages to MainframeConnect: SPRC, SPSTATUS, and SPMSG. See “SPAREA definitions” on page 142 for a description of how they are used.

Note Your code should always check the SPRC field after issuing any RSP command. See “Specifying error handling” on page 32 for more information on error handling.

Client application processing

RSP3C uses both input and output data pipes in STD format to transmit data to and from the client application. It includes a sample of the ISQL you might use to call it.

You can use STD format input and output pipes to transmit data when you have mirror applications on the host and on the LAN. If both programs contain the same data definitions, or if only one column is returned, the additional data structure information that would come from a SQLDA definition is not needed.

The statement that can invoke RSP3C from the client application is shown in the next subsection, followed by the results echoed back to the client application. RSP3C requires at least one data record. This program reads standard input records of up to 55 characters in length. It allows any number of rows to be sent and returned.

Invoking from the client application (ISQL)

The following ISQL invokes RSP3C:

```
C:\DIRECTCONNECT>> isql -Sdcservice -Uuserid
1 USE PROCEDURE WITH DATA RSP3C ;
2 THIS IS THE FIRST STRING OF DATA
3 AND THIS IS THE SECOND RECORD OF DATA
4 AND THIS IS THE THIRD AND SO ON
5 ;
6 GO
```

The USE PROCEDURE statement includes a WITH DATA clause preceding the RSP name. WITH DATA indicates that ISQL should send the ASCII format data following the USE PROCEDURE statement to the RSP.

Returning results to the client application

RSP3C returns the following results to the client.

```
COLUMN01
 ------------------------------------------------------
  **-- THE FOLLOWING IS A LIST OF THE DATA RECORDS SENT.
 REC#- 01:THIS IS THE FIRST STRING OF DATA
 REC#- 02:AND THIS IS THE SECOND RECORD OF DATA
 REC#- 03:AND THIS IS THE THIRD AND SO ON

 (4 rows affected)
 1 QUIT

 C:\DIRECTCONNECT>>
```

RSP3C STD input and output pipe sample code

```
 IDENTIFICATION DIVISION.
  PROGRAM-ID.  RSP3C.
 ****************************************************************
 *  RSP3C - STD INPUT PIPES PROCEDURE                           *
 *                                                              *
 *  THIS SAMPLE STORED PROCEDURE WAS WRITTEN TO USE A "STD" INPUT *
 *  AND OUTPUT PIPE FOR ILLUSTRATION.  IT REQUIRES AT LEAST ONE   *
 *  DATA RECORD TO BE PASSED TO IT WHEN INVOKED.                *
 *  AN EXAMPLE OF INVOKING IT:                                  *
 *                                                              *
 *  USE PROCEDURE WITH DATA RSP3C ;                             *
 *  THIS IS THE FIRST AND ONLY DATA RECORD                      *
 *  ;                                                           *
 *                                                              *
 *  DATA RECORDS ARE SET FOR UP TO 55 CHARS IN LENGTH.  ALL     *
 *  DATA RECORDS WILL BE RETURNED THROUGH THE OUTPUT PIPE AS    *
 *  VERIFICATION.                                               *
 *                                                              *
 ****************************************************************

  ENVIRONMENT DIVISION.

  DATA DIVISION.

  WORKING-STORAGE SECTION.

*****************************************************************
 * ONE POINTER IS USED FOR BOTH INPUT AND OUTPUT RECORD AREA
 *   IN THIS CASE BECAUSE THE RECORDS WILL BE THE SAME LENGTH.
 ****************************************************************
  01  SAMPLE-POINTER.
      10 PARTSPOINTER             USAGE IS POINTER.

*****************************************************************
 * SWITCHES FOR RECORD PROCESSING CONTROL.                      *
 ****************************************************************
  01  WS-SWITCHES.
      10 WS-MORE-RECORDS-IN-SW        PIC X(01) VALUE 'Y'.
        88 MORE-RECORDS-IN           VALUE 'Y'.
         88 NO-MORE-RECORDS-IN        VALUE 'N'.

      10 WS-ERROR-HAPPENED-SW         PIC X(01) VALUE 'N'.
         88 ERROR-HAPPENED            VALUE 'Y'.
         88 NO-ERROR-YET              VALUE 'N'.
```

```
*****************************************************************
 * A NUMBER FOR INCRIMENTING.                                    *
 *****************************************************************
 01  WS-VARIABLES.
      05  WS-INCRINUM                 PIC 99  VALUE ZEROES.
      05  INREC-CTR                   PIC S9(8) COMP VALUE 0.
      05  WS-DIS-NUM                  PIC 9(4) VALUE ZEROES.

  01 MESSAGES.
     05  ERROR1-MSG.
         07  ERROR1-TEXT1             PIC X(19)  VALUE
             'ERROR WITH CALL TO '.
         07  ERROR1-CALL              PIC X(10)  VALUE SPACES.
         07  ERROR1-TEXT2             PIC X(14)  VALUE
             ' - SPRC CODE: '.
         07  ERROR1-SPRC              PIC X(03)  VALUE SPACES.

*****************************************************************
 * OUTPUT RECORD DESCRIPTIONS.                                   *
 *****************************************************************
  01  WS-OUTPUT-REC.
      10 WS-OUT-MSG-AREA.
         15 FILLER                    PIC X(07) VALUE 'REC#-> '.
         15 WS-OUT-MSG-NUM            PIC X(02) VALUE SPACES.
         15 FILLER                    PIC X(01) VALUE ':'.
      10 WS-OUT-SOME-DATA             PIC X(45) VALUE SPACES.

  01  WS-OUT-DATA-MSG.
      10 FILLER                       PIC X(55) VALUE
      '**--> THE FOLLOWING IS A LIST OF THE DATA RECORDS SENT.'.

 *  THIS  SWITCH IS USED FOR TESTING IF RPC CALL
  77  RSPRPC-SWITCH   PIC S9(4) COMP VALUE 0.
      88 RPC-CALL                    VALUE 0.

  01  COMMAREA-POINTER               USAGE IS POINTER.

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

  01  DFHCOMMAREA.
      05  NOT-USED                 PIC X(1).
```

```
*****************************************************************
*  THIS IS THE ACTUAL SPAREA POINTER AND DEFINITION             *
*****************************************************************
 01  LWKCOMMAREA.
     COPY SPAREAC.

************************************************************
* THIS AREA IS USED FOR BOTH INPUT AND OUTPUT BECAUSE BOTH
* TYPES OF RECORDS ARE THE SAME LENGTH IN THIS CASE.
************************************************************
 01  WS-INPUT-REC.
     10 WS-INPUT-DATA.
        15  WS-INPUT-1ST-5          PIC X(05).
        15  FILLER                  PIC X(40).
     10 WS-INPUT-REST               PIC X(10).

 PROCEDURE DIVISION.

 000-MAIN-PROCESSING.

     PERFORM 100-INITIALIZE           THRU 100-EXIT.

     IF NO-ERROR-YET
         PERFORM 500-PROCESS-I-O      THRU 500-EXIT.

     PERFORM 900-WRAP-UP              THRU 900-EXIT.

     EXEC CICS
          RETURN
     END-EXEC.

     GOBACK.

 000-EXIT.
     EXIT.

100-INITIALIZE.
*****************************************************
* IF THIS IS A RPC CALL, CALL RPSETUP TO INITIALIZE SPAREA
* AND OPEN SERVER (TRANSACTION ROUTER SERVICE)
* IF THIS IS A RSP CALL, SPAREA IS PASSED IN THE COMMAREA.
* (DIRECTCONNECT).
* FOR TRACING, MOVE 'Y' TO SPTRCOPT
*****************************************************
```

```
     MOVE EIBCALEN TO RSPRPC-SWITCH.

     IF RPC-CALL
        EXEC CICS GETMAIN
              SET      (COMMAREA-POINTER)
              FLENGTH  (LENGTH OF LWKCOMMAREA)
        END-EXEC
        SET ADDRESS OF LWKCOMMAREA TO COMMAREA-POINTER
        CALL 'RPSETUP'               USING SPAREA
     ELSE
        SET ADDRESS OF LWKCOMMAREA TO ADDRESS OF DFHCOMMAREA.

     MOVE 'OK'                       TO SPSTATUS.
     SET MORE-RECORDS-IN             TO TRUE.

*****************************************************************
* ALLOCATE A BLOCK OF STORAGE TO BE USED FOR THE DATA
* SET POINTER VARIABLE TO ADDRESS OF ALLOCATED STORAGE
*****************************************************************

     EXEC CICS
       GETMAIN SET(PARTSPOINTER)
               FLENGTH(55)
     END-EXEC.
     SET ADDRESS OF WS-INPUT-REC       TO PARTSPOINTER.

     PERFORM 110-OPEN-INPUT-PIPE       THRU 110-EXIT.

     IF NO-ERROR-YET
         PERFORM 120-OPEN-OUTPUT-PIPE  THRU 120-EXIT.

 100-EXIT.
     EXIT.

110-OPEN-INPUT-PIPE.
*****************************************************************
* OPEN THE INPUT PIPE.                                          *
*****************************************************************
     MOVE 'INPUT'                    TO SPMODE.
     MOVE 'STD'                      TO SPFORMAT.
     MOVE 55                         TO SPMAXLEN.
     CALL 'OPENPIPE' USING SPAREA.

*****************************************************************
* IF OPEN FAILED, THEN ISSUE AN ERROR MESSAGE.                  *
*****************************************************************
```

```
     IF SPRC NOT = '000'
         SET ERROR-HAPPENED            TO TRUE
         MOVE 'OPENPIPE'               TO ERROR1-CALL
         PERFORM 800-DO-MESSAGE        THRU 800-EXIT.

 110-EXIT.
     EXIT.

120-OPEN-OUTPUT-PIPE.
*********************************************************************
*  AFTER A SUCCESSFUL OPENPIPE FOR OUTPUT: HEADER, TABLE, AND
*    COLUMN IXF RECORDS ARE GENERATED AND SENT TO APPC.
*********************************************************************
     MOVE 'OUTPUT'                     TO SPMODE.
     MOVE 'STD'                        TO SPFORMAT.
     MOVE 55                           TO SPMAXLEN.

     CALL 'OPENPIPE' USING SPAREA.

*********************************************************************
* IF OPEN FAILED, THEN ISSUE AN ERROR MESSAGE.                      *
*********************************************************************
     IF SPRC NOT = '000'
         SET ERROR-HAPPENED            TO TRUE
         MOVE 'OPENPIPE'               TO ERROR1-CALL
         PERFORM 800-DO-MESSAGE        THRU 800-EXIT.

 120-EXIT.
     EXIT.

500-PROCESS-I-O.

     MOVE 0                            TO WS-INCRINUM.

     PERFORM 510-SEND-RECORDS-HEADING  THRU 510-EXIT.

     IF NO-ERROR-YET
         PERFORM 540-PROCESS-DATA-RECS THRU 540-EXIT
         UNTIL NO-MORE-RECORDS-IN.

 500-EXIT.
     EXIT.

510-SEND-RECORDS-HEADING.

     MOVE WS-OUT-DATA-MSG              TO WS-INPUT-REC.
```

```
      SET SPFROM TO ADDRESS OF WS-INPUT-REC.

 *-------------------------------------------------------*
 * PUTPIPE SENDS A RESULT ROW TO THE OUTPUT BUFFER, WHICH*
 * WILL EVENTUALLY BE SENT DOWN TO THE CLIENT APPLICATION*
 *-------------------------------------------------------*
      CALL 'PUTPIPE' USING SPAREA.

      IF SPRC NOT = '000'
          SET ERROR-HAPPENED            TO TRUE
          MOVE 'PUTPIPE '               TO ERROR1-CALL
          PERFORM 800-DO-MESSAGE        THRU 800-EXIT.

  510-EXIT.
      EXIT.

 540-PROCESS-DATA-RECS.
 ****************************************************************
 * OBTAIN THE DATA RECORDS SENT WITH PROGRAM AND SEND BACK TO PIPE*
 ****************************************************************

      IF NO-ERROR-YET
          PERFORM 542-READ-RECORDS      THRU 542-EXIT.

      IF NO-ERROR-YET
      AND MORE-RECORDS-IN
          PERFORM 544-WRITE-RECORDS     THRU 544-EXIT.

  540-EXIT.
      EXIT.

 542-READ-RECORDS.
 ****************************************************************
 * READ AN INPUT RECORD THROUGH THE INPUT PIPE                  *
 * NOTE THAT THE SPRECLEN DOESN'T NEED TO BE SET BECAUSE THE    *
 * MAINFRAMECONNECT SETS THIS FIELD WHEN IT SENDS THE INPUT RECORD.
****************************************************************

      ADD 1                             TO INREC-CTR
      SET SPINTO TO ADDRESS OF WS-INPUT-REC.
      CALL 'GETPIPE' USING SPAREA.

      EVALUATE SPRC
         WHEN '000' CONTINUE
         WHEN 'EOF' SET NO-MORE-RECORDS-IN TO TRUE
         WHEN OTHER PERFORM
```

```
            SET NO-MORE-RECORDS-IN     TO TRUE
            SET ERROR-HAPPENED         TO TRUE
            MOVE 'GETPIPE '            TO ERROR1-CALL
            PERFORM 800-DO-MESSAGE     THRU 800-EXIT
         END-PERFORM
      END-EVALUATE.

 ****************************************************************
 * THIS IS JUST TO PREVENT ACCIDENTAL RUNAWAY.
****************************************************************
      IF WS-INPUT-1ST-5 = SPACES
      OR INREC-CTR > 500
          SET NO-MORE-RECORDS-IN        TO TRUE
          SET ERROR-HAPPENED            TO TRUE
          MOVE 'RUNAWAY '               TO ERROR1-CALL
          PERFORM 800-DO-MESSAGE        THRU 800-EXIT
      END-IF.

  542-EXIT.
      EXIT.

 544-WRITE-RECORDS.
 ****************************************************************
 * REFORMAT THE INPUT RECORD AND SEND BACK DOWN THE OUTPUT PIPE   *
 * NOTE THAT SPRECLEN IS RESET TO 55 EACH TIME BECAUSE THE VALUE  *
 * MIGHT BE CHANGED BY THE PREVIOUS GETPIPE.                      *
 ****************************************************************

      ADD 1                             TO WS-INCRINUM.
      MOVE WS-INCRINUM                  TO WS-OUT-MSG-NUM.
 *    MOVE WS-INPUT-DATA                TO WS-OUT-SOME-DATA.
      MOVE SPACES                       TO WS-OUT-SOME-DATA.
      MOVE WS-INPUT-DATA (1:SPRECLEN)   TO WS-OUT-SOME-DATA.
      MOVE WS-OUTPUT-REC                TO WS-INPUT-REC.
      MOVE 55                           TO SPRECLEN.
      SET SPFROM TO ADDRESS OF WS-INPUT-REC.

 *-------------------------------------------------------*
 * PUTPIPE SENDS A RESULT ROW TO THE OUTPUT BUFFER, WHICH*
 * WILL EVENTUALLY BE SENT DOWN TO THE CLIENT APPLICATION*
 *-------------------------------------------------------*
      CALL 'PUTPIPE' USING SPAREA.

      IF SPRC NOT = '000'
          SET NO-MORE-RECORDS-IN        TO TRUE
          SET ERROR-HAPPENED            TO TRUE
```

```
         MOVE 'PUTPIPE '               TO ERROR1-CALL
         PERFORM 800-DO-MESSAGE        THRU 800-EXIT.

 544-EXIT.
     EXIT.

800-DO-MESSAGE.
*****************************************************************
* SOMETHING FAILED, SO ISSUE AN ERROR MESSAGE AND GET OUT.       *
*****************************************************************
     MOVE SPRC                         TO ERROR1-SPRC.
     MOVE ERROR1-MSG                   TO SPMSG.
     MOVE 'E'                          TO SPSTATUS.

*-------------------------------------------------------*
* MESSAGE WILL WRITE THE 100 BYTE SPMSG TO A MSG BUFFER,*
* WHICH WILL EVENTUALLY BE WRITTEN TO THE CLIENT        *
* APPLICATION                                           *
*-------------------------------------------------------*
     CALL 'MESSAGE' USING SPAREA.

     IF SPRC NOT = '000'
         SET NO-MORE-RECORDS-IN        TO TRUE
         SET ERROR-HAPPENED            TO TRUE.

 800-EXIT.
     EXIT.

900-WRAP-UP.
*****************************************************************
* CLOSE PIPES - ISSUE STATUS.                                   *
*****************************************************************

*-------------------------------------------------------*
*CLOSEPIPE IS LIKE CLOSING A FILE,  PLACES AN EOF MARKER*
*-------------------------------------------------------*
     IF NO-ERROR-YET
         MOVE 'INPUT'                  TO SPMODE
         CALL 'CLOSPIPE' USING SPAREA
         IF SPRC NOT = '000'
             SET ERROR-HAPPENED        TO TRUE
             MOVE 'CLOSPIPE'           TO ERROR1-CALL
             PERFORM 800-DO-MESSAGE    THRU 800-EXIT.

     IF NO-ERROR-YET
         MOVE 'OUTPUT'                 TO SPMODE
```

```
         CALL 'CLOSPIPE' USING SPAREA
         IF SPRC NOT = '000'
             SET ERROR-HAPPENED        TO TRUE
             MOVE 'CLOSPIPE'           TO ERROR1-CALL
             PERFORM 800-DO-MESSAGE    THRU 800-EXIT
         END-IF
     END-IF.

     IF NO-ERROR-YET
         MOVE 'OK'                     TO SPSTATUS
 *-------------------------------------------------------------*
 *    CALLING STATUS WILL FLUSH ANY RESULTS AND/OR             *
 *    MESSAGES FROM THE BUFFERS, TO THE CLIENT APPLICATION     *
 *-------------------------------------------------------------*
         CALL 'STATUS'   USING SPAREA
         IF SPRC NOT = '000'
             SET ERROR-HAPPENED        TO TRUE
             MOVE 'STATUS  '           TO ERROR1-CALL
             PERFORM 800-DO-MESSAGE    THRU 800-EXIT
         END-IF
     ELSE
         MOVE 'E'                      TO SPSTATUS
         MOVE 'MYERCODE'               TO SPCODE
         CALL 'STATUS'   USING SPAREA
     END-IF.

 ********************************************************
 *    CLOSE OPEN SERVER
 *    IF THIS IS AN RPC CALL, PERFORM OPEN SERVER CLOSE
 ********************************************************
     IF RPC-CALL
        CALL 'RPDONE' USING SPAREA.

  900-EXIT.
     EXIT.
```

APPENDIX D

RSP4C Keyword Variable Sample RSP

If you want to pass keyword values, use sample RSP4C. RSP4C is an RSP that reads up to 50 keywords and echoes them to a client application through a STD format output pipe. It also includes code that allows you to control whether messages and return codes return as output. The examples in this section illustrate its capabilities.

This appendix discusses the following topics:

- Client application processing
- Sample input and results
- RSP4C error handling
- Keyword sample code fragment
- RSP4C keyword variable sample code

Client application processing

The RSP4C sample RSP is written to receive keywords that are up to 15 characters in length (including the &) and keyword values up to 28 characters in length. All keywords and their values are returned to the client application through a STD format output pipe for display.

For display purposes only, RSP4C overwrites the rightmost five characters (positions 24–28) of the keyword values with the length of the values (determined by Open ServerConnect or MainframeConnect) and sends them to the RSP through the keyword variable table. RSP4C does not corrupt the actual data.

Sample input and results

Figure D-1 on page 110 shows an example of a file used as input to ISQL.EXE to send keywords and values to an RSP program named RSP4C. Figure D-2 on page 111 shows an example of the echoed input.

You can use input and output files in your ISQL command. This example uses *RSP4C.SQL* as the input file and *RSP4C.LOG* as the output file:

```
ISQL -SDB2T -Uxxxxxxxx -Pyyyyyyyy -iRSP4C.SQL -oRSP4C.LOG
```

RSP4C.SQL sample input

The following figure illustrates the use of keyword variables.

Figure D-1: RSP4C.SQL

```
=====================================================================
C:\DIRECTCONNECT>> isql -Sdcservice -Uuserid
USE PROCEDURE RSP4C &KEY1='A Test of keywords' &KEY2=Another test
&KEY3="SO?"
GO
=====================================================================
```

The RSP accepts a text string and converts it to uppercase for processing.

To process text strings with embedded blanks, mixed-case, or special characters, enclose them within delimiters. The value passed in *&KEY2* is counted only to the blank and is only partially echoed. The value passed in *&KEY1* is enclosed in single quotes, while the value passed in *&KEY3* is enclosed in double quotes.

RSP4C.LOG sample results

RSP4C.LOG, the following figure, contains the results the client application receives after invoking RSP4C:

Figure D-2: RSP4C.LOG

```
================================================================
1 2 1 2
 COLUMN01
 -----------------------------------------------------
 **-- THE FOLLOWING IS A LIST OF THE KEYWORDS SENT.
 KEYW- 01:&KEY1          = 'A Test of keywords'    0020
 KEYW- 02:&KEY2          = ANOTHER                 0007
 KEYW- 03:&KEY3          = 'SO?'                   0005

(4 rows affected)
1
================================================================
```

You can see that *&KEY2*, input as *ANOTHER*, is counted only to the blank.

RSP4C error handling

The examples in this section demonstrate how the sample RSP suppresses the error code or the text of the error message.

No error code

The code in the following figure passes &ERRORMSG= to ERROR-CHECK.

Figure D-3: ERRORMSG example

```
================================================================
C:\DIRECTCONNECT>> isql -Sdcservice -Uuserid
USE PROCEDURE RSP4C &ERRORMSG=TESTIT
GO
1 2
================================================================
```

The following figure contains the results that the client application receives:

Figure D-4: ERRORMSG response

```
=====================================================================
RSP_STD_PIPE
 ------------------------------------------------------

**-- THE FOLLOWING IS A LIST OF THE KEYWORDS SENT.
 KEYW- 01:&ERRORMSG      = TESTIT                  0006

 (2 rows affected)

 THIS IS YOUR ERROR MESSAGE TEXT.

 RSP Completion Code=152183236

=====================================================================
```

The RSP code does not set

```
SPSTATUS='E'
```

and so does not pass a value through the SPRC field. As a result, the "DG21002: Result failed. Database server error code" message does not display an error code.

No message

The code in the following figure passes &STATUSMSG= to STATUS-CHECK.

Figure D-5: STATUSMSG example

```
=====================================================================
USE PROCEDURE RSP4C &STATUSMSG=YES
GO
1 2
=====================================================================
```

The following figure contains the results that the client application receives:

Figure D-6: STATUSMSG response

```
=====================================================================
RSP_STD_PIPE
 ----------------------------------------------------
 **-- THE FOLLOWING IS A LIST OF THE KEYWORDS SENT.
 KEYW- 01:&STATUSMSG     = YES                    0003




RSP Completion Code=152183220
=====================================================================
```

RSP4C's paragraph 522-SEND-KEYWORD-HEADING on writes the following:

```
**-- THE FOLLOWING IS A LIST OF THE KEYWORDS SENT.
```

In RSP4C's paragraph 524-READ-WRITE-KEYWORDS on, however, STATUS-CHECK sets the ERROR-HAPPENED switch.

Keyword sample code fragment

The following COBOL II code fragment shows one way to code an RSP to handle keyword variables.

```
LINKAGE SECTION.
 01  DFHCOMMAREA.
       COPY SPAREAC.
 ****************************************************************
 * LINKAGE TO CALLING PROGRAM                                   *
 ****************************************************************

 01  KEYWORD-VTABLE.
       10 VTABLE-SIZE                  PIC S9(8) COMP.
       10 VTABLE-ENTRY OCCURS 0 TO 50 TIMES
             DEPENDING ON VTABLE-SIZE
             INDEXED BY VTABLE-INDEX.
       15 VTABLE-NAME               USAGE IS POINTER.
       15 VTABLE-VALUE              USAGE IS POINTER.
       15 VTABLE-NAME-LENGTH        PIC S9(4) COMP.
```

```
      15 VTABLE-VALUE-LENGTH       PIC S9(4) COMP.

 01  TABLE-NAME                     PIC X(15).
 01  TABLE-VALUE                    PIC X(28).

 01  WS-INPUT-REC.
       10 WS-INPUT-DATA               PIC X(45).
       10 WS-INPUT-REST               PIC X(10).
                 :
                 :
                 :
 ****************************************************************
 * MAKE SURE AT LEAST ONE KEYWORD WAS SENT ALONG WITH PROGRAM     *
 ****************************************************************

 MOVE 0                          TO WS-INCRINUM.

 IF SPVARTAB = NULL
       PERFORM 700-LOAD-KEYWORD-ERROR   THRU 700-EXIT
       GO TO 510-EXIT.

 IF VTABLE-SIZE NOT > 0
       PERFORM 700-LOAD-KEYWORD-ERROR   THRU 700-EXIT
       GO TO 510-EXIT.

 SET ADDRESS OF KEYWORD-VTABLE   TO SPVARTAB.
```

RSP4C keyword variable sample code

RSP4C is an example of a COBOL II RSP written to handle keyword variables sent to it from the client application.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID.  RSP4C.
 ****************************************************************
 *  RSP4C - DOCTORED STORED PROCEDURE                           *
 *                                                              *
 *  THIS SAMPLE STORED PROCEDURE WAS WRITTEN TO USE A "STD"     *
 *  OUTPUT PIPE AND KEYWORDS FOR ILLUSTRATION.  IT REQUIRES AT  *
 *  LEAST ONE KEYWORD/VALUE BE PASSED TO IT WHEN INVOKED.
 *
 *                                                              *
 *  USE PROCEDURE RSP4C &FIRSTKEYWORD=FIRSTVALUE ;              *
```

```
*                                                                *
*  THIS PROGRAM IS SET UP TO ACCEPT KEYWORDS OF UP TO 15 CHARS   *
*  IN LENGTH AND UP TO 28 CHARS FOR THE KEYWORD VALUES.  ALL     *
*  KEYWORDS, KEYWORD VALUES, WILL BE RETURNED                    *
*  THROUGH THE OUTPUT PIPE AS VERIFICATION.                      *
*                                                                *
*  ALSO:  2 SPECIAL KEYWORDS ARE SET UP TO TEST ERROR MESSAGING  *
*  THE ERROR CONDITIONS SEND 'E' TO SPSTATUS                     *
*   - ONE USING "MESSAGE" AND ONE USING "STATUS".                *
*  &ERRORMSG  : 'E' TO SPSTATUS, MSG TO SPMSG, CALLS 'MESSAGE'   *
*  &MESSAGE   : 'OK'TO SPSTATUS, MSG TO SPMSG, CALLS 'MESSAGE'   *
*  &STATUSMSG : 'E' TO SPSTATUS, MSG TO SPCODE, CALLS 'STATUS'   *
*  &STATNEMSG : 'OK'TO SPSTATUS, MSG TO SPCODE, CALLS 'STATUS'   *
*                                                                *
******************************************************************

ENVIRONMENT DIVISION.

DATA DIVISION.

WORKING-STORAGE SECTION.

******************************************************************
* POINTERS TO INPUT AND OUTPUT RECORD AREA.                      *
******************************************************************
 01  WS-POINTERS.
     10 WS-OUTPUT-POINTER        USAGE IS POINTER.

******************************************************************
* SWITCHES FOR RECORD PROCESSING CONTROL.                        *
******************************************************************
 01  WS-SWITCHES.
     10 WS-ERROR-MSG-SW              PIC X(01) VALUE 'N'.
        88 SEND-TEST-ERROR-MSG       VALUE 'Y'.
        88 NO-MSG-REQUIRED           VALUE 'N'.

     10 WS-ERROR-STATUS-MSG-SW       PIC X(01) VALUE 'N'.
        88 SEND-TEST-ERR-STATUS-MSG  VALUE 'Y'.
        88 NO-STATUS-REQUIRED        VALUE 'N'.

     10 WS-NOERR-STATUS-MSG-SW       PIC X(01) VALUE 'N'.
        88 SEND-NOERROR-STATUS-MSG   VALUE 'Y'.
        88 NO-ERROR-REQUIRED         VALUE 'N'.

     10 WS-ERROR-HAPPENED-SW         PIC X(01) VALUE 'N'.
        88 ERROR-HAPPENED            VALUE 'Y'.
```

```
        88 NO-ERROR-YET              VALUE 'N'.

*****************************************************************
* A NUMBER FOR INCRIMENTING.                                    *
*****************************************************************
 01  WS-VARIABLES.
     05  WS-INCRINUM                PIC 99  VALUE ZEROES.
     05  WS-DIS-NUM                 PIC 9(4) VALUE ZEROES.
     05  VTABLE-CTR                 PIC S9(8) COMP VALUE 1.
     05  ERROR-CHECK                PIC X(15) VALUE
                                    '&ERRORMSG      '.
     05  STATUS-CHECK               PIC X(15) VALUE
                                    '&STATUSMSG     '.
     05  STATNE-CHECK               PIC X(15) VALUE
                                    '&STATNEMSG     '.
     05  MESSNE-CHECK               PIC X(15) VALUE
                                    '&MESSAGE       '.

 01 MESSAGES.
    05  ERROR1-MSG.
        07  ERROR1-TEXT1            PIC X(19)  VALUE
            'ERROR WITH CALL TO '.
        07  ERROR1-CALL             PIC X(10)  VALUE SPACES.|
        07  ERROR1-TEXT2            PIC X(14)  VALUE
            ' - SPRC CODE: '.
        07  ERROR1-SPRC             PIC X(03)  VALUE SPACES.

*****************************************************************
* OUTPUT RECORD DESCRIPTION.                                    *
*****************************************************************

 01  WS-OUT-KEYWORD-MSG.
     10 FILLER                      PIC X(55) VALUE
     '**--> THE FOLLOWING IS A LIST OF THE KEYWORDS SENT.    '.

 01  H-TABLE-NAME.
     10 H-TABLE-NAME-T OCCURS 15 TIMES.
        15 H-T-NAME                 PIC X.
 01  H-TABLE-VALUE.
     10 H-TABLE-VALUE-T OCCURS 28 TIMES.
        15 H-T-VALUE                PIC X.

 01  WS-KEYWORD-REC.
     10 WS-KEY-MSG-AREA.
        15 FILLER                   PIC X(07) VALUE 'KEYW-> '.
        15 WS-KEY-MSG-NUM           PIC X(02) VALUE SPACES.
```

```
           15 FILLER                      PIC X(01) VALUE ':'.
        10 WS-KEYWORD-OUT                 PIC X(15) VALUE SPACES.
        10 FILLER                         PIC X(02) VALUE '= '.
        10 WS-KEY-VALUE-OUT.
           15 FILLER                      PIC X(24) VALUE SPACES.
           15 WS-KEY-VAL-LEN              PIC X(04) VALUE SPACES.

 *  THIS  SWITCH IS USED FOR TESTING IF RPC CALL
  77  RSPRPC-SWITCH   PIC S9(4) COMP VALUE 0.
      88 RPC-CALL                     VALUE 0.

  01  COMMAREA-POINTER                USAGE IS POINTER.

  LINKAGE SECTION.

  01  DFHCOMMAREA.
      05  NOT-USED                 PIC X(1).

*****************************************************************
 *  THIS IS THE ACTUAL SPAREA POINTER AND DEFINITION              *
 *****************************************************************
  01  LWKCOMMAREA.
      COPY SPAREAC.

************************************************************
 * THIS IS THE MASK FOR THE KEYWORD VARIABLE TABLE THAT THE
 * MAINFRAMECONNECT  WILL CREATE FOR YOUR RSP TO PROCESS.
 ************************************************************
  01  KEYWORD-VTABLE.
      10 VTABLE-SIZE                   PIC S9(8) COMP.
      10 VTABLE-ENTRY OCCURS 0 TO 50 TIMES
                      DEPENDING ON VTABLE-SIZE
                      INDEXED BY VTABLE-INDEX.
         15 VTABLE-NAME                USAGE IS POINTER.
         15 VTABLE-VALUE               USAGE IS POINTER.
         15 VTABLE-NAME-LENGTH         PIC S9(4) COMP.
         15 VTABLE-VALUE-LENGTH        PIC S9(4) COMP.

************************************************************
 * THESE ARE THE DATA VARIABLES THAT THE KEYWORDS AND THE
 * KEYWORD VALUES WILL BE PLACED INTO FOR ACCESS BY THE RSP.
 * IN THIS CASE THE LENGTHS WERE SET TO 15 FOR KEYWORDS AND
 * 28 FOR THE KEYWORD VALUE FOR TESTING PURPOSES.
 ************************************************************
  01  TABLE-NAME                      PIC X(15).
  01  TABLE-VALUE                     PIC X(28).
```

```
  01  LS-OUTPUT-REC.
      10 LS-OUTPUT-DATA               PIC X(55).

*=============================================================*
 PROCEDURE DIVISION.
 *=============================================================*

  000-MAIN-PROCESSING.

      PERFORM 100-INITIALIZE             THRU 100-EXIT.

      IF NO-ERROR-YET
          PERFORM 500-PROCESS-I-O        THRU 500-EXIT.

      PERFORM 900-WRAP-UP                THRU 900-EXIT.

      EXEC CICS
           RETURN
      END-EXEC.

      GOBACK.

  000-EXIT.
      EXIT.

 100-INITIALIZE.

 *****************************************************
 * IF THIS IS A RPC CALL, CALL RPSETUP TO INITIALIZE SPAREA
 * AND OPEN SERVER (TRANSACTION ROUTER SERVICE)
 * IF THIS IS A RSP CALL, SPAREA IS PASSED IN THE COMMAREA.
 * (DIRECTCONNECT).
 * FOR TRACING, MOVE 'Y' TO SPTRCOPT
 *****************************************************

      MOVE EIBCALEN TO RSPRPC-SWITCH.

      IF RPC-CALL
         EXEC CICS GETMAIN
               SET      (COMMAREA-POINTER)
               FLENGTH  (LENGTH OF LWKCOMMAREA)
         END-EXEC
         SET ADDRESS OF LWKCOMMAREA TO COMMAREA-POINTER
         CALL 'RPSETUP'               USING SPAREA
      ELSE
```

```
        SET ADDRESS OF LWKCOMMAREA TO ADDRESS OF DFHCOMMAREA.

*************************************************

     MOVE 'OK'                          TO SPSTATUS.

*****************************************************************
* ALLOCATE A BLOCK OF STORAGE TO BE USED FOR THE DATA
* SET POINTER VARIABLE TO ADDRESS OF ALLOCATED STORAGE
*****************************************************************

     EXEC CICS
       GETMAIN SET(WS-OUTPUT-POINTER)
               LENGTH(55)
    END-EXEC.
    SET ADDRESS OF LS-OUTPUT-REC TO WS-OUTPUT-POINTER.

    PERFORM 120-OPEN-OUTPUT-PIPE      THRU 120-EXIT.

 100-EXIT.
     EXIT.

120-OPEN-OUTPUT-PIPE.
*****************************************************************
* OPEN THE OUTPUT PIPE.                                         *
*****************************************************************
     MOVE 'STD'                         TO SPFORMAT.
     MOVE 55                            TO SPMAXLEN.
     MOVE 'OUTPUT'                      TO SPMODE.

*-------------------------------------------------------*
* AN OPEN PIPE WILL SET UP THE COLUMN INFORMATION, WHICH*
* WILL EVENTUALLY BE SENT TO THE CLIENT.......          *
*-------------------------------------------------------*
     CALL 'OPENPIPE' USING SPAREA.

*****************************************************************
* IF OPEN FAILED, THEN ISSUE AN ERROR MESSAGE.                  *
*****************************************************************
     IF SPRC NOT = '000'
         SET ERROR-HAPPENED             TO TRUE
         MOVE 'OPENPIPE'                TO ERROR1-CALL
         PERFORM 800-ERROR-MESSAGE      THRU 800-EXIT.

 120-EXIT.
     EXIT.
```

```
500-PROCESS-I-O.

     PERFORM 510-KEYWORD-INPUT-CHECK   THRU 510-EXIT.

     IF NO-ERROR-YET
         PERFORM 520-PROCESS-KEYWORDS  THRU 520-EXIT.

 500-EXIT.
     EXIT.

510-KEYWORD-INPUT-CHECK.
*****************************************************************
* MAKE SURE AT LEAST ONE KEYWORD WAS SENT ALONG WITH PROGRAM     *
*****************************************************************

     MOVE 0                            TO WS-INCRINUM.

     IF SPVARTAB = NULL
         PERFORM 700-LOAD-KEYWORD-ERROR   THRU 700-EXIT
         GO TO 510-EXIT.

     IF VTABLE-SIZE NOT > 0
         PERFORM 700-LOAD-KEYWORD-ERROR   THRU 700-EXIT
         GO TO 510-EXIT.

     SET ADDRESS OF KEYWORD-VTABLE     TO SPVARTAB.

 510-EXIT.
     EXIT.

520-PROCESS-KEYWORDS.

     PERFORM 522-SEND-KEYWORD-HEADING  THRU 522-EXIT.

     IF NO-ERROR-YET
         PERFORM 524-READ-WRITE-KEYWORDS THRU 524-EXIT.

     IF NO-ERROR-YET
         PERFORM 548-TEST-FOR-ERR-KEY  THRU 548-EXIT.

 520-EXIT.
     EXIT.

522-SEND-KEYWORD-HEADING.
```

```
     MOVE WS-OUT-KEYWORD-MSG           TO LS-OUTPUT-REC.
     MOVE 55                           TO SPRECLEN.
     SET SPFROM TO ADDRESS OF LS-OUTPUT-REC.

*-------------------------------------------------------*
* PUTPIPE SENDS A RESULT ROW TO THE OUTPUT BUFFER, WHICH*
* WILL EVENTUALLY BE SENT DOWN TO THE CLIENT APPLICATION.*
*-------------------------------------------------------*
     CALL 'PUTPIPE' USING SPAREA.

     IF SPRC NOT = '000'
         SET ERROR-HAPPENED            TO TRUE
         MOVE 'PUTPIPE '               TO ERROR1-CALL
         PERFORM 800-ERROR-MESSAGE     THRU 800-EXIT.

 522-EXIT.
     EXIT.

524-READ-WRITE-KEYWORDS.
*****************************************************************
* OBTAIN THE KEYWORD VARIABLES AND DISPLAY THEM DOWN OUTPUT PIPE *
* THE KEYWORD VALUE LENGTH (VTABLE-VALUE-LENGTH(VTABLE-INDEX))   *
* PASSED FROM MAINFRAMECONNECT WILL BE PLACED AT THE LAST FOUR   *
* BYTES OF THE KEYWORD VALUE DISPLAY.  THIS WILL DEMONSTATE THE  *
* WAY MAINFRAMECONNECT  DETERMINES THE LENGTH OF THE KEYWORD     *
* VALUE MAY NOT MATCH EXACTLY WHAT WAS SENT BECAUSE THE COUNTING *
* STOPS AT THE FIRST SPACE IF THE DATA IS NOT DELIMITED.         *
* NOTE THAT THIS DOES NOT MEAN ONLY PART OF THE KEYWORD VALUE    *
* DATA WAS SENT - IT ONLY MEANS THE COUNTING STOPS AT THE SPACE  *
*****************************************************************
     PERFORM WITH TEST AFTER
             VARYING VTABLE-INDEX FROM 1 BY 1
             UNTIL VTABLE-SIZE = VTABLE-INDEX
         SET ADDRESS OF TABLE-NAME   TO VTABLE-NAME(VTABLE-INDEX)
         MOVE TABLE-NAME               TO H-TABLE-NAME
         MOVE VTABLE-NAME-LENGTH(VTABLE-INDEX)
                                       TO VTABLE-CTR
         ADD 1                         TO VTABLE-CTR
         PERFORM UNTIL VTABLE-CTR > 16
           MOVE SPACE                  TO H-T-NAME (VTABLE-CTR)
           ADD 1                       TO VTABLE-CTR
         END-PERFORM
         MOVE H-TABLE-NAME             TO WS-KEYWORD-OUT
         IF WS-KEYWORD-OUT = ERROR-CHECK
             MOVE 'Y'                  TO WS-ERROR-MSG-SW
         END-IF
```

```
          IF WS-KEYWORD-OUT = STATUS-CHECK
              MOVE 'Y'                  TO WS-ERROR-STATUS-MSG-SW
          END-IF
          IF WS-KEYWORD-OUT = STATNE-CHECK
              MOVE 'Y'                  TO WS-NOERR-STATUS-MSG-SW
          END-IF
          IF WS-KEYWORD-OUT = MESSNE-CHECK
              MOVE 'THIS IS YOUR NON ERROR MESSAGE TEXT.'
                                        TO SPMSG
              MOVE '14'                 TO SPCODE
              CALL 'MESSAGE' USING SPAREA
          END-IF
          SET ADDRESS OF TABLE-VALUE
                                    TO VTABLE-VALUE(VTABLE-INDEX)
          MOVE TABLE-VALUE              TO H-TABLE-VALUE
          MOVE VTABLE-VALUE-LENGTH(VTABLE-INDEX)
                                        TO VTABLE-CTR, WS-DIS-NUM
          ADD 1                         TO VTABLE-CTR
          PERFORM UNTIL VTABLE-CTR > 29
            MOVE SPACE                  TO H-T-VALUE (VTABLE-CTR)
            ADD 1                       TO VTABLE-CTR
          END-PERFORM
          MOVE H-TABLE-VALUE            TO WS-KEY-VALUE-OUT
          MOVE WS-DIS-NUM               TO WS-KEY-VAL-LEN
          ADD 1                         TO WS-INCRINUM
          MOVE WS-INCRINUM              TO WS-KEY-MSG-NUM
          MOVE WS-KEYWORD-REC           TO LS-OUTPUT-REC
          SET SPFROM TO ADDRESS OF LS-OUTPUT-REC
          MOVE 55                       TO SPRECLEN
          CALL 'PUTPIPE' USING SPAREA
          IF SPRC NOT = '000'
              SET ERROR-HAPPENED        TO TRUE
              MOVE 'PUTPIPE '           TO ERROR1-CALL
              PERFORM 800-ERROR-MESSAGE THRU 800-EXIT
          END-IF
      END-PERFORM.

  524-EXIT.
      EXIT.

 548-TEST-FOR-ERR-KEY.
*****************************************************************
* TEST FOR ERROR MESSAGE REQUESTED - SEND ONE IF SO.            *
*****************************************************************
      IF SEND-TEST-ERROR-MSG
          MOVE 'N'                      TO WS-ERROR-MSG-SW
```

```
         MOVE 'THIS IS YOUR ERROR MESSAGE TEXT.'
                                       TO SPMSG
         MOVE 'ERR54321'               TO SPCODE
*-------------------------------------------------------*
* MESSAGE WILL WRITE THE 100 BYTE SPMSG TO A MSG BUFFER,*
* WHICH WILL EVENTUALLY BE WRITTEN TO THE CLIENT        *
* APPLICATION.                                          *
*-------------------------------------------------------*
         CALL 'MESSAGE' USING SPAREA.

     IF SPRC NOT = '000'
         SET ERROR-HAPPENED            TO TRUE.

 548-EXIT.
     EXIT.

700-LOAD-KEYWORD-ERROR.
*****************************************************************
* IF AT LEAST ONE KEYWORD IS NOT SUPPLIED - SEND MSG AND STOP.   *
*****************************************************************

     SET ERROR-HAPPENED                TO TRUE.
     MOVE '* ERROR - NO KEYWORDS SENT' TO SPMSG.
     MOVE 'E'                          TO SPSTATUS.
*-------------------------------------------------------*
* MESSAGE WILL WRITE THE 100 BYTE SPMSG TO A MSG BUFFER,*
* WHICH WILL EVENTUALLY BE WRITTEN TO THE CLIENT        *
* APPLICATION                                           *
*-------------------------------------------------------*
     CALL 'MESSAGE' USING SPAREA.

     IF SPRC NOT = '000'
         SET ERROR-HAPPENED            TO TRUE.

 700-EXIT.
     EXIT.

800-ERROR-MESSAGE.
*****************************************************************
* SOMETHING FAILED, SO ISSUE AN ERROR MESSAGE AND GET OUT.       *
*****************************************************************
     MOVE SPRC                         TO ERROR1-SPRC.
     MOVE ERROR1-MSG                   TO SPMSG.
     MOVE 'E'                          TO SPSTATUS.

*-------------------------------------------------------*
```

```
 * MESSAGE WILL WRITE THE 100 BYTE SPMSG TO A MSG BUFFER,*
 * WHICH WILL EVENTUALLY BE WRITTEN TO THE CLIENT        *
 * APPLICATION                                           *
 *-------------------------------------------------------*
      CALL 'MESSAGE' USING SPAREA.

      IF SPRC NOT = '000'
          SET ERROR-HAPPENED            TO TRUE.

  800-EXIT.
      EXIT.

 900-WRAP-UP.
 *********************************************************************
 * CLOSE PIPES - ISSUE STATUS.
*********************************************************************

      IF NO-ERROR-YET
          MOVE 'OUTPUT'                 TO SPMODE
 *-------------------------------------------------------*
 *CLOSEPIPE IS LIKE CLOSING A FILE,  PLACES AN EOF MARKER*
 *-------------------------------------------------------*
          CALL 'CLOSPIPE' USING SPAREA
          IF SPRC NOT = '000'
              SET ERROR-HAPPENED        TO TRUE
              MOVE 'CLOSPIPE'           TO ERROR1-CALL
              PERFORM 800-ERROR-MESSAGE THRU 800-EXIT.

      IF SEND-TEST-ERR-STATUS-MSG
      OR ERROR-HAPPENED
          MOVE 'N'                      TO WS-ERROR-MSG-SW
          MOVE 'THIS IS YOUR STATUS MESSAGE TEXT.'
                                        TO SPMSG
          MOVE '-321'                   TO SPCODE
          MOVE 'E'                      TO SPSTATUS
      ELSE
          IF SEND-NOERROR-STATUS-MSG
              MOVE 'N'                  TO WS-ERROR-MSG-SW
              MOVE 'THIS IS YOUR STATUS NOERROR TEXT.'
                                        TO SPMSG
              MOVE '12'                 TO SPCODE
              MOVE 'OK'                 TO SPSTATUS
          ELSE
              MOVE 'OK'                 TO SPSTATUS
      END-IF.
```

```
*-------------------------------------------------------------*
*    CALLING STATUS WILL FLUSH ANY RESULTS AND/OR             *
*    MESSAGES FROM THE BUFFERS, TO THE CLIENT APPLICATION     *
*-------------------------------------------------------------*
     CALL 'STATUS'   USING SPAREA.
     IF SPRC NOT = '000'
         SET ERROR-HAPPENED            TO TRUE
         MOVE 'STATUS  '               TO ERROR1-CALL
         PERFORM 800-ERROR-MESSAGE     THRU 800-EXIT
     END-IF.

*******************************************************
*  CLOSE OPEN SERVER
*    IF THIS IS AN RPC CALL, PERFORM OPEN SERVER CLOSE
*******************************************************
     IF RPC-CALL
        CALL 'RPDONE' USING SPAREA.

 900-EXIT.
     EXIT.
```

APPENDIX E

RSP8C Variable Text Sample RSP

RSP8C is a sample RSP that reads variable text and uses output pipes to echo the data the client application sends to it. If you want to pass parameters to the RSP without using keywords, RSP8C is a useful sample.

This appendix discusses the following topics:

- Client application processing
- RSP8C variable text sample code

Client application processing

The following Figure E-1 contains an example that uses ISQL to invoke the RSP8C sample RSP. RSP8C reads up to 10,000 bytes of variable text as input and returns the same data for display in 50-byte blocks.

Figure E-1: Sample RSP8C input

```
================================================================
C:\DIRECTCONNECT>> isql -Sdcservice -Uuserid
USE PROCEDURE RSP8C
12345678911234567892123456789312345678941234567895
12345678911234567892123456789312345678941234567895
12345678911234567892123456789312345678941234567895
12345678911234567892123456789312345678941234567895
12345678911234567892123456789312345678941234567895
12345678911234567892123456789312345678941234567895
12345678911234567892123456789312345678941234567895
12345678911234567892123456789312345678941234567895
12345678911234567892123456789312345678941234567895
12345678911234567892123456789312345678941234567895
;
GO
================================================================
```

In the preceding figure, the variable input text string, 500 characters in length, is separated into ten 50-byte blocks that have a carriage-control character at the end of each block.

The carriage-control character counts as the 51st character of each block. The following figure shows that the carriage-control characters are reflected in the output data records as spaces, making the total number of characters returned 510.

Figure E-2: Sample RSP8C output

```
================================================================
1 2 3 4 5 6 7 8 9 10 11 12 13  COLUMN01
 ---------------------------------------------------------
 **-- THE FOLLOWING IS 50 BYTE BLOCKS OF VARIABLE TEXT RECVD
 REC#- 01:12345678911234567892123456789312345678941234567895
 REC#- 02: 123456789112345678921234567893123456789412345678 9
 REC#- 03:5 12345678911234567892123456789312345678941234567 8
 REC#- 04:95 1234567891123456789212345678931234567894123456 7
 REC#- 05:895 123456789112345678921234567893123456789412345 6
 REC#- 06:7895 12345678911234567892123456789312345678941234 5
 REC#- 07:67895 1234567891123456789212345678931234567894123 4
 REC#- 08:567895 123456789112345678921234567893123456789412 3
 REC#- 09:4567895 12345678911234567892123456789312345678941 2
 REC#- 10:34567895 1234567891123456789212345678931234567894 1
 REC#- 11:234567895

(12 rows affected)
================================================================
```

RSP8C variable text sample code

RSP8C is an example of an RSP written to handle variable text sent to it from the client application. The code in this sample RSP follows.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID.  RSP8C
*****************************************************************
*  RSP8C - DOCTORED STORED PROCEDURE                            *
*                                                               *
*  THIS SAMPLE STORED PROCEDURE WAS WRITTEN TO USE A VARIABLE   *
*  TEXT PARAMETER OF UP TO 10,000 BYTES AND ECHOES IT BACK THRU *
*  A STANDARD OUTPUT PIPE IN 50 BYTE INCREMENTS.                *
*                                                               *
*  USE PROCEDURE WITH DATA RSP8C 'THIS IS A VERY BIG PARAMETER' *
*                                                               *
* THE VARIABLE TEXT DOESN'T HAVE TO BE DELIMITED WITH QUOTES OR *
* DOUBLE QUOTES.                                                *
*****************************************************************
ENVIRONMENT DIVISION.

 DATA DIVISION.
```

```
  WORKING-STORAGE SECTION.

*****************************************************************
 * POINTERS TO INPUT AND OUTPUT RECORD AREA.                     *
 *****************************************************************
  01  WS-SAMPLE-POINTER.
      10 WS-OUTPUT-POINTER        USAGE IS POINTER.

*****************************************************************
 * SWITCHES FOR RECORD PROCESSING CONTROL.                       *
 *****************************************************************
  01  WS-SWITCHES.
      10 WS-ERROR-HAPPENED-SW         PIC X(01) VALUE 'N'.
         88 ERROR-HAPPENED            VALUE 'Y'.
         88 NO-ERROR-YET              VALUE 'N'.

      10 WS-OUTPUT-DONE-SW            PIC X(01) VALUE 'N'.
         88 OUTPUT-DONE               VALUE 'Y'.
         88 MORE-OUTPUT               VALUE 'N'.

 *  THIS  SWITCH IS USED FOR TESTING IF RPC CALL
  77  RSPRPC-SWITCH   PIC S9(4) COMP VALUE 0.
      88 RPC-CALL                    VALUE 0.

  01  COMMAREA-POINTER               USAGE IS POINTER.

*****************************************************************
 * A NUMBER FOR INCREMENTING.                                    *
 *****************************************************************
  01  WS-VARIABLES.
      05  WS-INCRINUM                PIC 99  VALUE ZEROES.
      05  VTABLE-CTR                 PIC S9(8) COMP VALUE 0.
      05  WS-LEN-HOLD                PIC 9(4) VALUE ZEROES.

  01 MESSAGES.
     05  ERROR1-MSG.
         07  ERROR1-TEXT1            PIC X(19)  VALUE
             'ERROR WITH CALL TO '.
         07  ERROR1-CALL             PIC X(10)  VALUE SPACES.
         07  ERROR1-TEXT2            PIC X(14)  VALUE
             ' - SPRC CODE: '.
         07  ERROR1-SPRC             PIC X(03)  VALUE SPACES.
*****************************************************************
 * OUTPUT RECORD DESCRIPTION.                                    *
 *****************************************************************
```

```
 01  WS-OUTPUT-REC.
     10 WS-OUT-MSG-AREA.
        15 FILLER                    PIC X(07) VALUE 'REC#-> '.
        15 WS-OUT-MSG-NUM            PIC X(02) VALUE SPACES.
        15 FILLER                    PIC X(01) VALUE ':'.
     10 WS-OUT-SOME-DATA             PIC X(50) VALUE SPACES.

 01  WS-OUT-DATA-MSG.
     10 FILLER                       PIC X(55) VALUE
     '**--> THE FOLLOWING IS 50 BYTE BLOCKS OF VARIABLE TEXT '.
     10 FILLER                       PIC X(05) VALUE 'RECVD'.

 01  V-TABLE-BLOCKS.
     10 V-TABLE-BLOCKS-T OCCURS 200 TIMES.
        15 V-ROW                     PIC X(50) VALUE SPACES.

 01  WS-VTABLE-REC.
     10 WS-VTABLE-AREA.
        15 FILLER                    PIC X(33) VALUE
           'THIS IS THE LENGTH IN SPVARLEN : '.
        15 WS-VTABLE-NUM             PIC X(04) VALUE SPACES.
        15 FILLER                    PIC X(03) VALUE SPACES.

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

*****************************************************************
* LINKAGE TO CALLING PROGRAM                                    *
*****************************************************************

 01  DFHCOMMAREA.
     05  NOT-USED                 PIC X(1).
     05  DUMMY-AREA               PIC X(1).

***************************************************************
*  THIS IS THE ACTUAL SPAREA POINTER AND DEFINITION           *
***************************************************************
 01  LWKCOMMAREA.
     COPY SPAREAC.
*****************************************************************
* VARIABLE FOR ALL INCOMING VARIABLE TEXT PARAMETERS            *
*****************************************************************
 01  INPUT-VALUE                    PIC X(10000).
```

```
  01  WS-OUTPUT-RECORD.
      10 WS-OUTPUT-DATA               PIC X(60).

*================================================================*
 PROCEDURE DIVISION.
 *================================================================*
 000-MAIN-PROCESSING.

      PERFORM 100-INITIALIZE            THRU 100-EXIT.

      IF NO-ERROR-YET
          PERFORM 500-PROCESS-I-O       THRU 500-EXIT.

      PERFORM 900-WRAP-UP               THRU 900-EXIT.

      EXEC CICS
           RETURN

      END-EXEC.

      GOBACK.

  000-EXIT.
      EXIT.

 100-INITIALIZE.

 ******************************************************
 * IF THIS IS A RPC CALL, CALL RPSETUP TO INITIALIZE SPAREA
 * AND OPEN SERVER (TRANSACTION ROUTER SERVICE)
 * IF THIS IS A RSP CALL, SPAREA IS PASSED IN THE COMMAREA.
 * (DIRECTCONNECT).
 * FOR TRACING, MOVE 'Y' TO SPTRCOPT
 ******************************************************

      MOVE EIBCALEN TO RSPRPC-SWITCH.

      IF RPC-CALL
         EXEC CICS GETMAIN
               SET      (COMMAREA-POINTER)
               FLENGTH  (LENGTH OF LWKCOMMAREA)
         END-EXEC
         SET ADDRESS OF LWKCOMMAREA TO COMMAREA-POINTER
         MOVE 'Y'                     TO SPTRCOPT
         CALL 'RPSETUP'               USING SPAREA
```

```
      ELSE
         SET ADDRESS OF LWKCOMMAREA TO ADDRESS OF DFHCOMMAREA
         MOVE 'Y'                   TO SPTRCOPT.

      MOVE 'OK'                     TO SPSTATUS.

      PERFORM 110-ESTABLISH-INPUT   THRU 110-EXIT.

 ****************************************************************
 * ALLOCATE A BLOCK OF STORAGE TO BE USED FOR THE DATA
 * SET POINTER VARIABLE TO ADDRESS OF ALLOCATED STORAGE
****************************************************************

      EXEC CICS
        GETMAIN SET(WS-OUTPUT-POINTER)
                LENGTH(60)
      END-EXEC.

      SET ADDRESS OF WS-OUTPUT-RECORD   TO WS-OUTPUT-POINTER.

      IF NO-ERROR-YET
          PERFORM 120-OPEN-OUTPUT-PIPE  THRU 120-EXIT.

  100-EXIT.
      EXIT.

 110-ESTABLISH-INPUT.
      IF SPVARLEN < 1
          SET ERROR-HAPPENED            TO TRUE
          MOVE 'NO PARMS'               TO ERROR1-CALL
          PERFORM 800-ERROR-MESSAGE     THRU 800-EXIT
          GO TO 110-EXIT
      ELSE
          MOVE SPVARLEN                 TO WS-LEN-HOLD
          MOVE WS-LEN-HOLD              TO WS-VTABLE-NUM
          MOVE WS-VTABLE-REC            TO SPMSG
          MOVE 'OK'                     TO SPSTATUS
 *-------------------------------------------------------*
 * MESSAGE WILL WRITE THE 100 BYTE SPMSG TO A MSG BUFFER,*
 * WHICH WILL EVENTUALLY BE WRITTEN TO THE CLIENT        *
 * APPLICATION                                           *
 *-------------------------------------------------------*
          CALL 'MESSAGE' USING SPAREA.

      SET ADDRESS OF INPUT-VALUE        TO SPVARTXT.
```

```
      MOVE INPUT-VALUE (1:SPVARLEN)     TO V-TABLE-BLOCKS.

      IF V-ROW (1) = SPACES
          SET ERROR-HAPPENED            TO TRUE
          MOVE 'SPACES  '               TO ERROR1-CALL
          PERFORM 800-ERROR-MESSAGE     THRU 800-EXIT.

      IF V-ROW (1) = LOW-VALUES
          SET ERROR-HAPPENED            TO TRUE
          MOVE 'LOWVALUE'               TO ERROR1-CALL
          PERFORM 800-ERROR-MESSAGE     THRU 800-EXIT.

  110-EXIT.
      EXIT.

 120-OPEN-OUTPUT-PIPE.
      MOVE 'STD'                        TO SPFORMAT.
      MOVE 60                           TO SPMAXLEN.
      MOVE 'OUTPUT'                     TO SPMODE.
 *---------------------------------------------------------*
 * AN OPEN PIPE WILL SET UP THE COLUMN INFORMATION, WHICH*
 * WILL EVENTUALLY BE SENT TO THE CLIENT APPLICATION     *
 *---------------------------------------------------------*

      CALL 'OPENPIPE' USING SPAREA.

      IF SPRC NOT = '000'
          SET ERROR-HAPPENED            TO TRUE
          MOVE 'OPENPIPE'               TO ERROR1-CALL
          PERFORM 800-ERROR-MESSAGE     THRU 800-EXIT.

  120-EXIT.
      EXIT.

500-PROCESS-I-O.

      IF NO-ERROR-YET
          PERFORM 540-PROCESS-DATA-RECS THRU 540-EXIT.

  500-EXIT.
      EXIT.

 540-PROCESS-DATA-RECS.
 ****************************************************************
 * OBTAIN VARIABLE TEXT SENT WITH PROGRAM.                      *
 ****************************************************************
```

```
     MOVE 0                              TO WS-INCRINUM.

     PERFORM 542-SEND-RECORDS-HEADING  THRU 542-EXIT.

     IF NO-ERROR-YET
         PERFORM 544-READ-WRITE-RECORDS THRU 544-EXIT
                 UNTIL OUTPUT-DONE OR ERROR-HAPPENED.

 540-EXIT.
     EXIT.

542-SEND-RECORDS-HEADING.

     IF SPSTATUS = 'OK'
         MOVE WS-OUT-DATA-MSG             TO WS-OUTPUT-RECORD
         MOVE 60                          TO SPRECLEN
         SET SPFROM TO ADDRESS OF WS-OUTPUT-RECORD
*-------------------------------------------------------*
* PUTPIPE SENDS A RESULT ROW TO THE OUTPUT BUFFER, WHICH*
* WHICH WILL EVENTUALLY BE WRITTEN TO THE CLIENT        *
* APPLICATION                                           *
*-------------------------------------------------------*
         CALL 'PUTPIPE' USING SPAREA
         IF SPRC NOT = '000'
             SET ERROR-HAPPENED           TO TRUE
              MOVE 'PUTPIPE '              TO ERROR1-CALL
              PERFORM 800-ERROR-MESSAGE    THRU 800-EXIT
         END-IF
     END-IF.

 542-EXIT.
     EXIT.

544-READ-WRITE-RECORDS.
*****************************************************************
* LOOP THROUGH VARIABLE TEXT TABLE AND SEND BACK TO CLIENT IN    *
* 50-BYTE CHUNKS UNTIL ALL ARE RETURNED.                         *
 *****************************************************************

     ADD 1                               TO WS-INCRINUM,
                                            VTABLE-CTR.

     IF V-ROW (VTABLE-CTR) IS = SPACES
     OR V-ROW (VTABLE-CTR) IS = LOW-VALUES
     OR VTABLE-CTR > 200
```

```
         IF VTABLE-CTR = 1
             MOVE WS-INCRINUM          TO WS-OUT-MSG-NUM
             MOVE V-ROW (VTABLE-CTR)   TO WS-OUT-SOME-DATA
             MOVE WS-OUTPUT-REC        TO WS-OUTPUT-RECORD
             SET SPFROM TO ADDRESS OF WS-OUTPUT-RECORD
*-------------------------------------------------------*
* PUTPIPE SENDS A RESULT ROW TO THE OUTPUT BUFFER,      *
* WHICH WILL EVENTUALLY BE WRITTEN TO THE CLIENT        *
* APPLICATION                                           *
*-------------------------------------------------------*
             CALL 'PUTPIPE' USING SPAREA
             IF SPRC NOT = '000'
                 SET ERROR-HAPPENED    TO TRUE
                 MOVE 'PUTPIPE '       TO ERROR1-CALL
                 PERFORM 800-ERROR-MESSAGE THRU 800-EXIT
             END-IF
         END-IF
         SET OUTPUT-DONE               TO TRUE
     ELSE
         MOVE WS-INCRINUM              TO WS-OUT-MSG-NUM
         MOVE V-ROW (VTABLE-CTR)       TO WS-OUT-SOME-DATA
         MOVE WS-OUTPUT-REC            TO WS-OUTPUT-RECORD
         SET SPFROM TO ADDRESS OF WS-OUTPUT-RECORD
*-------------------------------------------------------*
* PUTPIPE SENDS A RESULT ROW TO THE OUTPUT BUFFER, WHICH*
* WILL EVENTUALLY BE SENT DOWN TO THE CLIENT APPLICATION*
*-------------------------------------------------------*
         CALL 'PUTPIPE' USING SPAREA
         IF SPRC NOT = '000'
             SET ERROR-HAPPENED        TO TRUE
             SET OUTPUT-DONE           TO TRUE
             MOVE 'PUTPIPE '           TO ERROR1-CALL
             PERFORM 800-ERROR-MESSAGE THRU 800-EXIT
         END-IF
     END-IF.

 544-EXIT.
     EXIT.

800-ERROR-MESSAGE.
*****************************************************************
* SOMETHING FAILED, SO ISSUE AN ERROR MESSAGE AND GET OUT.      *
*****************************************************************
     MOVE SPRC                         TO ERROR1-SPRC.
     MOVE ERROR1-MSG                   TO SPMSG.
     MOVE 'E'                          TO SPSTATUS.
```

```
*-------------------------------------------------------*
* MESSAGE WILL WRITE THE 100 BYTE SPMSG TO A MSG BUFFER,*
* WHICH WILL EVENTUALLY BE WRITTEN TO THE CLIENT        *
* APPLICATION                                           *
*-------------------------------------------------------*
     CALL 'MESSAGE' USING SPAREA.

     IF SPRC NOT = '000'
         SET ERROR-HAPPENED            TO TRUE.

 800-EXIT.
     EXIT.

900-WRAP-UP.
*****************************************************************
* CLOSE PIPES - ISSUE STATUS.                                   *
*****************************************************************

     IF NO-ERROR-YET
         MOVE 'OUTPUT'                 TO SPMODE
*-------------------------------------------------------*
*CLOSEPIPE IS LIKE CLOSING A FILE,  PLACES AN EOF MARKER*
*-------------------------------------------------------*
         CALL 'CLOSPIPE' USING SPAREA
         IF SPRC NOT = '000'
             SET ERROR-HAPPENED        TO TRUE
             MOVE 'CLOSPIPE'           TO ERROR1-CALL
             PERFORM 800-ERROR-MESSAGE THRU 800-EXIT.

     IF NO-ERROR-YET
         MOVE 'OK'                     TO SPSTATUS
     ELSE
         MOVE 'E'                      TO SPSTATUS
         MOVE 'MYERCODE'               TO SPCODE
     END-IF.

*-------------------------------------------------------*
*    CALLING STATUS WILL FLUSH ANY RESULTS AND/OR       *
*    MESSAGES FROM THE BUFFERS, TO THE CLIENT APPLICATION   *
*-------------------------------------------------------*
     CALL 'STATUS'   USING SPAREA.
     IF SPRC NOT = '000'
         SET ERROR-HAPPENED            TO TRUE
         MOVE 'STATUS  '               TO ERROR1-CALL
         PERFORM 800-ERROR-MESSAGE     THRU 800-EXIT
```

```
     END-IF.

*********************************************************
*  CLOSE OPEN SERVER
*    IF THIS IS AN RPC CALL, PERFORM OPEN SERVER CLOSE
*********************************************************
     IF RPC-CALL
        CALL 'RPDONE' USING SPAREA.

900-EXIT.
    EXIT.
```

APPENDIX F

The SPAREA

The SPAREA contains all of the pointers, codes, and command details that the RSP needs to exchange with the RSP API. Every RSP receives or sends information using the SPAREA.

This appendix discusses the following topics:

- SPAREA field descriptions
- Copying SPAREA definitions to the RSP
- SPAREA definitions

SPAREA field descriptions

The RSP, Open ServerConnect, and MainframeConnect use the SPAREA by accessing the values from the SPAREA fields. The word *Reserved* in the descriptions indicates that the RSP cannot write to the field.

SPHEADER

SPHEADER contains the character string *SPAREA*. The character string serves as an eye catcher for locating the SPAREA in a dump. *Reserved*.

SPRESRVED

SPRESRVD contains values used by MainframeConnect to process commands. Reserved.

SPTRCOPT

SPTRCOPT controls the trace option. If the field contains 'Y' when an Open ServerConnect command is issued, trace records are written to the TSQ, CE*xxxxxx*, where *xxxxxx* is the first six characters of the user ID.

SPSTATUS

SPSTATUS is used by an RSP or by Open ServerConnect to indicate the success or failure of processing.

When used by an RSP, it refers to RSP processing. When used by Open ServerConnect, it refers to processing on the remote database.

Valid values are:

- 'OK' indicates success.

- 'E' indicates an error.
- 'W' indicates a warning.
- 'R' indicates results.

SPCODE The RSP uses SPCODE to supply user-defined error codes.

SPFORMAT The RSP uses SPFORMAT to specify the data format when opening a data pipe. Valid values are: DB2, STD, and BIN.

SPMODE The RSP uses SPMODE to specify the mode of the data pipe. Valid values are INPUT or OUTPUT.

SPRC MainframeConnect uses SPRC to indicate the success or failure of an RSP command. Valid return codes are:

- '000' indicates successful completion.
- *'xxx'* indicates a MainframeConnect error number.
- 'EOF' indicates an End of File on input data.
- 'ACE' indicates an APPC communication error (when the MainframeConnect Temporary Storage Type configuration property is set to None).
- 'CAN' indicates that the client application issued a DBCANCEL command.

SPFROM The RSP uses SPFROM to specify the address of the STD or BIN format data record that it writes to the output pipe. See PUTPIPE on page 68 for an example of using SPFROM.

SPINTO The RSP uses SPINTO to specify the address of a storage area where the STD or BIN format data record read from the input pipe can be placed. See GETPIPE on page 65 for an example of using SPINTO.

SPSQLDA The RSP and MainframeConnect uses SPSQLDA to specify the address of an SQLDA that describes the data records. This field is only used for DB2 format output data pipes. The RSP must build the SQLDA and supply this pointer when it opens the pipe.

For information on SQLDA structure, see the IBM SQL reference guide for DB2. A sample SQLDA definition is provided in Appendix B, “MODELRSP DB2 Output Pipe Sample RSP.”

SPVARTXT SPVARTXT contains the pointer of the variable text that the client application may optionally send to the RSP. This field contains null.

SPVARTAB SPVARTAB contains the pointer of the variable substitution table, which is created if the client sends keyword variables (that is, &KEYWORD=value format). If keyword variables are not sent, this field contains null.

SPMAXLEN The RSP uses SPMAXLEN to specify the maximum record length for records read from or written to a STD or BIN format pipe. See "Using data pipes" on page 19 for more information.

SPRECLEN The RSP and MainframeConnect uses SPRECLEN to specify the length of records read from or written to a STD or BIN format data pipe.

For output pipes, the RSP must set this field to the length of the record it writes (unless it is writing fixed-length records of the same size as SPMAXLEN). For input pipes, Open ServerConnect sets this field to the length of the record it is sending to the RSP.

For more information, see "SPMAXLEN and SPRECLEN" on page 95. Also see "Using data pipes" on page 19 for more information.

SPVARLEN SPVARLEN contains the length of the variable text that the client may optionally send to the RSP. This field contains zeros.

SPPREFIX Not used.

SPMSG The RSP uses SPMSG to place message text it sends the client application with a MESSAGE command.

Copying SPAREA definitions to the RSP

SPAREA definitions in assembler, COBOL II, PL/I, and C are distributed with Open ServerConnect and are reproduced in this appendix. You can copy the appropriate definition into your RSP and provide the necessary information for the relevant fields. The SPAREA definitions are in the SYBASE.ORSP310B.CICS.SOURCE library, and their definitions are reproduced on the indicated page:

- SPAREAA—Assembler on "SPAREAA assembler definition" on page 143
- SPAREAC—COBOL II on "SPAREAC COBOL II definition" on page 143
- SPAREAP—PL/1 on "SPAREAP PL/1 definition" on page 144
- SPAREAX—C on "SPAREAX C definition" on page 145

Within your RSP, copy the SPAREA definition as shown in the following table. For an example of copying the SPAREA in the context of an RSP written in COBOL II, see the samples in Chapter 3, “Writing an RSP.”

Table F-1: SPAREA copy statements

Language	Copy syntax
Assembler	`COPY SPAREAA`
COBOL II	`COPY SPAREAC.`
PL/I	`EXEC SQL INCLUDE SPAREAP;`
C	`#include "SPAREAX.H"`

When you compile the RSP, the concatenation sequence for SYSLIB must include a DD statement for the MainframeConnect sample program library. See Chapter 4, “Compiling an RSP” and Chapter 5, “Testing and invoking an RSP” for details.

The SPAREA definitions are reproduced on the following pages.

Note There are several fields in the SPAREA definitions in the following section that are used only for Client Services Applications (CSAs). Those fields are described in the Mainframe Connect Client Option *Programmer’s Reference for Client Services Applications.*

SPAREA definitions

This section contains the following SPAREA definitions:

- SPAREAA assembler definition
- SPAREAC COBOL definition
- SPAREAP PL/1 definition
- SPAREAX C definition

These examples show how each programming language opens an input pipe for a STD format data pipe with a maximum record length of 400 bytes.

SPAREAA assembler definition

```
*---------------------------------------*
 *  STORED PROCEDURE COMMUNICATION AREA  *
 *---------------------------------------*
 SPAREA   DSECT
 SPHEADER DS    CL8                     EYE CATCHER
 SPRESRVD DS    CL33                    SERVER INFORMATION
 SPTRCOPT DS    CL1                     TRACE OPT
 SPSTATUS DS    CL2                     STATUS INDICATOR
 SPCODE   DS    CL8                     ERROR CODE
 SPFORMAT DS    CL3                     PIPE FORMAT
 SPMODE   DS    CL6                     PIPE MODE
 SPRC     DS    CL3                     RETURN CODE
 SPFROM   DS    0F                      FROM ADDRESS
 SPINTO   DS    0F                      INTO ADDRESS
 SPSQLDA  DS    F                       SQLDA ADDRESS
 SPVARTXT DS    F                       VARIABLE TEXT
 SPVARTAB DS    F                       VARIABLE TABLE
 SPROWS   DS    F                       ROWS AFFECTED
 SPMAXLEN DS    0H                      MAXIMUM LENGTH OF STD RECORD
 SPRECLEN DS    H                       RECORD LENGTH
 SPVARLEN DS    H                       VARIABLE TEXT LENGTH
 SPPREFIX DS    CL1                     MESSAGE FILE PREFIX
 SPMSG    DS    CL100                   MESSAGE AREA
 SPFILL2  DS    CL3                     NOT USED
 SPSQL    DS    F                       SQL BUFFER ADDRESS
 SPATTACH DS    CL8                     ATTACHMENT NAME
 SPUSERID DS    CL8                     USERID
 SPPWD    DS    CL8                     PASSWORD
 SPCMPOPT DS    CL1                     COMPRESSION OPTION
 SPIND    DS    CL1                     MESSAGE INDICATOR
 SPDATE   DS    CL8                     DATE
 SPTIME   DS    CL8                     TIME
 SPCONFIG DS    CL4                     CONFIGURATION ID
 SPSERVER DS    CL30                    SERVER NAME
          DS    CL32                    FILLER
 SPEND    EQU   *
```

SPAREAC COBOL II definition

```
*---------------------------------------*
 *  STORED PROCEDURE COMMUNICATION AREA  *
```

```
 *---------------------------------------*
 03  SPAREA.
     05  SPHEADER         PIC X(8).
     05  SPRESRVD         PIC X(33).
     05  SPTRCOPT         PIC X.
     05  SPSTATUS         PIC X(2).
     05  SPCODE           PIC X(8).
     05  SPFORMAT         PIC X(3).
     05  SPMODE           PIC X(6).
     05  SPRC             PIC X(3).
     05  SPFROM           USAGE IS POINTER.
     05  SPINTO           REDEFINES SPFROM USAGE IS POINTER.
     05  SPSQLDA          REDEFINES SPINTO USAGE IS POINTER.
     05  SPVARTXT         USAGE IS POINTER.
     05  SPVARTAB         USAGE IS POINTER.
     05  SPROWS           PIC S9(8) COMP.
     05  SPMAXLEN         PIC S9(4) COMP.
     05  SPRECLEN         REDEFINES SPMAXLEN PIC S9(4) COMP.
     05  SPVARLEN         PIC S9(4) COMP.
     05  SPPREFIX         PIC X.
     05  SPMSG            PIC X(100).
     05  FILLER           PIC X(3).
     05  SPSQL            USAGE IS POINTER.
     05  SPATTACH         PIC X(8).
     05  SPUSERID         PIC X(8).
     05  SPPWD            PIC X(8).
     05  SPCMPOPT         PIC X(1).
     05  SPIND            PIC X(1).
     05  SPDATE           PIC X(8).
     05  SPTIME           PIC X(8).
     05  SPCONFIG         PIC(4).
     05  SPSERVER         PIC(30).
     05  FILLER           PIC X(32).
```

SPAREAP PL/1 definition

```
/*************************************/
 /* STORED PROCEDURE COMMUNICATION AREA  */
 /*************************************/
DCL 1 COMMPTR               POINTER;
 DCL 1 SPAREA BASED(COMMPTR),
       3  SPHEADER           CHAR(8),
       3  SPRESRVD           CHAR(33),
```

```
       3  SPTRCOPT           CHAR(1),
       3  SPSTATUS           CHAR(2),
       3  SPCODE             CHAR(8),
       3  SPFORMAT           CHAR(3),
       3  SPMODE             CHAR(6),
       3  SPRC               CHAR(3),
       3  SPFROM             POINTER ALIGNED,
       3  SPVARTXT           POINTER,
       3  SPVARTAB           POINTER,
       3  SPROWS             FIXED BIN(31) ALIGNED,
       3  SPMAXLEN           FIXED BIN(15) ALIGNED,
       3  SPVARLEN           FIXED BIN(15) ALIGNED,
       3  SPPREFIX           CHAR,
       3  SPMSG              CHAR(100),
       3  SPFILL2            CHAR(3),
       3  SPSQL              POINTER ALIGNED,
       3  SPATTACH           CHAR(8),
       3  SPUSERID           CHAR(8),
       3  SPPWD              CHAR(8),
       3  SPCMPOPT           CHAR(1),
       3  SPIND              CHAR(1),
       3  SPDATE             CHAR(8),
       3  SPTIME             CHAR(8);
       3  SPCONFIG           CHAR(4),
       3  SPSERVER           CHAR(30),
       3  SPFILL3            CHAR(32);
 DCL SPINTO  POINTER BASED(AD_SPFROM);
 DCL SPSQLDA POINTER BASED(AD_SPFROM);
 DCL SPRECLEN POINTER BASED(AD_SPMAXLEN);
 DCL SPSQL   POINTER BASED(AD_SPSQL);
 DCL (AD_SPFROM, AD_SPMAXLEN, AD_SPSQL) POINTER;
 AD_SPFROM=ADDR(SPFROM);
 AD_SPMAXLEN=ADDR(SPMAXLEN);
 AD_SPSQL=ADDR(SPSQL);
```

SPAREAX C definition

```
#ifndef SP_DEFS
 #define SP_DEFS
 /*
    Various declarations and definitions for Stored Procedures for C.
    Should be usable with the SAS/C compiler, and with slight
    modification, the IBM C/370 compiler.  Uses the SAS/C digraphs for
```

```
    square brackets - "(]" for the left square bracket, and "])" for the
    right square bracket.
   SAS/C and C/370 are trademarks of the SAS Institute, Inc. and IBM
    Corporation respectively.
*/
 #include "sqlda.h"
/*
    Keyword variable table declaration.
 */
struct VARTAB {
   unsigned long varTabL;    /* Number of entries in table (<<= 50)  */
   struct VARENT {
     char *varName;          /* Variable name                        */
     char *varValue;         /* Variable value                       */
     short varNameL;         /* Variable name length                 */
     short varValL;          /* Variable value length                */
   } varent(]50]);
 };
/*
    Stored Procedure Communication Area declaration.
 */
 struct SPAREA {
   char spheader(]8]);       /* DS    CL8   Eye catcher              */
   char spresrvd(]33])       /* DS    CL33  Server information       */
   char sptrcopt;            /* DS    CL1   Trace options            */
   char spstatus(]2]);       /* DS    CL2   Status indicator         */
   char spcode(]8]);         /* DS    CL8   Error code               */
   char spformat(]3]);       /* DS    CL3   Pipe format              */
   char spmode(]6]);         /* DS    CL6   Pipe mode                */
   char sprc(]3]);           /* DS    CL3   Return code              */
   union {
     char *spfrom;           /* DS    0A    From address             */
     char *spinto;           /* DS    0A    Into address             */
     struct SQLDA *spsqlda;  /* DS    A     SQLDA address            */
   };
   char *spvartxt;           /* DS    A     Variable text            */
   struct VARTAB *spvartab;  /* DS    A     Variable table           */
   int sprows;               /* DS    F     Rows affected            */
   union {
     short spmaxlen;         /* DS    0H    Max length of STD rec    */
     short spreclen;         /* DS    H     Record length            */
   };
   short spvarlen;           /* DS    H     Variable text length     */
   char spprefix;            /* DS    CL1   Message file prefix      */
   char spmsg(]100]);        /* DS    CL100 Message area             */
   char _f0(]3]);            /* Padding for alignment                */
```

```
   struct SQLBUF *spsql;     /* DS    A     SQL buffer address       */
   char spattach(]8]);       /* DS    CL8   Attachment name          */
   char spuserid(]8]);       /* DS    CL8   Userid                   */
   char sppwd(]8]);          /* DS    CL8   Password                 */
   char spcmpopt;            /* DS    CL1   Compression option       */
   char spind;               /* DS    CL1   Message indicator        */
   char spdate(]8]);         /* DS    CL8   Request execution date   */
   char sptime(]8]);         /* DS    CL8   Request execution time   */
   char spconfig]4]);        /* DS    CL4   Configuration name       */
   char spserver(]30]);      /* DS    CL30  Server name              */
   char _f1(]32]);           /* Padding to end of record             */
 };
/*
    Stored procedure function declarations.
 */
 void attach(struct SPAREA *);         /* Attach to remote server     */
 void clospipe(struct SPAREA *);       /* Close input/output pipe     */
 void commit(struct SPAREA *);         /* Issue SYNCPOINT w/COMMIT    */
 void cssetup(struct SPAREA *);        /* Initialize SPAREA           */
 void detach(struct SPAREA *);         /* Detach from remote server   */
 void getmsg(struct SPAREA *);         /* Get a message               */
 void getpipe(struct SPAREA *);        /* Get row from input pipe     */
 void getpipe(struct SPAREA *);        /* Put row to output pipe      */
 void message(struct SPAREA *);        /* Issue message               */
 void openpipe(struct SPAREA *);       /* Open input/output pipe      */
 void reqexec(struct SPAREA *);        /* Execute SQL request         */
 void rescheck(struct SPAREA *);       /* Check for results           */
 void rollback(struct SPAREA *);       /* Issue SYNCPOINT w/ROLLBACK  */
 void status(struct SPAREA *);         /* Issue status                */
#endif
```

APPENDIX G

The SQLDA

The SQLDA is a collection of variables and pointers that provide column information about data being transmitted to the client application.

Note The SQLDA is an IBM standard. See the *IBM DB2 SQL Reference* for more information.

This appendix discusses the following topics:

- SQLDA variables and fields
- SQLDA datatypes
- Writing a SQLDA
- Sample COBOL II SQLDA
- Sample C SQLDA

SQLDA variables and fields

A SQLDA consists of four variables (*SQLDAID*, *SQLDABC*, *SQLN*, and *SQLD*), followed by an arbitrary number of SQLVARs. A SQLVAR is a structure containing five fields.

The following table describes the SQLDA variables.

Table G-1: SQLDA variables

This SQLDA variable:	Performs this function:
SQLDAID	Contains an eye catcher of "SQLDA" for use in storage dumps
SQLDABC	Contains the length of the SQLDA, equal to SQLN*44+16
SQLN	Contains the total number of occurrences of SQLVAR
SQLD	Indicates the number of columns described by occurrences of SQLVAR

Each occurrence of SQLVAR describes one column of the result row you are sending to the client application. The following table describes the five fields that each occurrence of SQLVAR contains.

Table G-2: SQLDA fields

This SQLDA field:	Performs this function:
SQLTYPE	Contains a 3-digit value that represents the datatype of the column and whether or not it allows null values. Table G-3 on page 151 contains the valid datatype values.
SQLLEN	Contains the external length of a value from the column.
SQLDATA	Contains the address of the data being transmitted
SQLIND	Contains the address of an indicator, which tells whether the column is nullable. Use a value less than zero if null.
SQLNAME	Contains the name or label of the column, or a string of length zero if the name or label does not exist.
SQLNAMEL	Contains the length of the column.

SQLDA datatypes

The following table contains the SQLDA datatypes and their 3-digit values. Each datatype has two available values to indicate whether an occurrence of the datatype allows nulls. (For up-to-date information, see the current SQL manual.

Table G-3: SQLDA datatypes

Datatype	Nulls not allowed	Nulls allowed
DATE	384	385
TIME	388	389
TIMESTAMP	392	393
CHAR VARIABLE LENG	448	449
CHAR FIXED LENGTH	452	453
CHAR LONG VARIABLE	456	457
FLOATING-POINT	480	481
DECIMAL	484	485
LARGE INTEGER	496	497
SMALL INTEGER	500	501

Writing a SQLDA

To write a model SQLDA definition, perform the following steps:

1 In the WORKING-STORAGE section of the RSP, include a SQLDA with a SQLVAR definition for each column you send in your result.

Note Sybase APIs use pointers; COBOL can only handle setting pointers in its linkage section.

2 Include a description of the SQLDA template.

The SQLDA template and the description go in the LINKAGE SECTION so they can be accessed by programs outside the RSP, such as MainframeConnect.

3 Optionally, re-calculate the size of your SQLDA definition or as an alternative, you can have the compiler do this for you with (LENGTH OF).

For an example of the compiler alternative, see Appendix B, “MODELRSP DB2 Output Pipe Sample RSP” in the 1100-TEST-SQLDA paragraph.

4 Allocate storage for the model SQLDA definition and set a pointer to that address.

For an example of this, see Appendix B, “MODELRSP DB2 Output Pipe Sample RSP” in the 1200-GET-STORAGE paragraph.

5 Move the model SQLDA definition residing in WORKING-STORAGE into the template SQLDA (in the allocated storage in the linkage section).

For an example of this, see Appendix B, “MODELRSP DB2 Output Pipe Sample RSP” in the 1300-SET-ADDRESSES paragraph.

Sample COBOL II SQLDA

```
*****************************************************************
 * The following sample description of the SQLDA is for COBOL II.
 * A complete description of each field and its purpose may be
 * found in the "DB2 SQL Reference."  Note that SQLDABC (SQLDA
 * Byte Count) may be initialized with:
 *
 * MOVE LENGTH OF SQLDA TO SQLDABC.
 *****************************************************************
 01  SQLDA.
     03  SQLDAID                    PIC X(8).
     03  SQLDABC                    PIC S9(8) COMP.
     03  SQLN                       PIC S9(4) COMP.
     03  SQLD                       PIC S9(4) COMP.
     03  SQLVAR                     OCCURS 0 TO 300 TIMES
                                    DEPENDING ON SQLN.
         05  SQLTYPE               PIC S9(4) COMP.
         05  SQLLEN                PIC S9(4) COMP.
         05  SQLDATA               USAGE IS POINTER.
         05  SQLIND                USAGE IS POINTER.
         05  SQLNAME.
             07  SQLNAMELENGTH     PIC S9(4) COMP.
             07  SQLNAMEVALUE      PIC X(30).
```

Sample C SQLDA

```
/*
   Sample SQLDA declaration and #defines for all DB2 datatypes.
 */
#ifndef SQLDA_DEF
 #define SQLDA_DEF
 struct SQLDA {
   unsigned char sqldaid[8];
```

```
   long sqldabc;
   short sqln;
   short sqld;
   struct sqlvar {
     short sqltype;
     union {
       short sqllen;
       struct {
         unsigned char precision;
         unsigned char scale;
       } SQLDECIMAL;
     } SQLLEN;
     unsigned char *sqldata;
     short *sqlind;
     struct sqlname {
       short length;
       unsigned char data [30];
     } sqlname;
 } sqlvar[0];
 };
#define DATE 384           /* SQLTYPE for DATE              */
 #define NDATE 385          /* SQLTYPE for DATE w/NULL       */
 #define TIME 388           /* SQLTYPE for TIME              */
 #define NTIME 389          /* SQLTYPE for TIME w/NULL       */
 #define TIMESTAMP 392      /* SQLTYPE for TIMESTAMP         */
 #define NTIMESTAMP 393     /* SQLTYPE for TIMESTAMP W/NULL  */
 #define VARCHAR 448        /* SQLTYPE for VARCHAR           */
 #define NVARCHAR 449       /* SQLTYPE for VARCHAR w/NULL    */
 #define CHAR 452           /* SQLTYPE for CVARCHAR          */
 #define NCHAR 453          /* SQLTYPE for VARCHAR w/NULL    */
 #define LONGVARCHAR 456    /* SQLTYPE for LONG VARCHAR      */
 #define NLONGVARCHAR 457   /* SQLTYPE for LVARCHAR w/ NULL  */
 #define FLOAT 480          /* SQLTYPE for FLOAT             */
 #define NFLOAT 481         /* SQLTYPE for FLOAT  w/ NULL    */
 #define DECIMAL 48         /* SQLTYPE for DECIMAL           */
 #define NDECIMAL 485       /* SQLTYPE for DECIMAL w/ NULLS  */
 #define INTEGER 496        /* SQLTYPE for INTEGER           */
 #define NINTEGER 497       /* SQLTYPE for INTEGER w/ NULL   */
 #define SMALLINT 500       /* SQLTYPE for SMALLINT Sa          */
 #define NSMALLINT 501      /* SQLTYPE for SMALL w/ NULL  Sa    */
#endif
```

Glossary

access management — A DirectConnect feature that provides connectivity to non-Sybase targets.

access service — The named set of properties, used with a DirectConnect Access Service Library, to which clients connect. Each DirectConnect Server can have multiple services.

access service library — A component of DirectConnect. A service library that provides access to non-Sybase data contained in a database management system or other type of repository. Each such repository is called a "target." Each access service library interacts with exactly one target and is named accordingly. See also **service library**.

ACSLIB — See **access service library**.

Adaptive Server Enterprise — The server in the Sybase Client-Server architecture. It manages multiple databases and multiple users, tracks the actual location of data on disks, maintains mapping of logical data description to physical data storage, and maintains data and procedure caches in memory.

administrative service library — A service library that provides remote management capabilities and server-side support. It supports a number of remote procedures (invoked as RPC requests) that enable remote DirectConnect management. See also **remote procedure call** and **service library**.

ADMLIB — See **administrative service library**.

American Standard Code for Information Interchange — The standard code used for information interchange among data processing systems, data communication systems, and associated equipment. The code uses a coded character set consisting of seven-bit coded characters (eight bits including a parity check).

API — See **application program interface**.

application program interface — A functional interface, supplied by an operating system or other licensed program, that allows an application program written in a high-level language to use specific data or functions of the operating system or the licensed program.

ASCII — See **American Standard Code for Information Interchange**.

ASE/CIS Adaptive Server Enterprise/ Component Integration Services (formerly OmniConnect). An add-on product for Adaptive Server that provides a Transact-SQL interface to external data sources, including host data files and tables in other database systems. OmniConnect replaces OmniSQL Gateway and OmniSQL Server.

bulk copy transfer A transfer method in which multiple rows of data are inserted into a table in the target database. See also **transfer**. Compare with **destination-template transfer**.

call level interface A programming style that calls database functions directly from the top level of the code. Usually it is contrasted with embedded SQL. See also **dynamic SQL** and **embedded SQL**.

catalog A system table that contains information about objects in a database, such as tables, views, columns, and authorizations.

catalog stored procedure A stored procedure that provides information about tables, columns, and authorizations. It is used in SQL generation and application development. See also **stored procedures**.

character set A set of specific (usually standardized) characters with an encoding scheme that uniquely defines each character. ASCII is a common character set.

CLI See **call level interface**.

client In client/server systems, the part of the system that sends requests to servers and processes the results of those requests. See also **client/server**. Compare with **server**.

client application Software that is responsible for the user interface, including menus, data entry screens, and report formats. See also **client/server**.

Client-Library A library of routines that is part of Open ClientConnect™. See also **Open ClientConnect**.

client-server An architecture in which the client is an application that handles the user interface and local data manipulation functions, while the server provides data processing access and management for multiple clients. See also **client**, **client application**, and **server**.

clustered index An index in which the physical order and the logical (indexed) order is the same. Compare with **nonclustered index**.

codeset See **character set.**

commit — An instruction to a database to make permanent all changes made to one or more database files since the last commit or rollback operation and to make the changed records available to other users. Compare with **rollback**.

commitment control — A means of grouping file operations that allows a group of database changes to be processed as a single unit, or the removal of a group of database changes as a single unit. See also commit, rollback

configuration file — A file that specifies the characteristics of a system or subsystem.

configuration set — A section into which service library configuration files are divided.

connection specification — Information required to make an Open ClientConnect or Open ServerConnect™ connection. The connection specification consists of the server name, platform, Net-Library™ driver name, and address information required by the Net-Library driver being used.

conversion — The transformation between values that represent the same data item but which belong to different datatypes. Information can be lost due to conversion because accuracy of data representation varies among different datatypes.

CSP — See **catalog stored procedure**.

CT-Library — See **Client-Library**.

data definition language — A language for describing data and data relationships in a database.

database management system — A computer-based system for defining, creating, manipulating, controlling, managing, and using databases.

datatype — A keyword that identifies the characteristics of stored information on a computer.

DB-Library — A Sybase and Microsoft API that allows client applications to interact with ODS applications. See also **application program interface**.

DBMS — See **database management system**.

DDL — See **data definition language**.

destination-template transfer — A transfer method in which source data is briefly put into a template where the user can specify that some action be performed on it before execution against a target database. See also **transfer**. Compare with **bulk copy transfer**.

direct resolution — A type of service name resolution that relies upon a client application specifying the exact name of the service to be used. See also **service name resolution**. Compare with **service name redirection**.

DirectConnect — A Sybase Open Server application that provides access management for non-Sybase databases, copy management, and remote systems management. Each DirectConnect consists of a server and one or more service libraries to provide access to a specific data source. DirectConnect replaces the MDI Database Gateway™ and the OmniSQL Access Module™.

DirectConnect Anywhere™ — A Sybase solution that gives client applications ODBC data access. It combines the functionality of the DirectConnect architecture with ODBC to provide dynamic SQL access to target data, as well as the ability to support stored procedures and text and image pointers.

DirectConnect Manager — A Sybase application for Microsoft Windows that provides remote management capabilities for DirectConnect products. These capabilities include starting, stopping, creating, and copying services.

DirectConnect Server — The component that provides general management and support functions (such as log file management) to service libraries.

DirectConnect Service — A named set of properties, used with a DirectConnect Service Library, to which clients connect.

DirectConnect Service Library — The component that provides a set of functions within the DirectConnect Server environment.

dll — See **dynamic link library**.

dynamic link library — A file containing executable code and data bound to a program at load time or run time, rather than during linking. The code and data in a dynamic link library can be shared by several applications simultaneously.

dynamic SQL — A term pertaining to the preparation and processing of SQL source statements within a program while the program runs. The SQL source statements are contained in host-language variables rather than coded directly into the application program. Compare with **static SQL**.

embedded SQL — A SQL statement embedded within a source program and prepared before the program executes. After it is prepared, the statement itself does not change, although values of host variables specified within the statement can change.

event handler — A device that processes requests and manages client communication.

global variable — System-defined variables that DirectConnect or the client application updates on an ongoing basis.

globalization — The combination of internationalization and localization. See also **internationalization**, **localization**.

interfaces file — An operating system file that must be available on each machine from which connections to DirectConnect Anywhere or other Sybase products are made. Each entry in the file determines how the host client software connects to the Sybase product.

internationalization — The process of extracting locale-specific components from the source code and moving them into one or more separate modules, making the code culturally neutral so it can be localized for a specific culture. See also **globalization**. Compare with **localization**.

keyword — A word or phrase reserved for exclusive use by Transact-SQL.

localization — The process of preparing an extracted module for a target environment, in which messages are displayed and logged in the user's language. Numbers, money, dates, and time are represented using the user's cultural convention, and documents are displayed in the user's language. See also **globalization**. Compare with **internationalization**.

MDI Database Gateway — An MDI legacy product that gives client applications access to supported data sources, such as AS/400 and DB2.

Net-Library — A Sybase product that lets PC applications become clients of Adaptive Server or Open Server. See also **client**, **Open Server**.

nonclustered index — An index that stores key values and pointers to data. Compare with **clustered index**.

ODBC — See **Open Database Connectivity**.

ODS — See **Open Data Services**.

OmniConnect — The CIS functionality of ASE has incorporated the functionality of OmniConnect and is referred to as ASE/CIS. See **ASE/CIS**.

Open Client — A Sybase product that provides customer applications, third-party products, and other Sybase products with the interfaces required to communicate with Open Server and Open Server applications.

Open ClientConnect — A Sybase product that provides capability for the mainframe to act as a client to LAN-based resources.

Open Data Services — A product that provides a framework for creating server applications that respond to DB-Library clients. See also **DB-Library**.

Open Database Connectivity — A Microsoft API that allows access to both relational and nonrelational databases.

Open Server	A Sybase product that provides the tools and interfaces required to create a custom server.
Open ServerConnect	A Sybase product that provides capability for programmatic access to mainframe data.
parameter	A variable with a constant value for a specified application that can denote the application. Compare with **property**.
Partner Certification Reports	Sybase publications that certify third-party or Sybase products to work with other Sybase products.
precision	The maximum number of digits that can be represented in a decimal, numeric, or float column.
precision minus scale	The number of digits to the left of the decimal point.
primary database	In transfer processing, the database accessed by the access service in a transfer statement. Compare with **secondary database**.
property	A setting for a server or service that defines characteristics, such as how events are logged or how datatypes are converted. Compare with **parameter**.
protocol	A set of rules that governs the behavior of the computers communicating on a network.
Registry	The part of the Windows NT operating system that holds configuration information for a particular machine.
relational operators	Operators supported in search conditions.
relops	See **relational operators**.
remote procedure call	A stored procedure executed on a different server from the one onto which a user is logged or on which the initiating application resides.
remote systems management	A feature that allows a System Administrator to manage multiple DirectConnect Servers and multiple services from a client.
request	One or more database operations an application sends as a unit to the database. During a request, the application gives up control to the DBMS and waits for a response. See also **commit**, **rollback**, and **unit of work**.
rollback	An instruction to a database not to implement the changes requested in a unit of work and to return to the pretransaction state. See also **transaction** and **unit of work**. Compare with **commit**.

RPC — See **remote procedure call**.

scale — The maximum number of digits that can be stored to the right of the decimal point by a numeric or decimal datatype.

secondary connection — The connection specified in the transfer statement. It represents anything that can be accessed using Open ClientConnect, such as Adaptive Server or another access service.

secondary database — In transfer processing, the supported database that is specified in the transfer statement. Compare with **primary database**.

server — A functional unit that provides shared services to clients over a network. See also **client/server**. Compare with **client**.

server process ID — A positive integer that uniquely identifies a client connection to the server.

service — A functionality available to DirectConnect applications. It is the pairing of a service library and a set of specific configuration properties.

service library — A set of configuration properties that determines service functionality. Examples of service libraries include access service libraries and administrative service libraries. See also **access service library** and **administrative service library**.

service name redirection — A type of service name resolution that allows a System Administrator to map alternative connections to services. See also **service name resolution**. Compare with **direct resolution**.

service name redirection file — The default name of the file used for the service name redirection feature. See also **service name redirection**.

service name resolution — The DirectConnect Server mapping of an incoming service name to an actual service. See also **direct resolution**, **service name redirection**.

SNRF — See **service name redirection file**.

SPID — See **server process ID**.

SQL — See **structured query language**.

SQL descriptor area — A set of variables used in the processing of SQL statements.

SQL stored procedure — A single SQL statement that is statically bound to the database. See also **stored procedures**.

SQLDA — See **SQL descriptor area**.

sqledit	A utility for creating and editing *sql.ini* files and file entries.
sql.ini	The interfaces file containing definitions for each DirectConnect Server to which a workstation can connect. See also **interfaces file**.
statement	A single SQL operation, such as select, update, or delete.
static SQL	SQL statements that are embedded within a program and prepared before the program runs. The statement itself does not change, although values of host variables specified by the statement can change. Compare with **dynamic SQL**.
stored procedures	A collection of SQL statements and optional control-of-flow statements stored under a particular name. See also **Catalog Stored Procedure**, **SQL stored procedure**, and **system stored procedure**.
structured query language	An IBM industry-standard language for processing data in a relational database.
System Administrator	The user in charge of server system administration. For DirectConnect, the user responsible for installing and maintaining DirectConnect Servers and DirectConnect Service Libraries.
system stored procedure	A Sybase-supplied store procedure that returns information about the access service and the target database. See also **stored procedures**.
table	An array of data or a named data object that contains a specific number of unordered rows. Each item in a row can be identified unambiguously by means of one or more arguments.
Tabular Data Stream	An application-level protocol that Sybase clients and servers use to communicate.
target	A system, program, or device that interprets and replies to requests received from a source.
target database	The database to which DirectConnect transfers data or performs operations on specific data.
TDS	See **Tabular Data Stream**.
transaction	An exchange between a program on a local system and a program on a remote system that accomplishes a particular action or result.
Transact-SQL	A Sybase enhanced version of the SQL database language used to communicate with Adaptive Server.

transfer A DirectConnect feature that allows users to move data or copies of data from one database to another. See also **bulk copy transfer** and **destination-template transfer**.

trigger A form of stored procedure that automatically executes when a user issues a change statement to a specified table.

T-SQL See **Transact-SQL**.

unit of work One or more database operations grouped under a commit or rollback. A unit of work ends when an application commits or rolls back a series of requests, or when the application terminates. See also **commit**, **rollback**, and **transaction**.

view An alternative representation of data from one or more tables. A view can include all or some of the columns contained in the table or tables on which it is defined.

wildcard A special character that represents a range of characters in a search pattern.

Index

Symbols

A

B

C

D

E

F

G

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W