Transaction Processing Facility

# Application Requester User's Guide

*Version 4 Release 1*

SH31-0133-04

Transaction Processing Facility

# Application Requester User's Guide

*Version 4 Release 1*

SH31-0133-04

**Note!**
Before using this information and the product it supports, be sure to read the general information under “Notices” on page ix.

**Fifth Edition (June 2002)**

This is a major revision of, and obsoletes, SH31-0133-03 and all associated technical newsletters.

This edition applies to Version 4 Release 1 Modification Level 0 of IBM Transaction Processing Facility, program number 5748-T14, and to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct edition for the level of the product.

IBM welcomes your comments. Address your comments to:

IBM Corporation
TPF Systems Information Development
Mail Station P923
2455 South Road
Poughkeepsie, NY 12601-5400
USA

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

# Contents

# Figures

# Tables

# Notices

References in this book to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service in this book is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Department 830A
Mail Drop P131
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Any pointers in this book to non-IBM Web sites are provided for convenience only and do not in any way serve as an endorsement. IBM accepts no responsibility for the content or use of non-IBM Web sites specifically mentioned in this book or accessed through an IBM Web site that is mentioned in this book.

## Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AIX
APPN
C/370
CICS
DATABASE 2
DB2
Distributed Relational Database Architecture
DRDA
ESCON
IBM
Language Environment

MVS
OS/2
PS/2
RACF
RISC System/6000
RS/6000
SAA
S/370
System/370
Systems Application Architecture
VTAM

Other company, product, and service names may be trademarks or service marks of others.

# About This Book

This book describes the TPF Application Requester (TPFAR) feature; it includes information about installing the TPFAR feature and writing TPFAR application programs using the Structured Query Language (SQL), a common interface. The TPFAR feature supports the IBM System Application Architecture.

In this book, abbreviations are often used instead of spelled-out terms. Every term is spelled out at first mention followed by the all-caps abbreviation enclosed in parentheses; for example, Systems Network Architecture (SNA). Abbreviations are defined again at various intervals throughout the book. In addition, the majority of abbreviations and their definitions are listed in the master glossary in the *TPF Library Guide*.

## Who Should Read This Book

The *TPF Application Requester User's Guide* is for TPF programmers who need to install the TPFAR feature and write TPFAR application programs using the common Structured Query Language (SQL) interface.

## Conventions Used in the TPF Library

The TPF library uses the following conventions:

| Conventions | Examples of Usage |
|---|---|
| *italic* | Used for important words and phrases. For example:<br>A *database* is a collection of data.<br>Used to represent variable information. For example:<br>Enter **ZFRST STATUS MODULE** *mod*, where *mod* is the module for which you want status. |
| **bold** | Used to represent text that you type. For example:<br>Enter **ZNALS HELP** to obtain help information for the ZNALS command.<br>Used to represent variable information in C language. For example:<br>**level** |
| `monospaced` | Used for messages and information that displays on a screen. For example:<br>`PROCESSING COMPLETED`<br>Used for C language functions. For example:<br>`maskc`<br>Used for examples. For example:<br>`maskc(MASKC_ENABLE, MASKC_IO);` |
| ***bold italic*** | Used for emphasis. For example:<br>You ***must*** type this command exactly as shown. |
| **Bold underscore** | Used to indicate the default in a list of options. For example:<br>**Keyword=OPTION1** \| **DEFAULT** |

| Conventions | Examples of Usage |
|---|---|
| Vertical bar \| | Used to separate options in a list. (Also referred to as the OR symbol.) For example:<br>**Keyword=Option1** \| **Option2**<br>**Note:** Sometimes the vertical bar is used as a *pipe* (which allows you to pass the output of one process as input to another process). The library information will clearly explain whenever the vertical bar is used for this reason. |
| CAPital LETters | Used to indicate valid abbreviations for keywords. For example:<br>KEYWord=*option* |
| Scale | Used to indicate the column location of input. The scale begins at column position 1. The plus sign (+) represents increments of 5 and the numerals represent increments of 10 on the scale. The first plus sign (+) represents column position 5; numeral 1 shows column position 10; numeral 2 shows column position 20 and so on. The following example shows the required text and column position for the image clear card.<br>`\|...+....1....+....2....+....3....+....4....+....5....+....6....+....7...`<br>`LOADER  IMAGE CLEAR`<br>**Notes:**<br>1. The word LOADER must begin in column 1.<br>2. The word IMAGE must begin in column 10.<br>3. The word CLEAR must begin in column 16. |

## Related Information

A list of related information follows. For information on how to order or access any of this information, call your IBM representative.

## IBM Transaction Processing Facility (TPF) 4.1 Books

- *TPF ACF/SNA Data Communications Reference*, SH31-0168
- *TPF ACF/SNA Network Generation*, SH31-0131
- *TPF Application Programming*, SH31-0132
- *TPF Application Requester User's Guide*, SH31-0133
- *TPF C/C++ Language Support User's Guide*, SH31-0121
- *TPF General Macros*, SH31-0152
- *TPF Library Guide*, GH31-0146
- *TPF Main Supervisor Reference*, SH31-0159
- *TPF Operations*, SH31-0162
- *TPF System Generation*, SH31-0171
- *TPF Transmission Control Protocol/Internet Protocol*, SH31-0120.

## Miscellaneous IBM Books

- *Character Data Representation Architecture Reference and Registry*, SC09-2190
- *IBM DATABASE 2 Administration Guide*, (order the correct version and release for your installation)
- *IBM DATABASE 2 Application Programming and SQL Guide*, (order the correct version and release for your installation)
- *IBM DATABASE 2 Command and Utility Reference*, (order the correct version and release for your installation)

- *IBM DATABASE 2 SQL Reference*, (order the correct version and release for your installation)
- *Distributed Data Management Architecture Reference*, SC21-9526
- *Distributed Relational Database Architecture Reference*, SC26-4651
- *Distributed Relational Database Connectivity Guide*, SC26-4783
- *ESCON User's Guide and Service Information*, SC31-8197
- *SNA Channel Connectivity for AIX*, GC31-8201
- *VTAM Resource Definition Reference*, (order the correct version and release for your installation).

## Online Information

- *Messages (Online)*
- *Messages (System Error and Offline)*.

## How to Send Your Comments

Your feedback is important in helping to provide the most accurate and highest quality information. If you have any comments about this book or any other TPF information, use one of the methods that follow. Make sure you include the title and number of the book, the version of your product and, if applicable, the specific location of the text you are commenting on (for example, a page number or table number).

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

- If you prefer to send your comments electronically, do either of the following:
  - Go to http://www.ibm.com/tpf/pubs/tpfpubs.htm.

    There you will find a link to a feedback page where you can enter and submit comments.
  - Send your comments by e-mail to tpfid@us.ibm.com
- If you prefer to send your comments by mail, address your comments to:

  IBM Corporation
  TPF Systems Information Development
  Mail Station P923
  2455 South Road
  Poughkeepsie, NY 12601-5400
  USA
- If you prefer to send your comments by FAX, use this number:
  - United States and Canada: 1 + 845 + 432 + 9788
  - Other countries: (international code) + 845 + 432 +9788

# The TPF Application Requester (TPFAR) Feature

## Introduction

The TPF Application Requester (TPFAR) feature supports IBM's Distributed Relational Database Architecture (DRDA) strategy for database distribution using the remote unit of work method of distributed access. DRDA is a relational database connection architecture, consisting of protocols for communications between an application and a remote database, and for communications between databases. Distributed access involves working with data that is located on remote systems. A relational database is a database in which the data is organized and accessed according to relations. Structured Query Language (SQL) is a programming language used to define relational data, access relational data, and control access to relational data resources.

The TPFAR feature permits you to share data between any DRDA- compliant database servers, such as a DATABASE 2 (DB2) relational database and a TPF application using the SQL interface. The component of DB2 that allows the TPFAR connection is known as an application server (AS). For more information about the concept of an application server, see "Access to Remote Data" or the *Distributed Relational Database Architecture Reference*. An LU 6.2 conversation or a TCP/IP connection is used to connect a DB2 subsystem with the TPFAR feature. By using the TPFAR feature, a TPF application can directly access and update the information residing on the remote DB2 subsystem.

## Access to Remote Data

The remote unit of work is a method of accessing distributed relational data in which users or applications can, within a single unit of work, read and update one system using multiple Structured Query Language (SQL) statements. For a complete list of SQL commands that TPFAR supports, see "SQL Commands Supported by TPFAR" on page 101. To better understand how TPFAR fits the concept of remote data access, DRDA needs to be explained.

There are two parts of accessing remote data defined in DRDA:

- The system requesting the data is generically known as the **application requester (AR)**.
- The system that needs to service the request for data is generically known as the **application server (AS)**.

Figure 1 on page 2 shows two non-TPF environments, both with a full DRDA implementation.

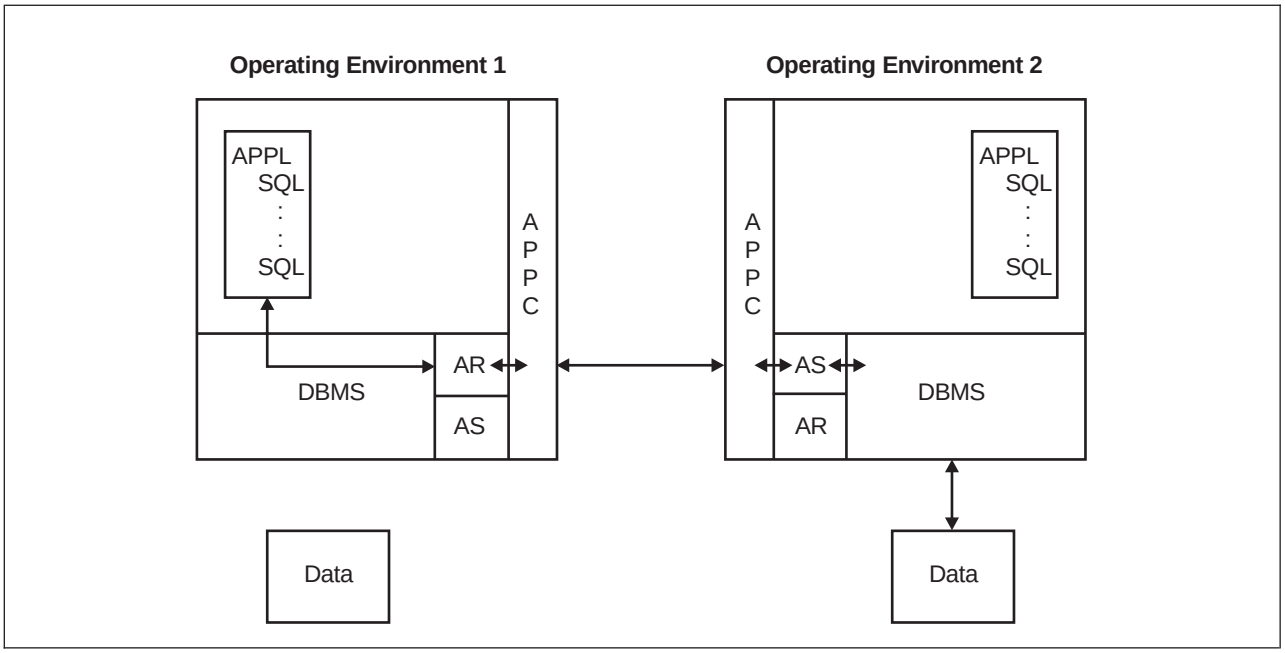


*Figure 1. Overview of Remote Data*

Operating Environment 1 is running with an SQL application (APPL). When a remote SQL request is made, the Operating Environment 1 application requester (AR) of the database management system (DBMS) takes control. The DBMS is a software system that has a catalog describing the data it manages. The DBMS controls access to the data stored within it. The AR then uses the Advanced Peer-to-Peer Communications (APPC) session to route the request to the application server (AS) of the remote database management system (DBMS) of Operating Environment 2 where the data resides.

When the Operating Environment 2 AS has collected the requested information, this information is passed back through APPC to the AR of Operating Environment 1 running the application. The AR then returns the data back to the application program on Operating Environment 1.

Operating Environment 2 can also have local applications accessing the same database. Because Operating Environment 1 and 2 both have implemented the AR and AS, the application in Operating Environment 2 can also access the data residing on Operating Environment 1.

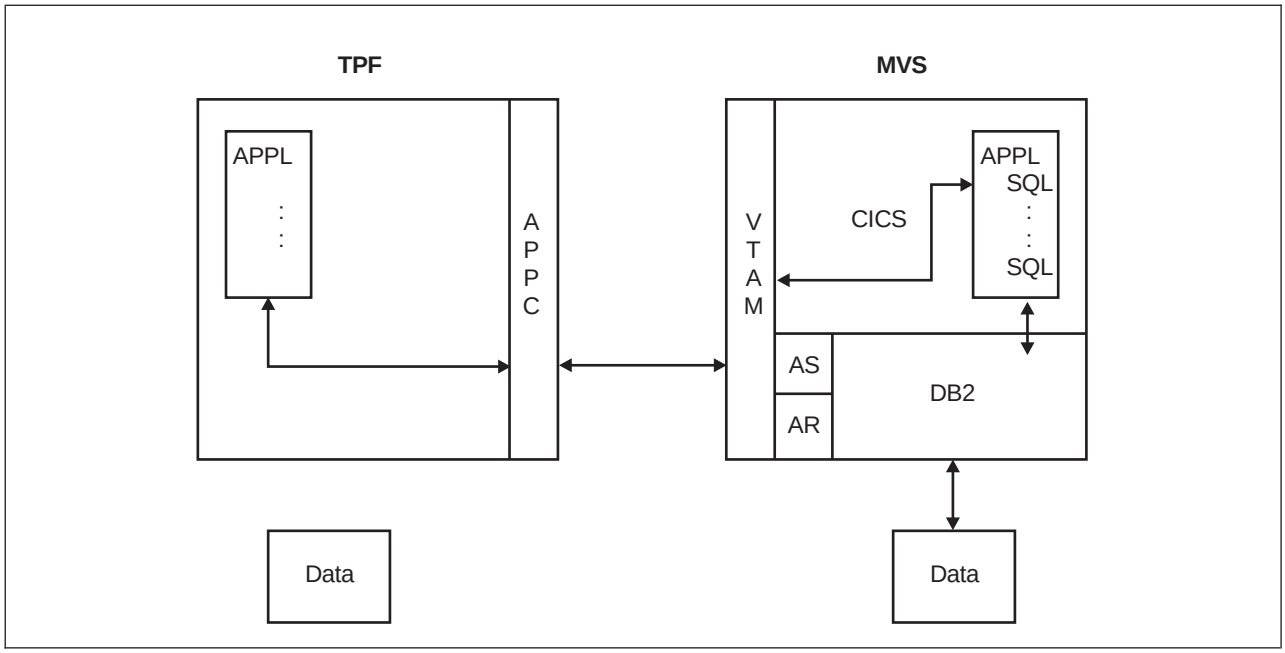


*Figure 2. Moving TPF Data to a DB2 Database without TPFAR*

## Moving TPF Data to a DB2 Database without TPFAR

Figure 2 shows one method of moving TPF data to a DB2 database without TPFAR. A TPF application program takes the data from TPF and, using TPF/APPC, sends the data to the Customer Information Control System (CICS) subsystem on MVS. An MVS CICS application receives the data from the TPF application and issues the necessary SQL commands to the DB2 subsystem to do the requested work.

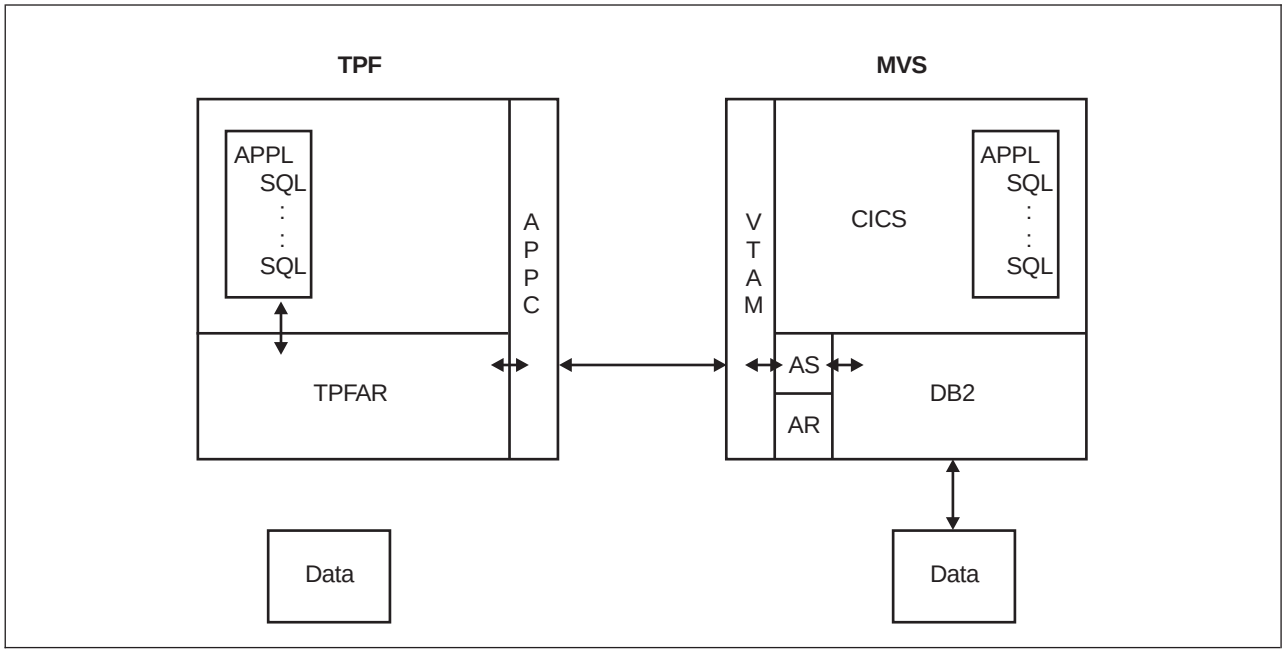


*Figure 3. Moving TPF Data to and from a DB2 Database Using TPFAR and LU 6.2*

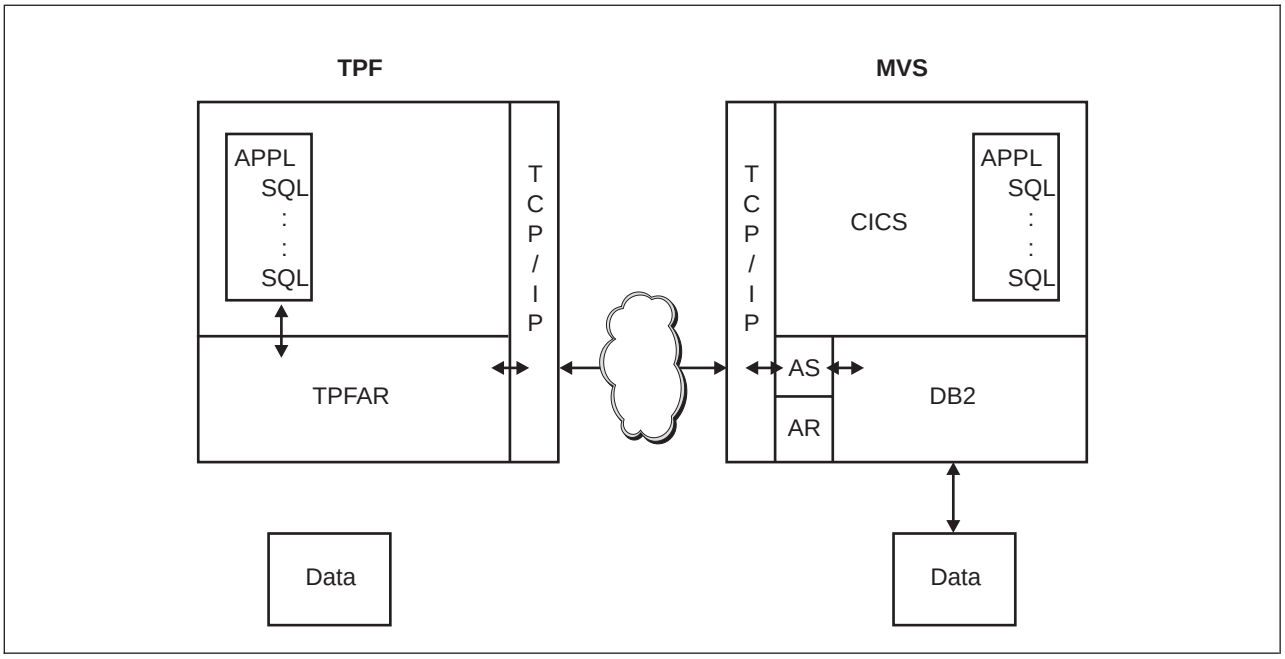


*Figure 4. Moving TPF Data to and from a DB2 Database Using TPFAR and TCP/IP*

## Moving TPF Data to and from a DB2 Database Using TPFAR

TPFAR implements the application requester (AR) part of the DRDA. Thus, with TPFAR, a TPF application can directly access and update data residing on the remote DB2 subsystem.

Figure 3 on page 3 shows how TPFAR moves data from TPF to a DB2 database using LU 6.2. Figure 4 shows how TPFAR moves data from TPF to a DB2 database using TCP/IP.The TPF application program now includes the SQL commands that previously were in the CICS application program. When the TPF application makes an SQL request, TPFAR forwards the request to the application server (AS) on DB2. DB2 completes the requested work and returns the results to TPFAR. TPFAR then returns the results of the request back to the application.

TPFAR can also be used to move DB2 data to TPF.

**Note:** TPF does not implement the AS portion of the DRDA. Applications on remote operating systems cannot access data on TPF through an application requester/application server DRDA implementation because the database facilities, such as indexes, access logic, and data formats are contained in the TPF application programs. Therefore, remote requests for data on TPF must use a non-DRDA mechanism to communicate with the TPF application programs that manage the data.

## How TPFAR Works

With TPFAR, SQL commands can now be included in a TPF application program that is coded in assembler or C language. The SQL commands are removed from the application at the DB2 precompile time, then they are assembled or compiled and replaced with system calls. (DB2 precompiler is a prerequisite for TPFAR). Additional information on the DB2 precompiler is found in “Preparing an Application” on page 32 and “Using the Same Cursor in Multiple Programs” on page 41. At

application execution time, the TPFAR code takes over when a system call is made and routes the request to the application server (DB2).

When DB2 has processed the request, the data is passed back to TPFAR, which places the data in the TPF application's host variables. (The formats of the data are determined by the host variables that are defined by the application). The application then regains control to continue processing.

## Methods for Using TPFAR in Your Enterprise

TPFAR can be used to enhance your enterprise in these processing areas:

- Transaction logging
- Bulk data transfer
- Single line query
- Block query.

**Note:** Within your enterprise, additional methods for using TPFAR are possible.

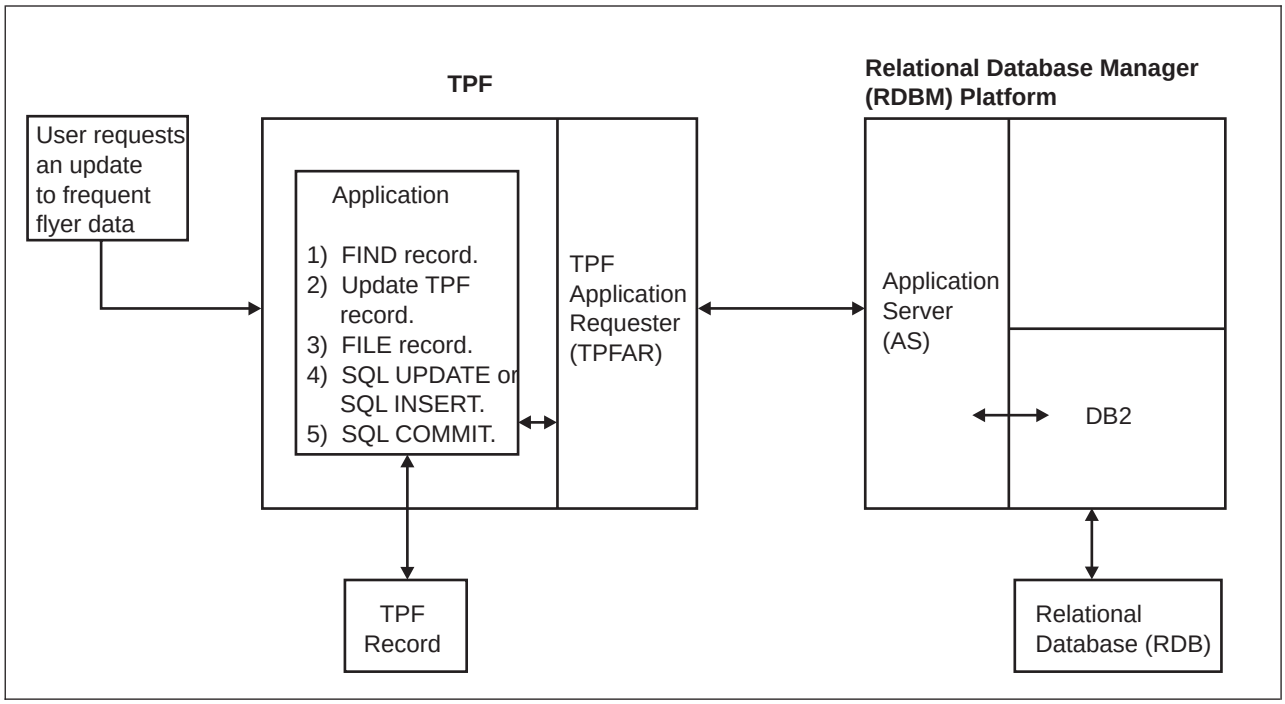


*Figure 5. Transaction Logging with a Frequent Flyer Database. Identical data resides on both the TPF and remote systems.*

## Transaction Logging

Transaction logging can be used when the data you are working with normally resides on TPF, but the decision support is done on the remote application server (AS). It can also be used if there is a requirement for the data on the AS to be current with the data on TPF. A transaction logging implementation can be used in which the TPF application issues the SQL INSERT or UPDATE command to the AS whenever the data on TPF is updated.[1] Thus, the application guarantees that the data on RDBM and TPF are identical. The remote unit of work method of accessing distributed relational data does not support a two phase commit. Therefore, if the

1. The SQL INSERT command inserts new rows into a table. The SQL UPDATE command changes the values of specified columns in rows of a table. See the *IBM DATABASE 2 Version 2 SQL Reference* for additional information on these commands.

data must be identical on both the relational database manager (RDBM) and TPF, the application must guarantee that the data is identical.

Figure 5 on page 5 shows how transaction logging can work with a frequent flyer database.

The user requests an update for frequent flyer data. The application finds the TPF record, makes the change and files it. Then, the application issues an SQL UPDATE or SQL INSERT command to make the same change to the remote relational database. In this case, the application maintains identical data on both TPF and the relational database manager (RDBM) at all times.

## Bulk Data Transfer

Bulk data transfer can be used when the data you are working with normally resides on TPF, but the decision support is done on the remote application server (AS). However, in this case, the data on the AS need not be current with the data on TPF, and the number of changes made to the data is high. A bulk data transfer implementation can be used in which the TPF application queues the data in a TPF file. At a specified time, another application reads all of the data from the TPF file and ships it to the AS using SQL INSERT and UPDATE commands. The savings occurs in that each application does not need to issue an SQL CONNECT and an SQL COMMIT command.[2] Instead, one application issues an SQL CONNECT command, does all of the inserts, and then issues the SQL COMMIT command. A detailed implementation of bulk data transfer is found in “Using Assembler Language with TPFAR” on page 79.

This method is also useful when only certain pieces of the data are needed on the remote DB2 subsystem. For example, on your DB2 subsystem you need the information on who flew on what flight. Until that flight takes off, this information is constantly changing. Keeping a DB2 database synchronized with the TPF database is costly. Instead, after a flight has taken off, a program can send the final information over to DB2 for processing either immediately or during off-peak hours.

Figure 6 on page 7 shows how bulk data transfer can work with a customer account database.

2. The SQL CONNECT command establishes a connection between an application and the application server, DB2. The SQL COMMIT command normally terminates a unit of work that is complete and saves the changes in the database. All locks are released. These changes can then be accessed by other users accessing the table. See the *IBM DATABASE 2 Version 2 SQL Reference* for additional information on these commands.

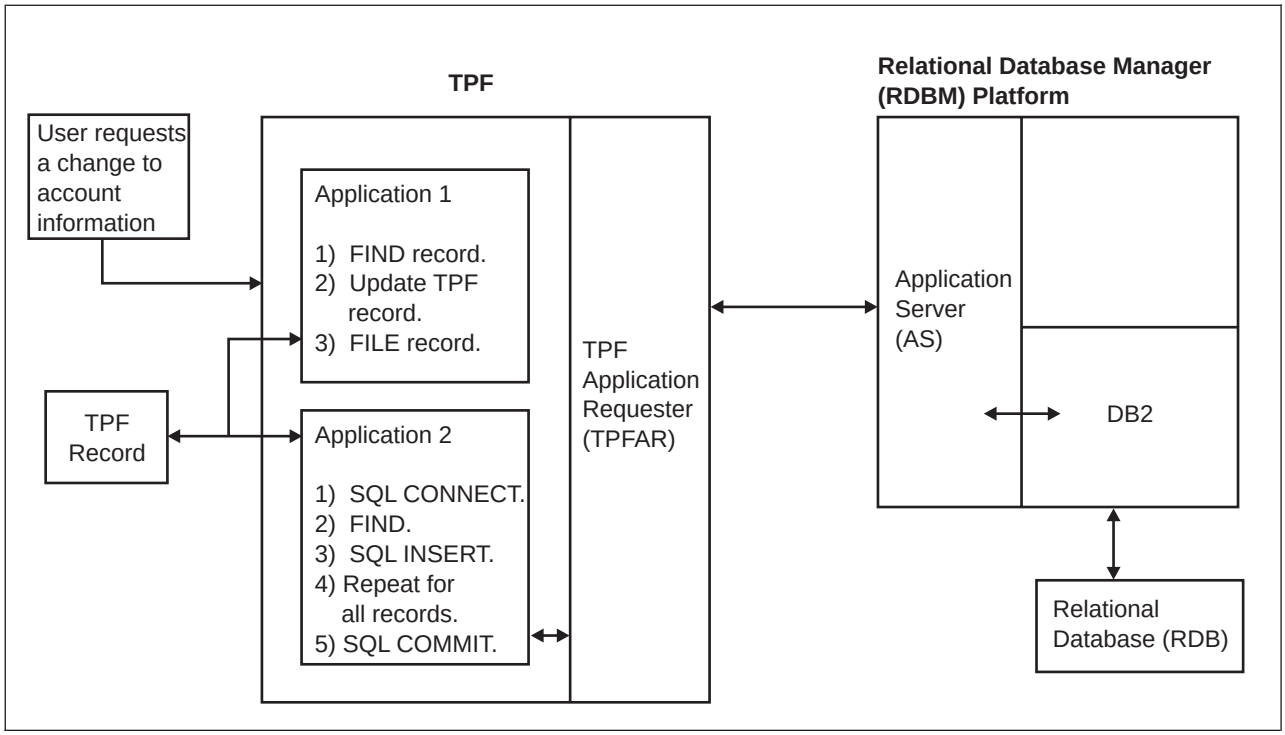


*Figure 6. Bulk Data Transfer. Large amounts of data are transferred to the remote system.*

The user requests a change to account information. The first application takes control and updates the TPF record with the new information and exits. A second application retrieves the record and issues an SQL INSERT command to log that this transaction occurred. A program can then be written on the remote system to gather statistical information on the types of transactions made by the user.

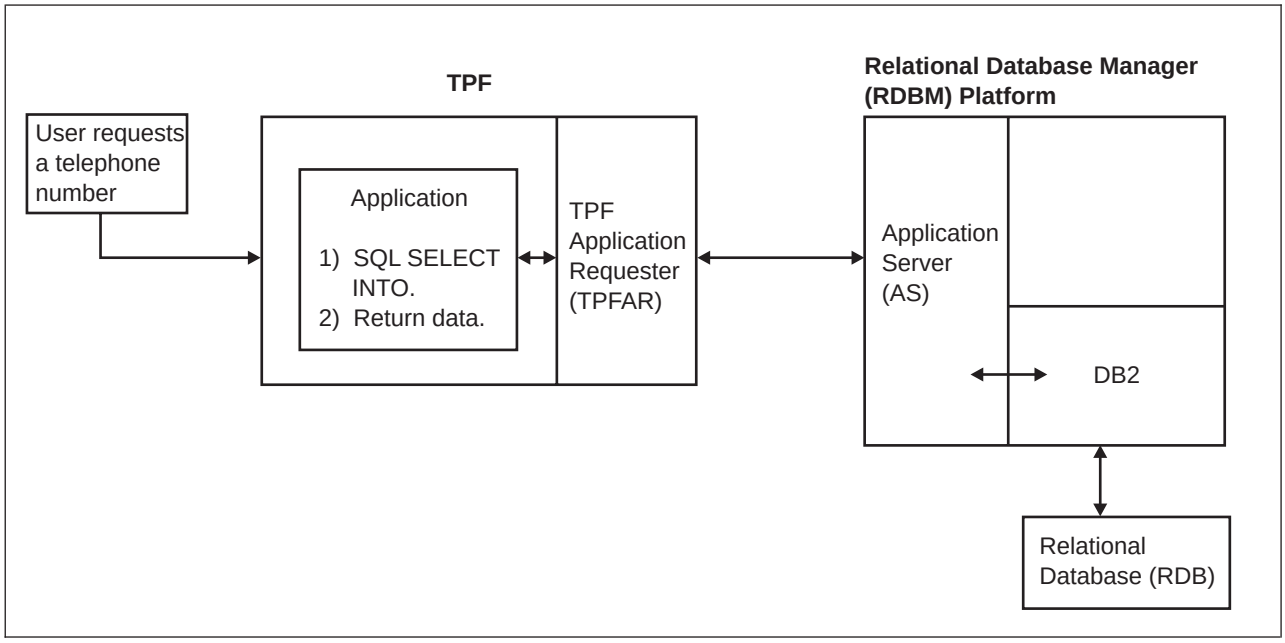


*Figure 7. Single Line Query. A single data record is retrieved from the remote system.*

## Single Line Query

Single line query can be used when there are a relatively low number of TPF transaction requests for data that normally reside on the remote application server (AS). A single query using the SQL SELECT INTO command can be issued to retrieve the needed information.[3]

Figure 7 on page 7 shows how single line query can work with a phone directory database.

The user requests a telephone number. The application issues an SQL SELECT INTO command to retrieve the information. The telephone number is returned to the application.

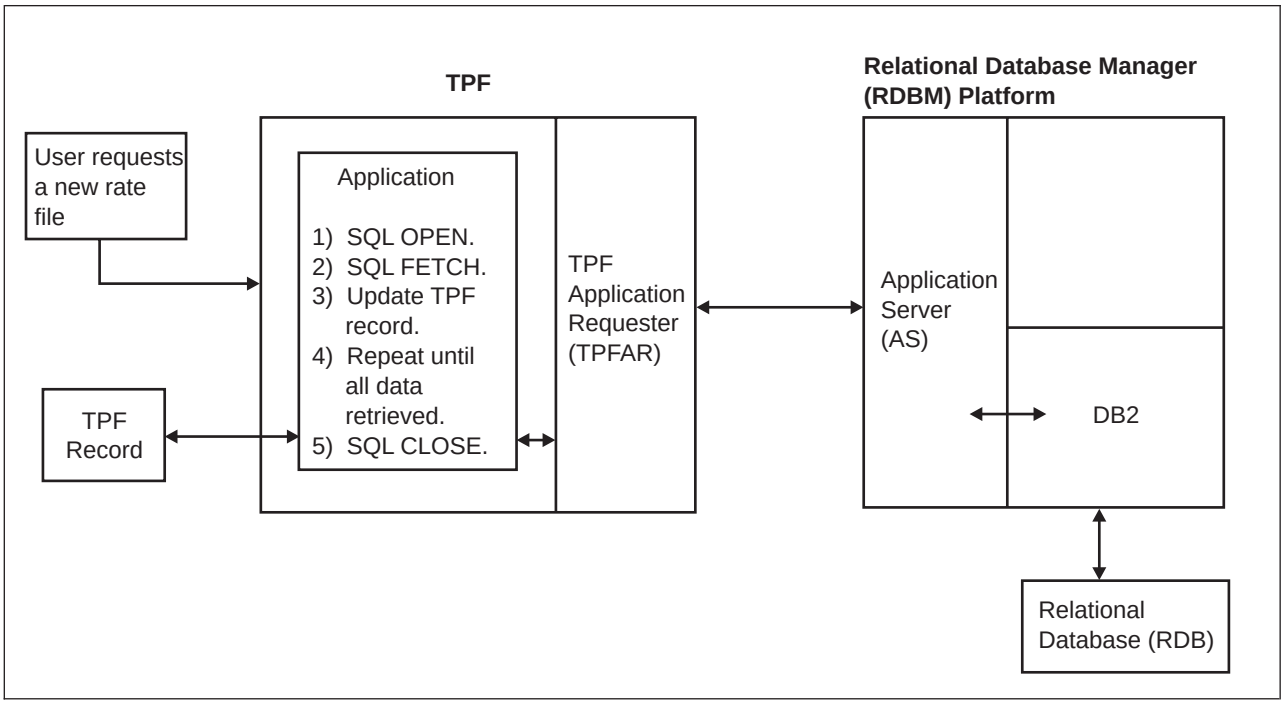


*Figure 8. Block Query. Many data records are retrieved from the remote system.*

## Block Query

A block query can be used when the data you are working with resides on the remote application server (AS), and a refresh of that data is needed on TPF. Using the SQL SELECT INTO command, a block query can be issued to retrieve the needed information.

Figure 8 shows how block query can work with a new rate file.

The user requests a refresh of the rate table. The application issues an SQL OPEN command to open a cursor to retrieve the rate information. The SQL FETCH command is then used to retrieve each individual row of data.[4] A cursor is a control

3. The SQL SELECT INTO command retrieves a certain column or columns of data to work on based upon requested conditions. See the *IBM DATABASE 2 Version 2 SQL Reference* for additional information on this command.

structure used by an application to retrieve, update, or point to information in a table. The SQL FETCH command is repeated until all of the information has been retrieved.

The main difference between the block query and single line query is that with block query, an SQL FETCH command goes across the network to DB2 only once to retrieve a block of data. From then on, the SQL FETCH takes place at the local level between the application and TPFAR. This is explained in more detail in “Request Unit Size Considerations” on page 28.

4. The SQL OPEN command initializes a cursor to fetch rows from its result table. The SQL FETCH command retrieves the next requested row in an answer set table. See the *IBM DATABASE 2 Version 2 SQL Reference* for additional information on these commands.

# Preparing Your Environment for TPF Application Requester

To install the TPF Application Requester (TPFAR) feature, you must make provisions in the following areas of your system:

- TPF system: Requires online and offline changes.
- If you are using LU 6.2 communications: Requires changes determined by the remote systems.
- Any DRDA-compliant database server, such as DATABASE 2 (DB2) system: Requires installation and communication changes.

Throughout this publication, the DB2 system is used to refer to any remote distributed relational database architecture (DRDA) level-1 compliant relational database.

You must carefully specify the various parameters required for the three different areas of your system when installing the TPFAR feature.

Because of the interdependency of these three areas of your system, certain parameter values specified in one area must be identically specified in other areas. Throughout this section, a parameter value specified in one example is carried forth into another example when the parameters must be identical. For instance, if there is a definition on the TPF system that requires the same value as a definition on the DB2 system, in both of the examples given, the same value is used.


A summary of the various parameters and their location in the system is found in “Putting It All Together” on page 22.

If you are using LU 6.2, many of the requirements for TPFAR are defined by the TPF Advanced Program-to-Program Communications (TPF/APPC) interface. For more information about TPF/APPC, see the *TPF ACF/SNA Data Communications Reference*.

## TPF Requirements

This section contains information about how to configure your system for the TPF Application Requester feature.

## Configuring TPFAR

To include TPFAR in your system, you must specify TPFAR=YES on the SIP CONFIG macro for TPF Application Requester support.

For additional information about the SIP CONFIG macro, see *TPF System Generation*.

## Defining the Applications for LU 6.2

For TPFAR to connect with the DB2 system by using LU 6.2, both the local TPF/APPC application in the TPF system and the remote DB2 system primary logical unit (PLU) must be defined to the communication network as TPF/APPC resources.

## Defining the Local TPF/APPC Applications for LU 6.2

To define the local TPF/APPC application to TPF, the SIP stage 1 deck must be updated with an entry for the TPF/APPC application. This is done with the MSGRTA macro. Use the MSGRTA APPL parameter to specify the local TPF/APPC application name. Use the name on the MSGRTA macro when you define the TPF application. The application is defined by one of two ways, depending on whether your system is loosely coupled. For example, we will define the TPF system application as TPAR, that is, APLIC=TPAR.

If your TPF system is loosely coupled, you need to define a separate local TPF/APPC application in TPF for each processor in the TPF system loosely coupled complex. These are known as service LUs. Note that the name of the service LU is SVC*x*, where *x* is the processor (CPU) ID.

Using the service LUs is the recommended way to define the local TPF/APPC application to the TPF system for maximum efficiency. Another method of defining the local TPF/APPC application is documented in *TPF ACF/SNA Data Communications Reference*.

## Defining the Remote DB2 PLU Application for LU 6.2

To define the DB2 system to the TPF system, the resource deck input to offline SNA table generation (OSTG) function can be updated to have the DB2 system defined as an LUTYPE=L6PLU resource, if dynamic LU is not used. For example: to have DB2 defined as an LUTYPE=L6PLU resource, for example:

```
TPFDB2T RSC LUTYPE=L6PLU
```

For more information about OSTG, see *TPF ACF/SNA Network Generation*. See the *TPF ACF/SNA Data Communications Reference* for more information about how to define SNA resources.

## Attaching to the LU 6.2 Communications Cloud

The TPF system attaches to the communications system, which attaches to the remote system running the DB2 system.

### Defining the Channel-Attached Link Station for MVS

The connection between the DB2 system and TPF Application Requester is either a 37x5 running a Network Control Program (NCP) defining the TPF system as a type PU 2.1 or type 5 node, or a 3088 Channel-to-Channel (CTC) connection defining the TPF system as a type PU 5 node. Both types of connections can have an entry in the adjacent link station (ALS) deck of the OSTG.

Each CTC link is uniquely defined to VTAM by its qualifier number. This number is needed when setting up the definitions for a CTC connection on VTAM and when defining the CDRSC statement on VTAM. See “Defining the TPF Application LUs to VTAM” on page 19 for more information about how this is needed for the VTAM definitions.

See the *TPF ACF/SNA Network Generation* for more information about coding CTC and ALS statements in an OSTG deck.

### Defining the Channel-Attached Link Station for the DB2/6000 System

TPF Application Requester and IBM DATABASE 2 AIX/6000 system (DB2/6000) connection uses an enterprise systems connection (ESCON) adapter (5765-603) or a block multiplexer channel adapter.

Activate a PU 2.1 link between the TPF system and the RISC System/6000 (RS/6000) in one of several ways:

- TPF NCP/CTC/ALS dynamic discovery processing that takes place during SNA restart
- The TPF system ZNETW ACT ID-ALSNODES command, which activates all PU 2.1 links
- Starting SNA support from the SNA Server/6000
- Starting a specific link station from the SNA Server/6000.

### Communications Requirements for the RS/6000 System

See *ESCON Users Guide and Service Information* and *SNA Channel Connectivity for AIX* for information about the required software.

### Communications Requirements for the PS/2

The TPF Application Requester communicates with the DB2 system on a Personal System/2 (PS/2) through an LU 6.2 connection.

## Defining the TPFAR Storage Areas

In the SNAKEY macro, the following additional definitions are required. All of these storage areas are carved out of storage above the 16-MB line.

- MAXHCT specifies the maximum number of hotcon table (HCT) entries between remote relational databases (RDBs) and TPFAR. This is the maximum number of hotcons for the entire processor, including all subsystems. The HCT contains TPFAR communication parameters that are saved as an entry in the HCT for later use by another transaction. You must use the ZSQLD command to assign the exact number of hotcons per database.

  Hotcons are described as follows depending on the communication protocol used:

  – A hotcon for LU 6.2 is a TPF/APPC conversation that remains allocated and active past the completion of the transaction; that is, active until the SNA session is deactivated or the conversation is deallocated by the remote transaction program. The TPF/APPC conversation parameters between TPFAR and the DB2 system are saved in an entry in the HCT. When another ECB requests a conversation with the same remote application server, TPFAR reuses the active conversation.

  – A hotcon for TCP/IP is a TCP/IP connection that remains active past the completion of the transaction. The socket descriptors are saved in an entry in the HCT. When another ECB requests a connection with the same remote application server, TPFAR reuses the active connection.
- MAXSDD is the maximum number of SQL database management systems (DBMSs) that communicate with each subsystem. The number indicated in MAXSDD is the number of entries reserved for each subsystem. This parameter is required to be nonzero in order for TPFAR to work.
- MAXSMTB is the number of 4K storage areas used for the SQL trace table. An SQL trace table is set up for each currently active I-stream.

**Note:** In a loosely coupled environment, each processor has its own copy of keypoint 2.

For detailed information about the SNAKEY macro, see the *TPF ACF/SNA Network Generation*.

## Other TPF System Storage Requirements

TPF Application Requester affects storage on the TPF base system. When an SQL call is issued, all processing for that ECB is suspended until control is returned.

**Note:** When developing an application, be careful of any locks, file records, or core blocks associated with an ECB over an SQL call. All processing for this ECB is suspended until a response is received from the remote AS. This can result in a TPF system resource problem.

For TPFAR, you need to assess the following TPF system resources to ensure that storage requirements are adequately met:

- **Heap Storage**: TPFAR uses the malloc function for all of its storage requirements. The exact amount of heap storage needed by TPFAR is dependent on the type of SQL requests made by the TPF system applications. For instance, an SQL INSERT command with many host variables defined as characters will take up more storage than an SQL INSERT command with two host variables defined as short integers.
- **#IBMMP4 Records**: Ten IBM miscellaneous processor unique records (#IBMMP4) are defined for the SQL database management system directory (SDD).
- **TPF/APPC Fastpath**: TPF/APPC uses OM (output message) records to queue messages received. In most TPF systems, most OM records are not VFA candidates. A separate record ID X'FF0F' is used to queue messages for conversations with TPFAR; the new record type is a VFA candidate. Copying data from VFA instead of doing I/Os improves performance.
- **Short Term Pools**: Short term pools are used to store the SQL transaction profile (STP) and associated blocks when the DBSAC and DBSDC macros/functions are used. The DBSDC macro/function detaches the associated blocks from the ECB and files them out in short term pools.

  **Attention:** Ensure that the short term pool cycle time is large enough for records to still be present when the DBSAC macro/function is used to reattach the STP and associated blocks.

  See “TPFAR Working Storage Blocks” on page 30 for more information about the DBSAC and DBSDC macros/functions. See also *TPF General Macros* and *TPF C/C++ Language Support User’s Guide* for additional information about the DBSAC and DBSDC macros/functions.

## LU 6.2 Requirements

TPFAR can use the TPF/APPC code for communication between the remote application server (AS) and TPF.

**Note:** Ensure that the following areas have values sufficient for TPFAR:

- MAXTPI in the SNAKEY macro may need to be updated to include a transaction program instance (TPI) identifier for each TPF/APPC session between the TPF system and the remote AS that will be in use at one time. See the *TPF ACF/SNA Network Generation* for more information about the MAXTPI parameter of the SNAKEY macro.
- The MAXSCB parameter of the SNAKEY macro may need to be updated to include additional session control blocks (SCB) for the TPF/APPC

sessions with the remote AS. One SCB is needed for every TPF/APPC session that is active. The maximum number of active sessions for TPFAR is specified on the ZNCNS command. See *TPF ACF/SNA Network Generation* for more information about the MAXSCB parameter of the SNAKEY macro. See *TPF Operations* for more information about the ZNCNS command.

- The MAXCCB parameter of the SNAKEY macro may need to be updated to include additional conversational control blocks (CCB) for the TPF/APPC conversations used by TPFAR. One CCB is needed for every TPF/APPC conversation that is active. This number should be at least as high and possibly higher than the number of SCBs defined. See *TPF ACF/SNA Network Generation* for more information about the MAXCCB parameter of the SNAKEY macro.

Additionally, in order to use TPFAR in a PU 5 environment, the SNAKEY parameter NETID must be coded with the network id of the TPF system.

See the *TPF ACF/SNA Data Communications Reference* and *TPF ACF/SNA Network Generation* for additional information about TPF/APPC requirements.

## TCP/IP Requirements

TPFAR can use the TCP/IP code for communication between the remote application server (AS) and the TPF system.

**Notes:**

1. Configure the remote AS to use the same port number that was specified with the ZSQLD command for TPF to connect to the server. See *TPF Operations* for more information about the ZSQLD command.
2. The TPFAR feature requires that the server accept a security protocol of USERID only. The userid that TPF provides is the complex name that is contained in keypoint I.
3. The TPFAR feature uses one socket for each connection, so the number of available sockets must at least equal the number of connections. Use the MAXSOCK parameter of the SNAKEY macro to set the maximum number of sockets, including any additional sockets needed for the remote AS. If hotcons are used, sockets are then saved in the hotcon table (HCT). See *TPF ACF/SNA Network Generation* for more information about the SNAKEY macro and the hotcon table.

See *TPF Transmission Control Protocol/Internet Protocol* for additional information about TCP/IP requirements.

## Commands

The following commands are particularly applicable when using TPFAR.

- **ZSQLD**: The ZSQLD command displays and maintains the SQL database management systems directory (SDD). This command must be used to set up the directory with the necessary database information before any TPFAR application programs are started. The ZSQLD command also controls SQL entry tracing. Tracing can be global or selective.

  The following parameters are important when using the ZSQLD command to add an entry to the SDD.

  – Rdb must be the same as the location name specified in the DB2 system bootstrap data set (BSDS). See “DB2 Requirements” on page 20 for more information about the location name.

- If you are using LU 6.2, the LU must be the same as the LU name defined in the VTAM APPL statement for the DB2 system as well as in the DB2 bootstrap data set. See “DB2 Requirements” on page 20 and “VTAM Requirements for LU 6.2” on page 18 for more information about the LU name.
- If you are using LU 6.2, the mode must be the APPC mode name used with this RDB when an APPC allocate verb is issued. The default value is TPFDB2MO. This value must match the mode name defined to LU 6.2, which is the mode name used in the ZNCNS command. In MVS, the mode name must match what is used in the SYSIBM.SYSLUNAMES table. For OS/2 and AIX, the mode name matches the LU 6.2 connection.
- If you are using TCP/IP, the host name is the 1- to 128-character alphanumeric host name of the server or a dotted decimal notation of the Internet Protocol (IP) address of the server.
- Ccsid must be the same as the coded character set identifiers (CCSIDs) defined in the server (in MVS, the DSNTIPF installation panel; in OS/2, the code page specified in the CONFIG.SYS file; in AIX, the CCSIDs specified in SMIT). The CCSID specifies what coded character sets are in use on machines that the TPF system communicates with. The default is 500, which is the single-byte default used by the DB2 system for installation on MVS. The TPF system supports single-, double- and mixed-byte character sets. When you specify CCSIDs using ZSQLD, you can define either a single-byte or a mixed-byte CCSID alone or you can define a triplet made up of a single-byte·double-byte·mixed-byte combination. These combinations must be consistent with each other. The CCSIDs that are available are specified in a table loaded to the TPF system and translations involving them are defined in individual DLMs (CPG*x*).
- TPFCCSID specifies the coded character sets used by the TPF system. The same considerations described for CCSIDCcsid are true for TPFCCSID.
- MAXHCT specifies the number of hotcons to use for this RDB. A value of zero means that no hotcons will be kept active for this RDB.

The following is an example of a ZSQLD command to add an SDD entry:

```
ZSQLD ADD RDB-DB23TST,LU-TPFDB2T,MAXHCT-5,MODE-TPFMOD1
ZSQLD MOD RDB-DB23TST,LU-TPFDB2T,CCSID-897.301.932,TPFCCSID-282.300.930
```

- **ZNKEY**: You can use the ZNKEY command to display and alter the number of hotcons (MAXHCT), the number of the SQL database management systems defined to TPF system (MAXSDD), and the number of 4K storage areas used for the SQL trace table (MAXSMTB). See “Defining the TPFAR Storage Areas” on page 13 for more information about these storage areas.
- **ZSTTD**: The ZSTTD command formats and displays the SQL trace table. This can be helpful in both online debugging and monitoring TPFAR performance.

  You can use the ZSTTD command to display trace entries either in summary or in detail. The detailed form includes exception fields from the SQLCA.

For detailed information about these commands, see *TPF Operations*, *Messages (System Error and Offline)*, and *Messages (Online)*.

## Coding the SQL Trace Table User Exit

Each SQL command that is issued by the application is logged in the SQL trace table (if it is defined via the MAXSMTB parameter in SNAKEY). When the table is full, it wraps and subsequent SQL commands overwrite the existing entries in the table.

A user exit (segment UAR1) can be utilized to process the information in the SQL trace table before it is overwritten.

## Character Sets

A number of offline tasks must be completed before you can use a new character set on the TPF system or connect to a remote server that uses a character set that does not have a translate table loaded on the TPF system. See *TPF Application Programming* for more detailed information about character sets.

### Choosing a New Character Set

Character sets are chosen on the basis of the kinds of letters and symbols required. Once these requirements are understood, you can choose the appropriate character sets. To learn more about character sets, see *Character Data Representation Architecture Reference and Registry*.

### Translating Character Sets

A character set is referred to by a number called a *coded character set identifier* (CCSID). Each DB2 system contains characters in one or more CCSIDs. The TPF system also contains characters in one or more CCSIDs (or TPFCCSIDs).

Characters residing in the DB2 system must have a corresponding form in the TPF system to be used in the TPF system database. When the CCSIDs match, no translation is necessary. If the character sets are different, a translation mechanism must exist to transform each character from the remote CCSID to a corresponding character of the TPF system CCSID.

### Seeing the Overall Character Flow

The ZSQLD command defines the TPF system CCSIDs and verifies the remote CCSIDs. Consider, for example, the CCSID of a remote DB2 system CCSID (*xx*) and the TPF system CCSID (*yy*). The *xxyy* table name must appear in the CPGS table to identify the name of the DLM to be used for the translation from the remote database to the TPF system. The DLM name found in CPGS is used to make the conversions from one character set to the other. The ZSQLD command verifies that the CCSIDs specified have the correct STYLE. It does not verify the compatibility of the CCSIDs.

### Converting Numbers

Processors differ in how they represent numbers. These differences are automatically detected and accounted for by the TPF system.

## Loosely Coupled Requirements

When using TPFAR in a loosely coupled environment, be aware of the following conditions:

1. Each SDD is processor unique. When setting up the relational database (RDB) information, you must enter the ZSQLD command on each processor that handles TPFAR applications.
2. Because each processor has its own copy of keypoint record 2 (CTK2), you have to setup the CTK2 (using SNAKEY) parameters need to be set up for each processor.
3. For LU 6.2, separate service LUs must be defined for each processor in the complex. For more information about service LUs, see “Defining the Local TPF/APPC Applications for LU 6.2” on page 12.

## Subsystem Requirements

The TPFAR applications must be defined in each of the subsystems that accesses TPFAR. See *TPF System Generation* for more information about defining applications for subsystems. In addition, each subsystem has its own unique SDD record; for each subsystem communicating with the DB2 system, the RDB information must be added to the SDD by using the ZSQLD command.

Additionally, the originating subsystem where an ECB issues its first SQL command must be the subsystem where all subsequent processing must occur. This ECB cannot change subsystems after an SQL command has been issued. This requirement also applies when using the DBSAC and DBSDC macros/functions.

## VTAM Requirements for LU 6.2

To connect to the VTAM network, each relational database (RDB) must have an LU name defined. To define the RDB to the network, a VTAM APPL definition statement must be coded with the specified LU name to register the RDB name to VTAM. The LU name must match the LU name defined in the DB2 system bootstrap data set (BSDS). See “DB2 Requirements” on page 20 for additional information about BSDS. If defined in the OSTG resource deck, this LU name must match the LU name used to define the DB2 system PLU to the TPF system. The name can also be defined dynamically. See “Defining the Remote DB2 PLU Application for LU 6.2” on page 12 for more information. Here is an example of a VTAM APPL definition statement, where TPFDB2T is the LU name:

```
TPFDB2T APPL APPC=YES,
             AUTH=(ACQ),
             AUTOSES=1,
             DMINWNL=10,
             DMINWNR=10,
             DSESLIM=20,
             EAS=9999,
             MODETAB=RDBMODES,
             PRTCT=PSWDBD1,
             SECACPT=ALREADYV,
             SRBEXIT=YES,
             VERIFY=NONE,
             VPACING=10
```

Some of the keywords in the VTAM APPL definition statement are described briefly here; for detailed information about these and other keywords for the VTAM APPL definition statement, see the *IBM Database 2 Version 2 Administration Guide, Volume 1* and the *VTAM Resource Definition Reference Manual*.

**Note:** Many of the keywords on the VTAM APPL statement can be overridden by defining entries in the DB2 system communications database (CDB). See the *DB2 Administration Guide Volume I* for more on this relationship.

- The EAS keyword specifies the total number of sessions for all partners.
- The PRTCT keyword identifies the VTAM password to use when the DB2 system attempts to connect to VTAM. If the PRTCT keyword is not specified on the definition statement, the password is not required, and the bootstrap data set must have not been coded with a password on the DB2 Distributed Data Facility (DDF) panel DSNTIPR, or the NOPASSWD option of the DDF statement of the DB2 system change log inventory utility must have been specified. If the PRTCT keyword is specified, it must match the password defined in the BSDS. For more information, see the *DB2 Administration Guide*.

- The SECACPT keyword denotes the highest SNA conversation-level security value accepted in the DB2 system subsystem when the system receives a distributed database request from a remote system.

  **Note:** You must specify ALREADYV for this keyword because conversational security is not supported in TPF/APPC.
- The VERIFY keyword identifies the level of SNA session security (partner LU verification) required by the DB2 system subsystem.

  **Note:** You must specify NONE for this keyword because session level security is not supported in TPF/APPC.

## Mode Names

TPFAR requires a mode definition in the VTAM Logon Mode Table. This mode name is needed for establishing the LU 6.2 sessions between TPFAR and the DB2 system. An example of mode definition is:

```
TPFMOD1  MODEENT LOGMODE=TPFMOD1,SSNDPAC=X'02',RUSIZES=X'F8F8'
```

See the *DB2 Administration Guide Volume 1*, and the *VTAM Resource Definition Reference* for more information about mode definitions.

## RU Sizes and Pacing Considerations

In order for the VTAM network to run efficiently in the distributed access environment, it is important that you carefully specify RU sizes and pacing. Suggested guidelines can be found in the *Distributed Relational Database Connectivity Guide*.

# Connecting a TPF System and a DB2 System Using LU 6.2

The routing between TPFAR and the DB2 system is through a channel-attached link station. This link station can be (1) a 37x5 with the Network Control Program (NCP) defining the TPF system as a type PU 2.1 or type 5 node or (2) a 3088 Channel-to-Channel (CTC) connection defining the TPF system as a type PU 5. At least one of these two types of channel link stations is required for communication between the TPF system and the DB2 system.

## Defining the TPF Application LUs to VTAM

The LU 6.2 applications are defined on the TPF system with the MSGRTA macro. See “Defining the Local TPF/APPC Applications for LU 6.2” on page 12. How the TPF LU 6.2 applications are defined to VTAM depends on whether the TPF system is defined as a type PU 5 node or type PU 2.1 node. You need to code a CDRSC statement on VTAM when the TPF system is a type PU 5 node. You need to code an APPL statement on VTAM and an LU statement in the NCP generation when the TPF system is a type PU 2.1 node.

The name you use on the APPC statement has to match the name you used on MSGRTA to define the local TPF/APPC application. The local LU you described in a previous example was defined as TPAR or SVC*x*. The names on the LU statement in the NCP gen are TPARA30 and TPARA34. For further information about the VTAM APPL statement, the VTAM CDRSC statement, and the NCP LU statement, see the *VTAM Resource Definition Reference Manual*. For examples showing how to define LUs, see the *TPF ACF/SNA Data Communications Reference*.

## Configuring the TPF System and a DB2/6000 System

To connect the TPF system to DB2 on RS/6000 systems there are changes required in a TPF system SIP macro and in the SNA Server/6000 settings.

SIP macro MSGRTA defines LU 6.2 applications running on the TPF system. When defining the TPF system applications for SNA Server/6000, the TPF system applications are defined as LU 6.2 partner LUs and the names used are the same as those used in the MSGRTA macro.

TPF system applications that are defined on the SNA Server/6000 have the following settings:

- A fully qualified name: The network identifier must be the same network identifier used by the TPF system for activation (in SNAKEY macro parameter LENNETID). The ZNKEY command displays this value.
- Parallel session support: Parallel sessions with the DB2 system are recommended (YES). Setting the parameter to NO means that only single sessions are available.
- Session security support: This support is set to NO because TPF/APPC does not support session-level security.
- Conversation-level security: This support is set to ALREADY_VERIFIED because TPF/APPC does not support conversation-level security.

You must define the DB2/6000 system to SNA Server/6000 support as a local LU 6.2 logical unit.

**Note:** You do not have to add the DB2/6000 LU name to the TPF system OSTG deck if dynamic LU definition support is active. If this support is not active, you must include the LU name in the OSTG deck as LUTYPE=L6PLU.

You must define all LU 6.2 mode names used for sessions between the TPF system and the DB2/6000 system to the SNA Server/6000. The important fields used as a part of mode definition are:

- Mode name.
- Total session limit for an (LU,mode) pair.
- Minimum number of contention winner sessions.
- Minimum number of loser sessions.
- Number of sessions automatically activated. This is the number of sessions driven by the DB2/6000 system when initialization of the number of sessions (CNOS) has been performed for the mode.
- Receive pacing window size (for fixed pacing).
- Maximum adaptive pacing receive window size.
- Maximum and minimum SNA response units (RUs) sizes.
- Class of service (COS) name.

See the *ESCON Users Guide and Service Information* and *SNA Channel Connectivity for AIX* for information about these settings.

## DB2 Requirements

To connect the TPF system with a DB2 system:

- For MVS systems, the DB2 system must be at Version 2 Release 3 to Version 6.

- For AIX or OS/2 systems, the DB2 system must be at Version 2 Release 1 to Version 6.

The following considerations pertain solely to the DB2 system running on MVS.

The DB2 system Installation Panel DSNTIPF allows you to set the coded character set identifier (CCSID) for the DB2 subsystem. This must match the CCSID in the TPF system SDD entry for this database. For detailed information about the DB2 Installation Panels DSNTIPR, DSNTIPE, and DSNTIPF, see the *DB2 Administration Guide*. The location name (RDB name), the LU name, and the CCSID specified on these panels are the same values that must be used when issuing the ZSQLD command to define this database to TPF. See the *TPF Operations* and "Commands" on page 15 for more information about ZSQLD command.

Under the DB2 system, the DB2 Distributed Data Facility (DDF) uses the system installation parameters to connect DB2 to VTAM. The DB2 Installation Panel DSNTIPR allows you to enter the required DDF BSDS information that describes the connection parameters between the DB2 system and VTAM, such as the location name (RDB name), LU name, and password. The DB2 Installation Panel DSNTIPE allows you to set the number of maximum number of remote database access threads that can be allocated concurrently. There must be one thread available for each LU 6.2 session from the TPF system you plan on using.

DDF connects to VTAM and passes the LU name and optional password to VTAM. VTAM verifies the LU name and password, if specified, with what was coded on the VTAM APPL statement. See "VTAM Requirements for LU 6.2" on page 18 for more information about the VTAM APPL definition.

The communications database is made up of five interrelated tables and is an important source of information about the system. To run TPFAR with a DB2 system, minimum changes are required in two tables, the SYSIBM.SYSLUNAMES (LU 6.2 only) and the SYSIBM.SYSUSERNAMES tables. These changes cause DB2 to bypass the verifying of the ID sent on the LU 6.2 ALLOCATE verb with RACF. For TCP/IP, USER.ID security will be used to verify the TPFAR client.See the *DB2 Administration Guide* for more information about these tables.

- For LU 6.2 only, the SYSIBM.SYSLUNAMES table defines the LU names that can connect to the DB2 system. This table initially contains one entry that is blank. You can delete the blank entry and specify the LU names that are allowed to connect to the DB2 system.

  To allow TPFAR to connect to the DB2 system a new entry must be added to this table for each local TPF/APPC application defined in TPF. Each entry must have:

  – The USERNAMES column of the entry set to I, meaning that inbound ID translation and "come from" checking is to occur. ("Come from" checking is an LU 6.2 security option, which defines a list of authorization IDs that are allowed to connect to the DB2 system from a partner LU).
  – The LUNAME column of the entry set to the LU name defined for the local TPF/APPC application in TPF. In our examples, it has been TPAR or SVC*x* in a loosely coupled environment.
  – The SYSMODENAME column of the entry set to the name of the LU 6.2 mode defined to VTAM in the mode name table and used on the ZNCNS command. In the following example, this is TPFMOD1.

  The following SQL command to accomplish this change can be issued through the SQL Processor Using File Input (SPUFI). SPUFI is a way to execute SQL commands from a TSO terminal.

```
INSERT INTO SYSIBM.SYSLUNAMES (USERNAMES, LUNAME, SYSMODENAME)
    VALUES ('I', 'TPAR', 'TPFMOD1');
```

For a loosely coupled environment using the service LUs, one entry is needed in this table for each SVCx defined in the complex.

See the *DB2 Application Programming and SQL Guide* for more information about SPUFI.

- The SYSIBM.SYSUSERNAMES table needs a new row to describe the translation that is needed for the inbound translation. The translation is necessary to bypass the step in which the DB2 system uses RACF to check the validity of the name. Because this only involves inbound translation, the type specified must be I, and the AUTHID must be the same as the TPF system complex name specified in keypoint I (CTKI). The LUNAME and NEWAUTHID fields are blank because no real translation is needed; all that is desired is to bypass the RACF check. For TCP/IP, USER.ID security will be used to verify the TPFAR client.

  The following SQL command to accomplish this change can be issued through SPUFI:

  ```
  INSERT INTO SYSIBM.SYSUSERNAMES (TYPE, AUTHID, LUNAME, NEWAUTHID )
         VALUES ('I', 'TPFNET', ' ', ' ');
  ```

Because the communications database consists of five interrelated tables, the information that you specify in one table might have to be specified in the other tables. See the *Distributed Relational Database Connectivity Guide* for detailed information about the communications database.

The system administrator also needs to grant access for the TPFAR application's AUTHID. TPFAR uses the TPF system complex name found in CTKI as its application AUTHID. Depending upon the SQL commands that are issued from the TPF system, the AUTHID needs sufficient privileges to perform the commands. See the *DB2 SQL Reference* guide for more information about the SQL GRANT command necessary to set up the access.

## Putting It All Together

Figure 9 shows the relationships between the various parameters that you have to specify for the TPF system, a DB2 system, and VTAM on MVS for LU 6.2.

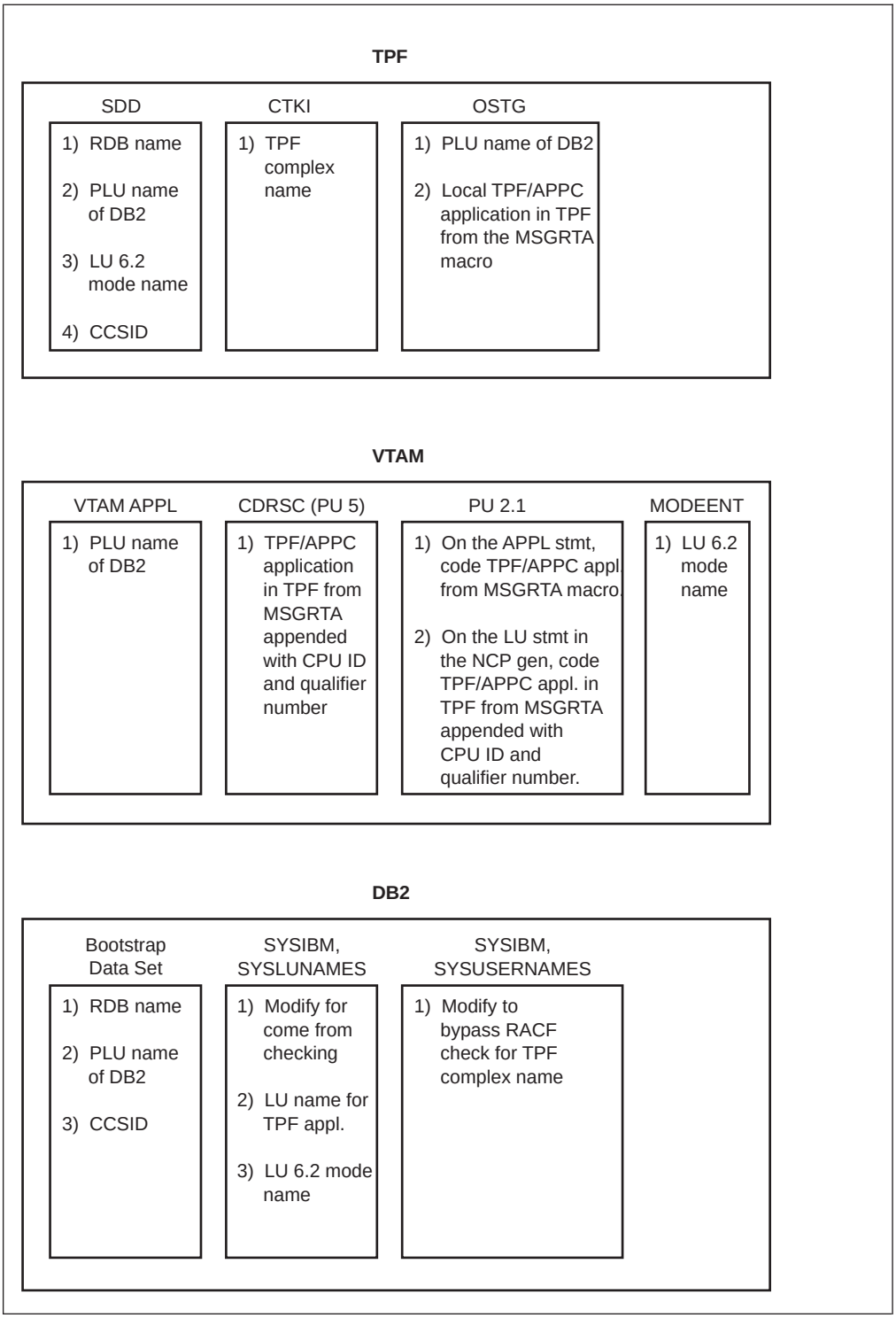


*Figure 9. Parameter Relationships on MVS for LU 6.2*

Figure 10 shows the relationships between the various parameters that you have to specify for the TPF system, a DB2 system, and VTAM on MVS for TCP/IP.

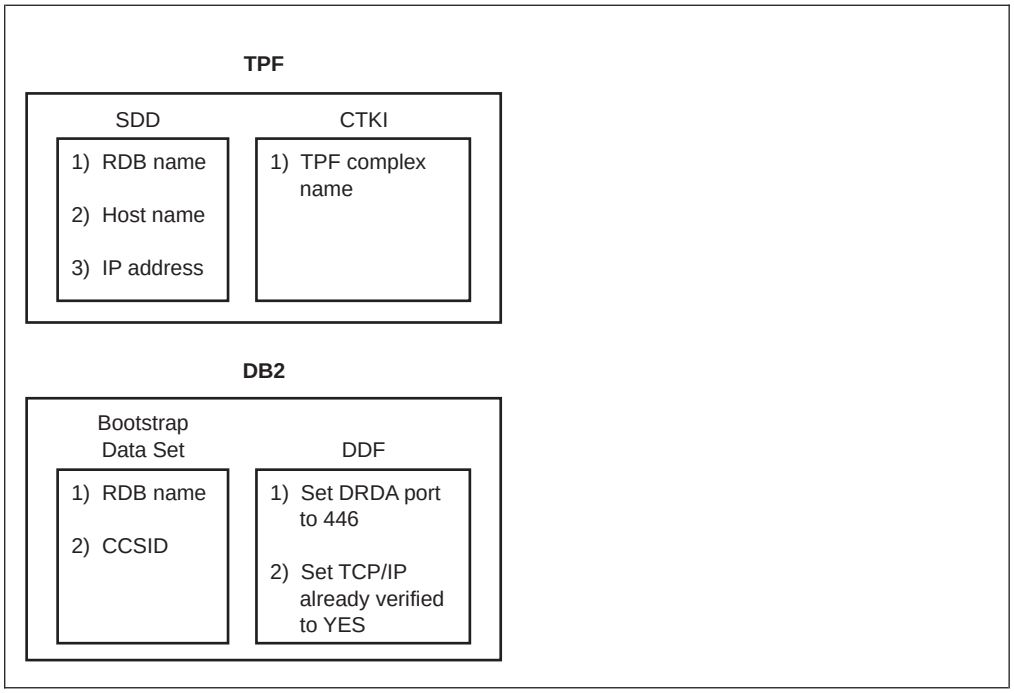


*Figure 10. Parameter Relationships on MVS for TCP/IP*

Figure 11 on page 25 shows the relationships of the TPF, DB2, and VTAM parameters on MVS with the specific values used in previous examples in this section.

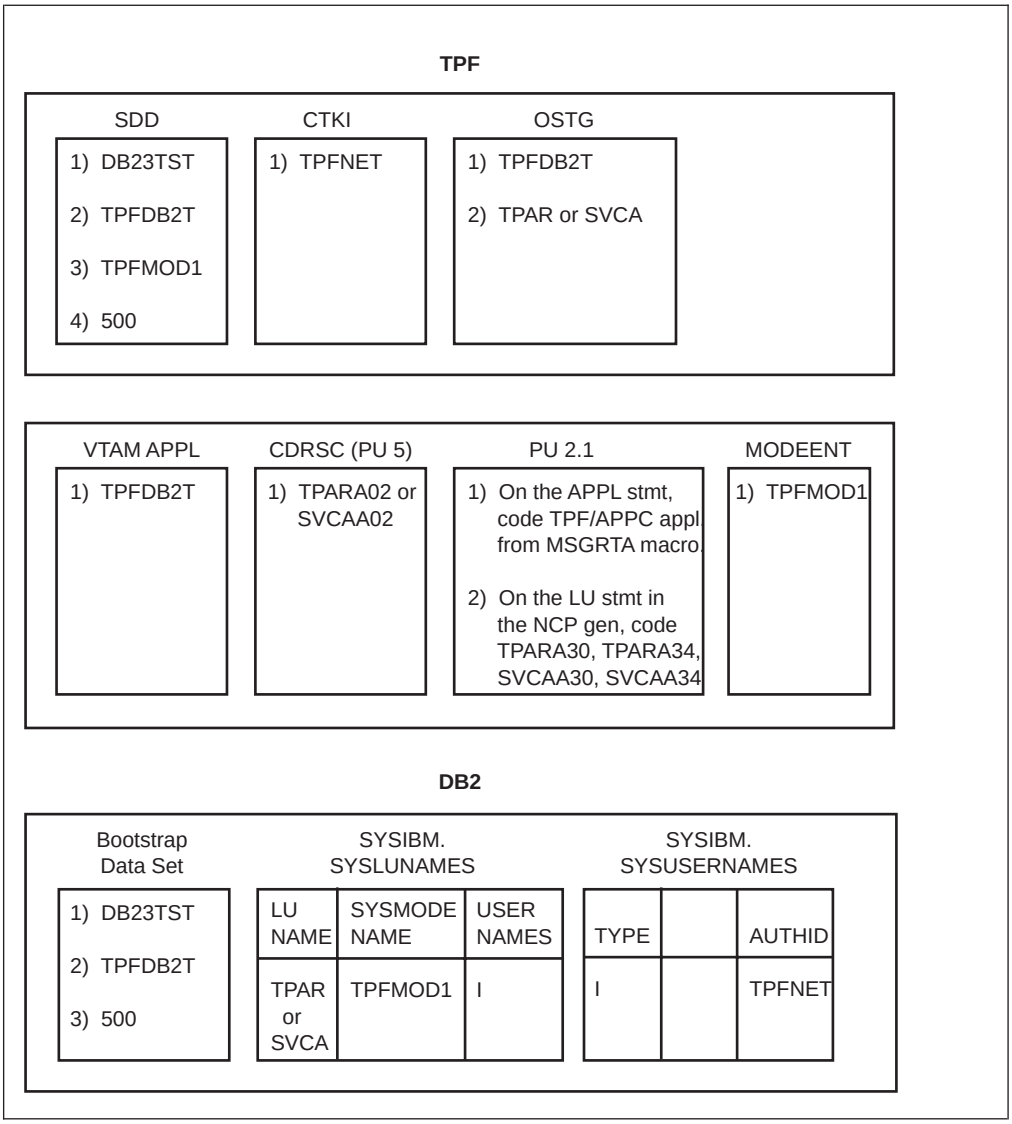


*Figure 11. Parameter Relationships with Specific Examples on MVS*

For the RS/6000 system the TPF system parameters and the communications parameters for LU 6.2 are substantially the same. The RS/6000 system, however, does add its own parameter requirements as shown in Figure 12.

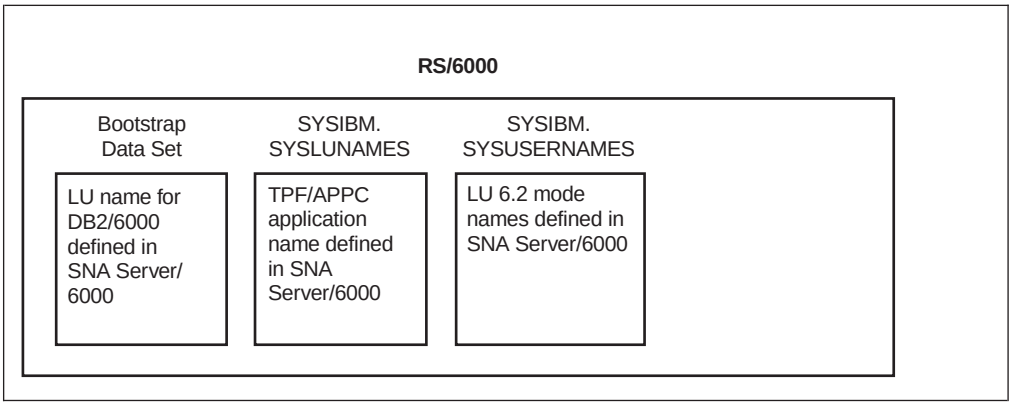


*Figure 12. Parameter Relationships on RS/6000*

## Starting the TPF/APPC Application for Use with TPFAR

After all of the definitions have been set up, the last item is to start the TPF/APPC application for use by TPFAR.

1. Start the TPF/APPC application on the TPF system with the ZROUT command. For example:

   ```
   ZROUT START TPAR

        or

   ZROUT START SVCx
   ```

2. Activate the TPF/APPC application on the TPF system with the ZNETW command. For example:

   ```
   ZNETW ACT ID-APPC

   or

   ZNETW ACT ID-SVCx
   ```

3. Start the logon manager session with the local CLU. This is done with the ZNETW command. For more information about how to activate CLU-CLU sessions, see *TPF ACF/SNA Data Communications Reference*.
4. Initialize the number of allowed sessions with the DB2 LU. Note that the LU name is the same as the PLU name of the DB2 system and mode name is the mode as defined on VTAM on the mode table. (See “Defining the Remote DB2 PLU Application for LU 6.2” on page 12.)

   ```
   ZNCNS I LU-TPFDB2T,MODE-TPFMOD1,LIMIT-10,CONW-10,LOCAL-TPAR

                    or

   ZNCNS I LU-TPFDB2T,MODE-TPFMOD1,LIMIT-10,CONW-10,LOCAL-SVCx
   ```

   **Note:** Because TPFAR initiates all conversations, LIMIT and CONW should be equal.

See *TPF Operations* for more information about these commands.

# Setting the Stage with a Telephone Directory Application

This section uses a telephone directory application as an example to illustrate the role TPFAR and relational databases play in developing an application. The programs shown in the following examples have been run in a TPF system using LU 6.2.

**Note:** The sample code presented in the following examples illustrates the different methods of working with SQL, TPF, and DB2; the examples are not intended to reflect the best method for implementing the code.

## A Few Words about Relational Databases

Relational databases consist of a set of related tables. The tables are made up of rows and columns. In a table, a column is a vertical arrangement of information, and a row is a horizontal arrangement of information. Table 1 contains an example of a corporate telephone directory. Each row contains phone information about an employee. This information is arranged by columns. Each column is a different part of the employee information. In our example, the employee record is broken up into last name, first name, middle initial, country code, city code, phone number, employee number, and an internal time stamp value. In our example, the time stamp indicates the last time the record was updated.

*Table 1. Example of a Corporate Telephone Directory*

| Last Name | First Name | Middle Initial | Country Code | City Code | Phone | Employee Number | Time Stamp |
|---|---|---|---|---|---|---|---|
| Villard | Jean | E | 33 | 6 | 555-2342 | 4 | 1991-05-04-10.30.30.123423 |
| Martinez | Juan | C | 52 | 748 | 555-1923 | 2 | 1991-05-04-12.30.30.345452 |
| Durr | Robert | | 1 | 914 | 555-1272 | 1 | 1991-05-04-16.00.12.324567 |
| Chiba | Takao | | 81 | 3 | 555-1625 | 6 | 1991-02-01-19.32.13.432342 |
| Johnson | Hans | | 49 | 89 | 555-1625 | 5 | 1990-11-21-22.45.32.875643 |
| Stewart | Mary | L | 44 | 1 | 555-2323 | 3 | 1991-08-08-14.14.08.847372 |
| Stewart | James | J | 44 | 1 | 555-2324 | 7 | 1991-03-23-15.39.59.948373 |

## Creating the Telephone Directory

You can create the telephone directory by using an SQL CREATE TABLE command. You can do this in DB2 using interactive SQL. On MVS, you access this through the SQL Processor Using File Input (SPUFI) application or from a TPF application. SPUFI is a way to execute SQL commands from a TSO terminal. See the *IBM DATABASE 2 Application Programming and SQL Guide* for more information about SPUFI. On AIX and OS/2 systems, you access interactive SQL by typing **db2** on the command line. You can also prepare data definition language (DDL) files containing SQL statements.

Use the SQL CREATE TABLE command through the DB2 system because:

- The SQL CREATE TABLE command is usually issued only once. Issuing this command through SPUFI is much easier and quicker than from a TPF application.
- When issuing an SQL CREATE TABLE command, all of the DB2 internal system tables are locked so that no other application can access table information. If the

SQL CREATE TABLE command is issued from TPF, the added communications overhead can cause problems on the DB2 side if any other application is accessing DB2 at that time.

Figure 13 shows the SQL CREATE TABLE command that creates the PHONE_DIRECTORY table.

```
CREATE TABLE TPFNET.PHONE_DIRECTORY
   (LAST_NAME        CHAR(17)  NOT NULL,
    FIRST_NAME       CHAR(8)   NOT NULL,
    MIDDLE_INITIAL   CHAR(1),
    COUNTRY_CODE     CHAR(4),
    CITY_CODE        CHAR(5),
    PHONE_NUMBER     CHAR(12),
    EMPLOYEE_NUMBER  SMALLINT  NOT NULL,
    TIME_STAMP       TIMESTAMP NOT NULL);
```

*Figure 13. SQL CREATE TABLE Command to Create the PHONE_DIRECTORY Table*

## SQL Considerations

When writing TPFAR application programs in C or assembler, you should keep the following SQL considerations in mind.

**Note:** You should have a basic knowledge of the SQL programming language.

## Length of Time Field

TPFAR requires that you use at least 8 numeric characters when specifying a time field. If a value of 5 to 7 characters is specified, truncation occurs, and SQLWARN is set in the structured query language communications area (SQLCA). The SQLCA contains information about the execution of SQL commands. If a value of 0 to 4 characters is specified, an SQLCODE and SQLSTATE is set indicating an error. SQLSTATE is a system-independent SQL return code field for the outcome of the last executed SQL command. SQLCODE is a system-dependent SQL return code.

## Request Unit Size Considerations

There are different LU 6.2 request unit (RU) (also known as datastream structure within DRDA) size requirements depending on the flow of data. For the outbound flow (information passed to DB2), TPF/APPC has a MAXRU of 3840 bytes.

When calculating the size of an RU, the following overhead needs to be taken into account. For each command, a 10-byte base overhead is needed, as well as a 3-byte per column overhead, and a 1-byte per null field overhead for the indicator variable. To better understand this, look at Figure 13 and Table 1 on page 27. To calculate the size of the outbound data, add up the different column sizes: 17 + 8 + 1 + 4 + 5 + 12 + 2 + 26= 75. Added to this is a 10-byte base overhead, plus 3 bytes for each column (3 × 8 = 24), plus 1 byte for each null column (4). This means that 113 total bytes are needed. Figure 14 on page 29 illustrates this calculation.

```
                LAST_NAME                    17
                FIRST_NAME                    8
                MIDDLE_INITIAL                1
                COUNTRY_CODE                  4
                CITY_CODE                     5
                PHONE_NUMBER                 12
                EMPLOYEE_NUMBER               2
                TIME_STAMP              +    26
                                           ------
Total number of bytes needed for the data:   75

10 bytes base overhead                       10

8 columns times 3 bytes per column:          24

4 nullable columns                      +     4
                                           ------
    Total bytes needed for this RU:         113
```

*Figure 14. Example of Calculating RU Size*

When receiving data from DB2, the same size calculations described above can be used to figure out the size of the data returned. The one case that deviates from this is when a cursor is used.

If you opened a cursor and issued an SQL FETCH command, as long as block fetch is being used, the application server (AS) sends back the data blocked into the query block size (QRYBLKSZ) that is used by TPFAR.[5] The QRYBLKSZ value used by TPFAR is 3800 bytes. Therefore, the first PIU returned from the AS returns 33 rows, if there were 33 rows in the answer set (3800 ÷ 113 = 33). If the answer set is less than 33 rows, the entire answer set is returned in the first PIU.

To force a block fetch to be used by DB2, you can open a cursor with the FOR FETCH ONLY option. No subsequent SQL commands can issue an UPDATE or DELETE WHERE CURRENT OF for this cursor. In addition, DB2 may decide to use block fetch even if the FOR FETCH ONLY option is not specified, depending upon the other SQL commands that are working on this same cursor. See the *IBM DATABASE 2 Application Programming and SQL Guide* for more detailed information on block fetch.

TPFAR keeps track of all the rows in the query block and returns only the first row to the application. Subsequent fetches for rows made by the application can then be accomplished with no further calls to the AS; instead, only local processing is performed by TPFAR. This greatly cuts down on the response time. In addition, while TPFAR and the application are working with this first set of data, the AS can be working on the next set to return; this too reduces response time.

See the *Distributed Data Management Architecture Reference* for more information about how to calculate how much storage TPFAR uses.

## Number of Cursors

The maximum number of cursors that an application program can have open at one time is 10. Over the lifetime of the application there can be as many opened cursors as desired, but at any one time, there can only be 10 open cursors.

5. The SQL FETCH command retrieves the next requested row in an answer set table. See the *IBM DATABASE 2 SQL Reference* for additional information on this command.

## Protect Key

When an SQL call returns to the application, its protect key is reset to application protect key 1. Therefore, any required protect keys have to be reestablished on return from the SQL call.

## Addressing Mode

When writing a TPFAR application in assembler, the TPFAR system code always returns to the application in the mode that the application call was made, either 24-bit or 31-bit mode.

## Registers

When writing a TPFAR application in assembler, no application registers are saved over the SQL call. Therefore, the application must save all the application registers needed by the TPFAR assembler application after the SQL call.

## Dynamic SQL

Dynamic SQL is supported by the TPFAR feature with the PREPARE, EXECUTE, EXECUTE IMMEDIATE, and DESCRIBE verbs. See “SQL Commands Supported by TPFAR” on page 101 for a list of the SQL commands that are supported and not supported by the TPF system.

## Collection Identifiers

A collection identifier specifies a group of packages. It is used in SQL applications to provide additional detail for the package identifier. See “Preparing an Application” on page 32 for more information about packages.

## TPFAR Working Storage Blocks

Between application SQL calls, TPFAR holds on to a number of storage areas for its own use. During this time, these areas may need to be freed up. For instance, if the application is returning the data to a terminal, quite a bit of time could pass before the terminal operator asks for more data. The DBSDC and DBSAC macros or functions can be used to file the storage areas out to DASD using TPF 4K short term pool. When the terminal asks for more data, the file records containing the TPFAR storage areas can be retrieved from DASD and reattached to the user’s ECB in order for more SQL commands to be issued. For more information about the holding and release of malloc blocks, see the information on heap storage in the *TPF Main Supervisor Reference*.

You need to be aware of the effect these macros have on the TPF and DB2 systems. Use attach (DBSAC) relatively quickly after the detach (DBSDC) because:

- The file types used are short term pools. Therefore, take care when using the DBSAC macro to reattach these records before the short term pools are recycled. Otherwise, all information pertaining to this SQL query is lost.
- All DB2 locks that are being held are still in effect. For example, if you have a write lock on a page on DB2, no one else can read or write to that page. If other transactions are running and require this page, they will eventually time out and get SQL error return codes. Issuing a query can cause problems because you are holding a read lock on the page.
- The LU 6.2 conversation that the current unit of work is using is not available for any other application’s use. This can lead to conversation resource problems on TPF if they are not returned for other applications to use.

- In short, this service should only be used after a thorough analysis of the impact on TPF and DB2 system performance.
- Because the storage areas acquired by malloc are not released until the ECB is exited, this service should only be used when exiting an ECB.

See the DBSAC and DBSDC macros in *TPF General Macros* for more information about the use of these macros.

**Note:** There are also C versions of the DBSAC and DBSDC macros. See *TPF C/C++ Language Support User's Guide* for information about these functions.

## Synchronizing Updates

Synchronizing the data is very important. The application program is responsible for synchronizing the updates on TPF with those on the remote RDB. The synchronizing is not done using LU 6.2 Sync_Level = CONFIRM or SYNC_PT.

An application program must consider the following when attempting to synchronize data on DB2 and TPF:

- The data on DB2 can only be guaranteed when a zero SQLSTATE is returned to the application from an SQL COMMIT command. Therefore, just because an SQL INSERT or an SQL UPDATE command returns a zero SQLSTATE does not mean that the record is out on DB2. The application must wait for the SQL COMMIT command with an SQLSTATE of zero to return before concluding any work, for example, when releasing pool files on TPF for the information altered, inserted, or deleted.

  If a zero SQLSTATE is not returned on the SQL COMMIT command, the changes made have not been reflected in the DB2 database, and the database looks the same as it did before the changes, even though the change itself may have had a zero SQLSTATE returned. For example, an SQL INSERT command may complete with a zero SQLSTATE, but if the SQL COMMIT command completes with a nonzero SQLSTATE, the change will not be reflected in the database.
- When the TPF or DB2 subsystem goes down, application restart logic is very important to guarantee data integrity. For example, if you issue an SQL COMMIT command and DB2 goes down before the SQL COMMIT command is received, an implied ROLLBACK is performed and all the work is backed out when DB2 comes back.

  Another example would be the case where the SQL COMMIT command makes it through DB2, but on the way back through VTAM, the connection between TPF and VTAM goes down and the response is lost. The data is out on DB2, but the TPF application does not receive the confirmation. One method of addressing this situation is to create another table to hold log records. Before the application program issues an SQL COMMIT command, an SQL INSERT command is issued to the table to log the progress of the application's processing. The same scenario occurs in assembler language; see "Using Assembler Language with TPFAR" on page 79 for the assembler example.

  If the SQL COMMIT command fails, and the connection can be reestablished, the log table can be interrogated to determine the last committed change by DB2. The application can then start from this point.

## C Language Header Files

When writing a TPFAR program in C, the header file `tpfarapi.h` must be included in every C program with SQL instructions. This header file contains the linkage for the SQL calls. For additional information on `tpfarapi.h`, see the *TPF C/C++ Language Support User's Guide*.

## Error Handling

Error handling for TPFAR follows four basic guidelines:

- All errors appear as relational database errors reported to the application through the SQLSTATE and SQLCODE fields of the SQL communications area (SQLCA). The SQLCA is defined in the *IBM SQL Reference Version 1*. SQLSTATE codes are consistent across all SAA platforms and provide a consistently portable interface.

  **Note:** The SQLCODE field is also reported; however, the application should not use this field to check for errors because it contains error codes specific to a particular relational database implementation. The SQLCODE provides more specific information and is better used for debugging problems in applications.
- All errors are reported back to the application program, except those for which the path back to the application has been destroyed by main memory corruption.
- If an application issues a SERRC with exit or if the application takes an error causing the ECB to exit, an implied ROLLBACK is issued to the remote AS if a unit of work was in progress.[6] All changes made since the last SQL COMMIT command are backed out.
- If an application issues an EXITC with a unit of work in progress, an implied COMMIT is issued for this application. Any changes made to the database are now permanent, and other users can view them. If for some reason the implied COMMIT fails, an implied ROLLBACK is attempted to back out all the changes made since the last successful SQL COMMIT command, if one was issued. The implied COMMIT feature must be used carefully.

  For example, if an application issues an SQL INSERT command and then exits without explicitly issuing an SQL COMMIT command, an implied COMMIT will be issued. TPF then issues a dump. If the implied COMMIT fails for any reason, an implied ROLLBACK is issued, and the SQL INSERT command into the table is backed out. Because the application has already exited, there is no way to report that the SQL INSERT command was backed out. This can cause the loss of data integrity. A case where the implied COMMIT could be used is when you are querying data. Because no tables are updated, if the SQL COMMIT command fails, it may not be important to the application.

## Preparing an Application

Figure 15 on page 33 shows the basic flow of transforming an application from source code to object code on TPF and into a bind file on DB2 through the DB2 bind process. The bind process converts output from a DB2 precompiler (DBRM) to the usable control structure known as a *bind file*. During this process, complete error checking and access strategy is performed for each SQL command. In ISO-C programs, you can perform binding directly on DB2 offline (static binding) before

6. An implied ROLLBACK is when an application voids any uncommitted changes. See the *IBM DATABASE 2 SQL Reference* for additional information on this command.

you run the application. Alternatively, TPF can perform a bind at run time (run-time binding).

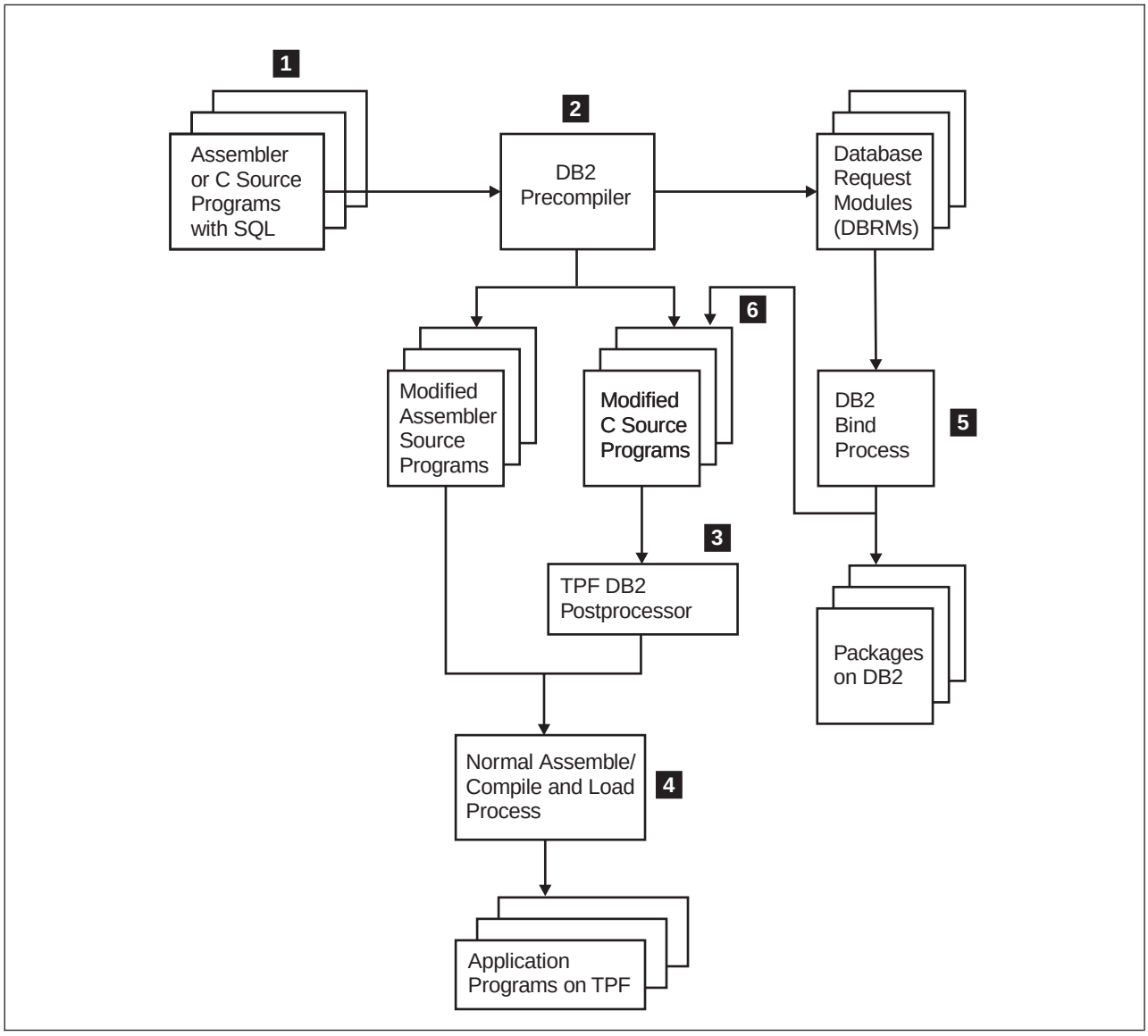


*Figure 15. Overview of TPF Application Preparation*

TPF application programs must be precompiled using the DB2 precompiler on the MVS system. The DB2 program preparation procedure is described in the *IBM Database 2 Application Programming and SQL Guide*.

The following explains each of the steps in Figure 15.

**Note:** You must ensure that the object code from the modified source produced from the DB2 precompiler matches the package produced by the database-request-module bind process from the DB2 precompiler.

1. The TPF application is written in C or assembler with imbedded SQL commands.
2. The DB2 precompiler extracts the SQL commands from the TPF application and places them in a database request module (DBRM). The DBRM is a data set member containing information about SQL commands. This data set contains

information on all SQL commands used in your application program. Information is included about how the SQL commands are executed and DB2's access strategy.

In place of the SQL commands, the DB2 precompiler places host language system calls into the application programs.

The following parameters are important to the DB2 precompiler:

**DEC** Specifies the maximum precision to be used in decimal arithmetic operations. This parameter must be set to DEC(15) for use with TPFAR. This parameter is ignored for C programs.

**HOST** Informs the DB2 precompiler about which host language is being used. This parameter must be set to one of the following:
- HOST(C) for TARGET(TPF) programs or ISO-C programs
- HOST(ASM) for assembler programs.

See "Using the TPF DB2 Postprocessor (TPF DB2PP)" on page 35 for more information about the TPF DB2PP HOST parameter.

**VERSION** Allows the DB2 precompiler to separate two different applications with the same name, so that the bind files created are not the same. This is important when supporting multiple copies of the same program (that is, when using the TPF online loader and fallback packages).

Supporting multiple versions can also be accomplished by changing the member name of the DBRM data set where the DB2 precomplier places the output DBRM. This member name is what DB2 uses to correlate different DB2 packages. By changing the DBRM member for different versions of the same program, multiple copies of the same program can have active DB2 packages. See step 5 on page 35 for more information on DB2 packages.

**Note:** For DB2 Version 3, you must specify the CONNECT(1) precompiler option to specify the rules for the CONNECT (Type 1) statement. The TPF system does not support the CONNECT (Type 2) option.

See the *IBM DATABASE 2 Command and Utility Reference* for more information regarding the DB2 precompiler options.

3. C language programs that are output from the DB2 precompiler are then run through the TPF DB2 postprocessor (TPF DB2PP), an offline program generated by SIP that alters some of the DB2 global variables placed in the program. (This step is not performed on assembler language programs.) See "Using the TPF DB2 Postprocessor (TPF DB2PP)" on page 35 for more information on this program.
4. The modified assembler or C programs are assembled or compiled. Care must be taken at this point to check the output. Because the DB2 precompiler has added code to the segments, you must check the size of the program to make sure that it fits in the segment if you are using assembler or TARGET(TPF). The amount of code added depends on individual SQL commands and their parameters. After this has been checked, the programs can be loaded to the TPF system using the online loader package.

   **Note:** When you are compiling an ISO-C program, the RENT option should be specified.

5. The DBRM, also referred to as a bind file, must be bound on the DB2 subsystem that is to connect to the TPF system. This bind process can run on the application server containing the target relational database ***before*** the TPF application can access this system. Target C programs and assembler programs must be bound offline. ISO-C programs can be bound offline or online at run-time. Run-time binding for ISO-C programs is automatic. For Target C and assembler programs, any attempt to run a TPF application before the DBRM has been bound results in an error.

   The output from the bind process is called a *package*. A package is an object containing a set of SQL statements that have been bound and are available for processing.

   The DB2 package is called at program execution time to execute the SQL commands extracted from the program during DB2 precompilation. (See step 2 on page 33.) The package is in a format understood only by the DB2. The DBRM could be transported to another MVS system, using MVS system utilities for example, and bound on the second system if the TPFAR application is to also connect and to work on this second system.

   When binding the DBRM, the following options are required:

   **PACKAGE or COLLECTION-ID**
   Must be the same as that used by TPFAR, which is made up of the TPF:

   - complex name
   - subsystem name
   - subsystem user name.

   Each of these pieces is separated by an underscore (_). For example, the parameter could be:

   ```
   TPFNET_BSS_HPN
   ```

   When running the bind process in a batch mode, the parameter is specified as PACKAGE. When running the bind process through the DB2 interactive panels, the parameter is specified at the COLLECTION-ID entry field. Both PACKAGE and COLLECTION-ID refer to the same value.

   **RELEASE** Specifies when the application server (AS) can release the resources reserved for this conversation or connection. This option must be COMMIT when using hotcons because TPF reuses this session if another program requests a connection to the same AS. If RELEASE was DEALLOCATE, these AS resources would continue to be held until the session was deallocated. When using hotcons, the deallocation may not happen for quite a while.

   **MEM** Indicates the member that is to be bound. This option must be the same as the member name used for the partitioned data set in which the DBRM output was placed in the DB2 precompiler.

   See the *IBM DATABASE 2 Command and Utility Reference* for more information about the bind options.

## Using the TPF DB2 Postprocessor (TPF DB2PP)

When writing a C application using SQL, the TPF DB2 postprocessor (TPF DB2PP) offline module needs to be run against the DB2 precompiler output. The TPF

DB2PP program alters and removes some variables that the DB2 precompiler sets up for the TPF C compiler with the TPF option to run successfully.

If the modified C program is not run through TPF DB2PP before C compiling, the program does not compile. Table 2 shows the strings that are altered by the TPF DB2PP program for C programs.

*Table 2. Strings Altered by TPF DB2PP for C Programs*

| Original String | Altered String |
|---|---|
| #pragma linkage (DSNHLI,OS); | (This field is removed by the TPF DB2PP program. The C header file `tpfarapi.h` will resolve the DSNHLI call.) |
| char SQLTEMP??( 19 ??); | static char SQLTEMP??(19??); |
| struct sqlca sqlca; | static struct sqlca sqlca; |
| struct SQLVTAG SQLVERS | static struct SQLVTAG SQLVERS |
| char DSNPNM??( 57 ??); | static char DSNPNM??( 57 ??); |

**Note:** Trigraph representations of square brackets are used in the preceding examples. The DB2 precompiler output may not always contain the trigraph representation. Regardless of which representation is used by the DB2 precompiler, DB2PP will always alter the string using the trigraph representation.

## Invoking TPF DB2PP

TPF DB2PP has different required PARM parameters, depending on whether an ISO-C or a TARGET(TPF) program is to be processed.

If an ISO-C program is being postprocessed, there are ***two*** required parameters. Any additional parameters result in an error.

**HOST** The HOST parameter specifies the language of the host program to the TPF DB2PP program. This parameter must be set to HOST(ISOC).

**PKGISO** The PKGISO parameter specifies the package isolation level, a security technique for DB2 packages, to be passed during the bind process. Acceptable values for this parameter are CS and RR. Any other value specified results in an error from TPF DB2PP. A typical PARM parameter specification is:

```
PARM='HOST(ISOC) PKGISOL(CS)'
```

If a TARGET(TPF) program is being postprocessed, there is ***one*** required parameter. Any additional parameters result in an error.

**HOST** The HOST parameter specifies the language of the host program to the TPF DB2PP program. You must set this parameter to HOST(C).

```
PARM='HOST(C)'
```

## Sample JCL

Figure 16 on page 37 shows sample JCL that you can use to process the output from the DB2 precompiler. The HOST(ISOC) and package isolation level PKISOL are also shown.

**Note:** The INFILE data set specified must be the data set where the DB2 precompiler placed the modified output file.

```
//MODIFY  EXEC PGM=DB2PP,REGION=4096K,PARM='HOST(ISOC) PKGISOL(CS)'
//STEPLIB   DD DSN=AR20000.DEVP.TEST.LK,DISP=SHR
//          DD DSN=ACP.LINK.INTG40.BSS,DISP=SHR
//          DD DSN=ACP.LINK.RLSE31.BSS,DISP=SHR
//          DD DSN=SYS1.CEE.SCEERUN,DISP=SHR
//SYSPRINT  DD SYSOUT=*
//INFILE    DD DSN=&&DSNHOUT,DISP=(OLD,DELETE)
//DBRMLIB   DD DSN=AR20000.DEVP.TEST.DBRM(QXMARF),DISP=(OLD,PASS)
//OUTFILE   DD DSN=&&TEMP,DISP=(NEW,PASS),UNIT=SYSDA,
//          DCB=(DSORG=PO,RECFM=FB,LRECL=80,BLKSIZE=80)
//*
//COPY    EXEC PGM=IEBGENER,REGION=58K
//SYSPRINT  DD SYSOUT=A
//SYSIN     DD DUMMY
//SYSUT2    DD DSN=TPFAR.TEST.SRCE(QXP1JV),DISP=OLD
//SYSUT1    DD DSN=&&TEMP,DISP=(OLD,PASS)
//*
```

*Figure 16. Sample JCL to Run TPF DB2PP*

## Database Resource Management (DBRM)

The DBRM consists of the SQL statements and variables pertaining to a particular program. The DB2 system uses the DBRM to optimize application program interaction. If the DBRM is bound by DB2 before running the application, the DBRM is considered statically bound. The DBRM can be bound at run time when it is combined with its ISO-C program by the postprocessor before the program is compiled. When the application runs, the DBRM is sent to the DB2 system to be bound.

The contents of the DBRM represent the variables, framework, and generated SQL calls required by the DB2 system when processing the SQL commands sent from the TPF system. The dbrm_array is a large character array that carries the variable portion and is added to the program source by the postprocessor. SQL statements bound by /***$$$ and $$$***/ are from the application program. They are followed by generated statements to use at run time. Each application that contains SQL statements must include the SQLCA and identify EXEC SQL INCLUDE SQLCA and EXEC SQL BEGIN DECLARE SECTION variable declarations.

```
static const char dbrm_array[] = {
'\xC4','\xC2','\xD9','\xD4','\x00','\x00','\x00','\xA0',     DBRM
'\xC2','\xD9','\xC1','\xD1','\x40','\x40','\x40','\x40',     BRAJ
'\xD8','\xE7','\xD4','\xC1','\xD9','\xC6','\x40','\x40',     QXMARF
'\x15','\xB5','\x7E','\x06','\x02','\x85','\x14','\x80',
'\x07','\x00','\xC4','\x00','\x00','\x02','\x00','\x00',
'\x00','\x00','\x00','\x00','\x00','\x00','\x00','\x00',
'\x00','\x00','\x00','\x00','\x00','\x00','\x00','\x00',
⋮
'\xC4','\xC5','\xC3','\xD3','\xC1','\xD9','\xC5','\x40',     DECLARE
'\xC3','\xF2','\x40','\xC3','\xE4','\xD9','\xE2','\xD6',     C2 CURSO:
'\xD9','\x40','\xC6','\xD6','\xD9','\x40','\xE2','\xC5',     R FOR SE
'\xD3','\xC5','\xC3','\xE3','\x40','\xC3','\xE4','\xE2',     LECT CUS
'\xE3','\xD5','\xD6','\x40','\x6B','\x40','\xD5','\xC1',     T,N O NA
'\xD4','\xC5','\x40','\x6B','\x40','\xD7','\xC2','\xC1',     ME , PBA
'\xD3','\x40','\x6B','\x40','\xD4','\xC1','\xE7','\xC2',     L , MAXB
'\xC1','\xD3','\x40','\xC6','\xD9','\xD6','\xD4','\x40',     AL FROM
'\xE2','\xC1','\xC1','\xC9','\xC4','\x40','\x4B','\x40',     SAAID .
'\xE2','\xC1','\xF1','\xC3','\xE4','\xE2','\xE3','\x40',     SA1CUST
'\xE6','\xC8','\xC5','\xD9','\xC5','\x40','\xD4','\xC1',     WHERE MA
```

```
'\xE7','\xC2','\xC1','\xD3','\x40','\xD5','\xD6','\xE3',     XBAL NOT
'\x40','\xC2','\xC5','\xE3','\xE6','\xC5','\xC5','\xD5',     BETWEEN
'\x40','\xF2','\xF0','\xF0','\xF0','\xF0','\x4B','\xF0',     20000.0
'\x40','\xC1','\xD5','\xC4','\x40','\xF1','\xF0','\xF0',     AND 100
'\xF0','\xF0','\xF0','\xF0','\x4B','\xF0','\x40','\xD6',     0000.0 O
'\xD9','\xC4','\xC5','\xD9','\x40','\xC2','\xE8','\x40',     ORDER BY
'\xD7','\xC2','\xC1','\xD3','\x40','\xC1','\xE2','\xC3',     PBAL ASC
'\x40','\x00','\x00','\x00','\x00','\x40','\x40','\x40',

:
'\x40','\x40','\x40','\x40','\x40','\x40','\x40','\x40',
'\xC4','\xC2','\xD9','\xD4','\x00','\x00','\x00','\x24',     DBRM
'\x00','\x01','\x00','\x00','\x00','\x00','\x00','\x9E',
'\x00','\x00','\x00','\x14','\x00','\x00','\x00','\x08',
'\xD6','\xD7','\xC5','\xD5','\x40','\xC3','\xF2','\x40',     OPEN C2
'\x00','\x00','\x00','\x00','\x40','\x40','\x40','\x40',
'\x40','\x40','\x40','\x40','\x40','\x40','\x40','\x40',
:
'\x40','\x40','\x40','\x40','\x40','\x40','\x40','\x40',
'\x40','\x40','\x40','\x40','\x40','\x40','\x40','\x40',
'\x40','\x40','\x40','\x40','\x40','\x40','\x40','\x40',
};

/*********************************************************************
**        THIS PRODUCT CONTAINS "RESTRICTED MATERIALS OF IBM"
**        COPYRIGHT = 5748-T13 (C) COPYRIGHT IBM CORP 1979,1989
**        LICENSED MATERIAL - PROGRAM PROPERTY OF IBM
**        REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083
*********************************************************************
**
** FUNCTION NAME...qxma_saamdc10
** SEGMENT NAME....QXMA                                            **
**                                                                  *
**   Load module name:   QXOZ                                       *
**                                                                  *
**
*********************************************************************
:
**
** EXTERNAL REFERENCES...printf, qxo2_check, qxo3_check
**
**
** HEADER FILES #INCLUDED...c$artst0
*/
#ifdef _TARGET_TPF
#error This segment cannot be compiled using TARGET(TPF)
#endif


  /*******************************************************************/
  /*                                                                 */
  /* TESTCASE NAME:         SAAMDC10          MEMBER:  D011YNN       */
  /*                                                                 */
  /* TESTCASE DESCRIPTION:  Queries and Predicates                   */
  /*                                                                 */
  /* CREATION DATE:         5/5/1989                                 */
  /*                                                                 */
  /* SPECIAL TESTING NOTES: None                                     */
  /*                                                                 */
  /* TERMINATION STATUS:    Cond code 0                              */
  /*                                                                 */
  /*******************************************************************/



typedef struct
             { short   SQLPLLEN;
```

```
                short   SQLFLAGS;
                short   SQLCTYPE;
                char    SQLPROGN[8];
                short   SQLTIMES[4];
                short   SQLSECTN;
                char   *SQLCODEP;
                char   *SQLVPARM;
                char   *SQLAPARM;
                short   SQLSTNUM;
                short   SQLSTYPE;
                char    SQLPKISL[2];
                const char   *DBRM_PTR;
                long    DBRM_SIZE;
              } SQLPLIST;
typedef struct
              { short    SQLTYPE;
                short    SQLLEN;
                char    *SQLADDR;
                char    *SQLIND;
              } SQLELTS;
typedef SQLELTS     *SQLELTS_PTR;
static char SQLTEMP??(19??) ;
struct SQLVTAG
    { char    *VERSPRE ;
      char    *VERSSTR ;
    } ;
static struct SQLVTAG SQLVERS = {
 "VER.",
 "RF"};
static char DSNPNM??(57??) ;

/***$$$
  EXEC SQL INCLUDE SQLCA
$$$***/
#ifndef SQLCODE
struct sqlca
  { unsigned  char     sqlcaid[8];
              long     sqlcabc;
              long     sqlcode;
              short    sqlerrml;
    unsigned  char     sqlerrmc[70];
    unsigned  char     sqlerrp[8];
              long     sqlerrd[6];
    unsigned  char     sqlwarn[11];
    unsigned  char     sqlstate[5];
            } ;
#define SQLCODE  sqlca.sqlcode
#define SQLWARN0 sqlca.sqlwarn[0]
#define SQLWARN1 sqlca.sqlwarn[1]
#define SQLWARN2 sqlca.sqlwarn[2]
#define SQLWARN3 sqlca.sqlwarn[3]
#define SQLWARN4 sqlca.sqlwarn[4]
#define SQLWARN5 sqlca.sqlwarn[5]
#define SQLWARN6 sqlca.sqlwarn[6]
#define SQLWARN7 sqlca.sqlwarn[7]
#define SQLWARN8 sqlca.sqlwarn[8]
#define SQLWARN9 sqlca.sqlwarn[9]
#define SQLWARNA sqlca.sqlwarn[10]
#define SQLSTATE sqlca.sqlstate
#endif
static struct sqlca sqlca ;

 #include <c$artst0.h>


  /* The following macros are defined to check if two given values
     of various types are equivalent and print out their differences. *
```

```
  static char d_tmp1[20], d_tmp2[20];
  #define Str_Comp(X, Y) (strncmp(X, Y, strlen(Y)) == 0)
  #define Dou_Comp(X, Y) ( \
          sprintf(d_tmp1, "%e\0", X), \
          sprintf(d_tmp2, "%e\0", Y), \
          (strcmp(d_tmp1, d_tmp2) == 0))
  #define PDiff_s(X, Y) if (!(Str_Comp(X, Y))) \
          printf(#X "    : %s\nexpected : %s\n", X, Y)
  #define PDiff_g(X, Y) if (!(Dou_Comp(X, Y))) \
          printf(#X " : %g  expected : %g\n", X, Y)
  #define PDiff_d(X, Y) if (X != Y) \
          printf(#X " : %d  expected : %d\n", X, Y)


extern void qxma_saamdc10(
                   long *return_ptr,  /* No. of Errors encountered  */
                   long *warning_ptr  /* No. of Warnings encountered */
                        )
{
  int    warn = 0;
  long   return_code;
  long   warning_code;


/***$$$
  EXEC SQL BEGIN DECLARE SECTION
$$$***/

    char    host1[7],  host2[40];
    double  host8, host9;

/***$$$
  EXEC SQL END DECLARE SECTION
$$$***/



  return_code=*return_ptr;
  warning_code=*warning_ptr;


   /************************************************************
                    DECLARE CURSOR C2
    ************************************************************/


/***$$$
  EXEC SQL
    DECLARE C2 CURSOR FOR
      SELECT CUSTNO, NAME, PBAL, MAXBAL
      FROM SAAID.SA1CUST
      WHERE MAXBAL NOT BETWEEN 20000.0 AND 1000000.0
      ORDER BY PBAL ASC
  $$$***/



  qxo2_check("DECLARE CURSOR C2",&sqlca,&return_code);

   /************************************************************
                    OPEN CURSOR C2
    ************************************************************/


/***$$$
  EXEC SQL OPEN C2
```

```
$$$***/
  {
  SQLPLIST SQLPLIST2 =
  {40, 0, 50, "QXMARF  ", 0, 0, 0 ,0,
   1, 0, 0, 0, 158, 3};
  SQLELTS_PTR SQLELTS_PTR2;
  SQLPLIST2.SQLCODEP = (char *) &sqlca;
  SQLPLIST2.SQLTIMES[0] = 0x15B5;
  SQLPLIST2.SQLTIMES[1] = 0x8055;
  SQLPLIST2.SQLTIMES[2] = 0x15B9;
  SQLPLIST2.SQLTIMES[3] = 0xE240;
  SQLPLIST2.SQLPKISL[0] =  0x24;
  SQLPLIST2.SQLPKISL[1] =  0x42;
  SQLPLIST2.DBRM_PTR =  dbrm_array;
  SQLPLIST2.DBRM_SIZE =  sizeof(dbrm_array);
  SQLPLIST2.SQLFLAGS =
  SQLPLIST2.SQLFLAGS | 0x1;
  DSNHLI ( (unsigned int * ) &SQLPLIST2);
  }


  qxo2_check("OPEN CURSOR C2",&sqlca,&return_code);

   /************************************************************
                    TEST FETCH CURSOR C2
    ************************************************************/

.
.
.
 }
```

## Using the Same Cursor in Multiple Programs

Because of the TPF system restriction to a 4 KB program size for assembler and TARGET(TPF) programs, you may have to use the same cursor throughout multiple TPF segments (the SQL source programs). If so, you must make the following modifications to your SQL source programs.

Segment size is not a problem for ISO-C programs, so this section is restricted to TARGET(TPF) and assembler considerations.

The DB2 precompiler needs to have all of the programs in one file in order to create the DBRM. The individual TPF segments must be placed into a single source file for the DB2 precompiler to work on. After the DB2 precompiler has finished, a single modified source output is returned. This modified source output contains the updated source code with the SQL commands removed. If the host language used was C, the modified source output needs to be run through the DB2PP processor before continuing. The modified source file now has to be separated into the different original TPF source program segments. Figure 17 on page 42 shows this process.

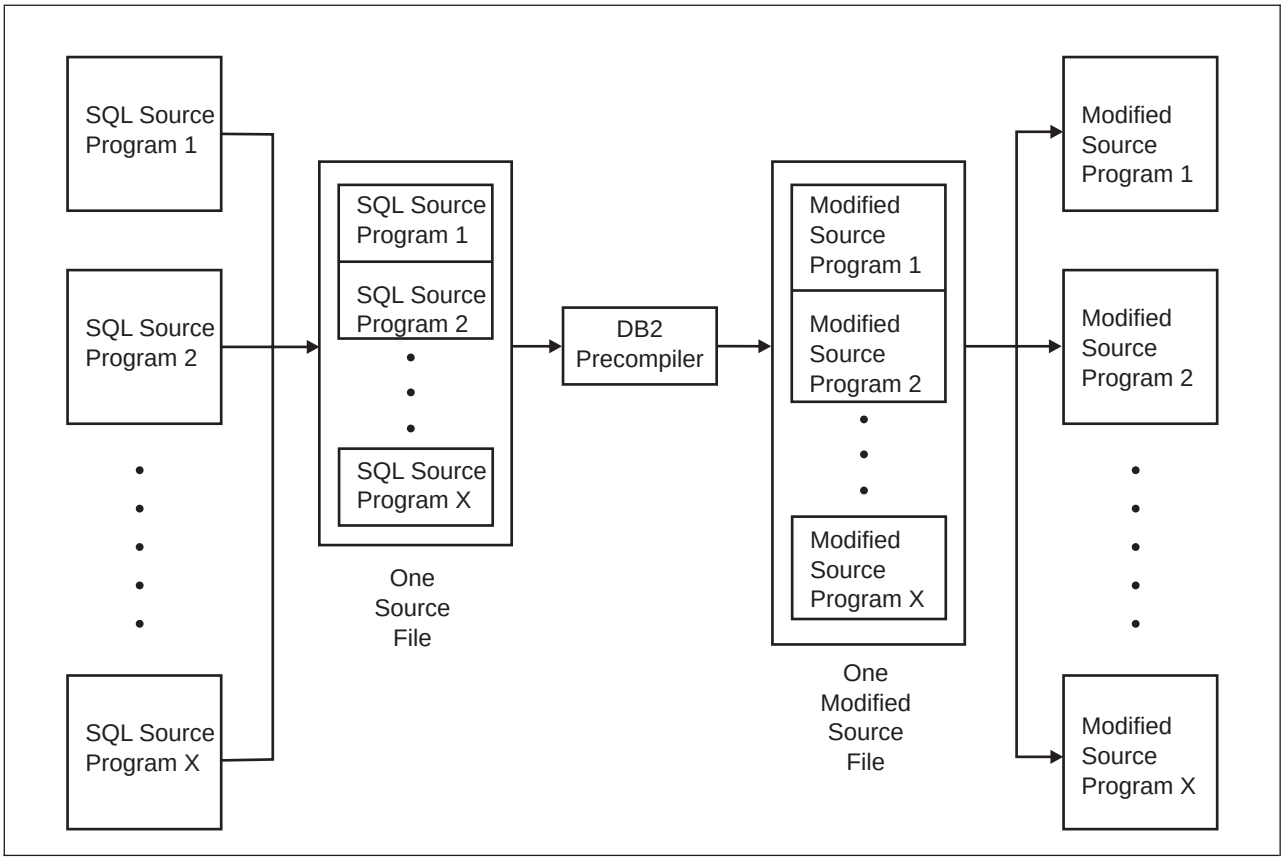


*Figure 17. Modifying Non-ISO-C SQL Source Programs for the DB2 Precompiler When Using the Same Cursor*

After the file has been separated into the different original segments, modifications need to be made, depending upon the host language used.

**Note:** When using the same cursor throughout multiple TPF segments, the same host language must be used in all of the segments.

## TARGET C Language Modifications

When the TPF segments are coded in C language, the large source file needs to be precompiled and then run through the TPF DB2PP program to process the precompiled output. After these have both been run with successful completion, the top of the returned source file in the first TPF segment contains the lines of code shown in Figure 18 on page 43.

**Note:** If the VERSION option was not used on the precompile run, the last seven lines starting with the SQLVTAG structure are not present.

```
typedef struct
               { short   SQLPLLEN;
                 short   SQLFLAGS
                 short   SQLCTYPE
                 char    SQLPROGN[8];
                 short   SQLTIMES[4];
                 short   SQLSECTN;
                 char   *SQLCODEP;
                 char   *SQLVPARM;
                 char   *SQLAPARM;
                 short   SQLSTNUM;
                 short   SQLSTYPE;
               } SQLPLIST;
typedef struct
               { short    SQLTYPE;
                 short    SQLLEN;
                 char    *SQLADDR;
                 char    *SQLIND;
               } SQLELTS;
typedef SQLELTS     *SQLELTS_PTR;
static char SQLTEMP??(19??) ;
struct SQLVTAG
    { char    *VERSPRE ;
      char    *VERSSTR ;
    } ;
static struct SQLVTAG SQLVERS = {
 "VER.",
 "JT"};
static char DSNPNM??(57??) ;
```

*Figure 18. C Code to Copy to Each TPF Segment*

When the modified output is separated into the different TPF segments, this code must be copied to each of the segments. When adding this code to the each segment, this code should be placed at the top of the individual segments after any ***#include***s that are used. (The ***#include*** is a C programming structure.)

The individual TPF segments can then be compiled and loaded, and the DBRM can be bound.

## Assembler Modifications

The assembler modifications are similar to those for C language.

In the modified assembler source file that is returned, two areas need to be copied: one at the top of the modified assembler (shown in Figure 19 on page 44), and the other at the bottom of the first TPF program, just before the END statement (shown in Figure 20 on page 44).

**Note:** Note that the SQLVERSP and SQLVERD1 tags are only found when the VERSION parameter is coded in the precompile options.

```
         MACRO
         SQLSECT &TYPE
         GBLC  &SQLSECT
         AIF ('&TYPE' EQ 'RESTORE').REST
&SQLSECT SETC  '&SYSECT'
         MEXIT
.REST    ANOP
&SQLSECT CSECT
         MEND
```

*Figure 19. Assembler Code Area Needed at the Top of Each Separated TPF Segment*

```
SQLVERSP DC    CL4'VER.' VERSION-ID PREFIX
SQLVERD1 DC    CL64'JT'        VERSION-ID
***$$$ SQL WORKING STORAGE
SQLDSIZ  DC    A(SQLDLEN) SQLDSECT SIZE
SQLDSECT DSECT
SQLPLIST DS    F
SQLPLLEN DS    H          PLIST LENGTH
SQLFLAGS DS    XL2        FLAGS
SQLCTYPE DS    H          CALL-TYPE
SQLPROGN DS    CL8        PROGRAM NAME
SQLTIMES DS    CL8        TIMESTAMP
SQLSECTN DS    H          SECTION
SQLCODEP DS    A          CODE POINTER
SQLVPARM DS    A          VPARAM POINTER
SQLAPARM DS    A          AUX PARAM PTR
SQLSTNUM DS    H          STATEMENT NUMBER
SQLSTYPE DS    H          STATEMENT TYPE
SQLPVARS DS    F,1CL12
SQLAVARS DS    F,1CL12
SQLTEMP  DS    CL18       TEMPLATE
         DS    0D
SQLDLEN  EQU   *-SQLDSECT
```

*Figure 20. Assembler Code Area Needed in Each TPF Segment before the END Statement*

Place the code in exactly the same places you found it in the first segment: at the top and before the END statement.

# Using TPF C with TPFAR

This section contains examples of TPF applications written in C that use TPFAR to access data in a relational database.

**Note:** The following applications demonstrate the use of TPFAR. These examples may not reflect the best method for implementing the application, but instead are intended to show different data access methods.

## The Root Segment for the Telephone Directory Application

Figure 21 on page 46 shows the root segment for the telephone directory application package. In the sample C code the following conventions have been established:

- The message block to be processed is on data level 0.
- The `puts` C function has been implemented by the user.

The basic format for all of the messages used in this package is as follows:

**rc /a add_parm**

where:

**rc** A routing code. This code is used to route messages to the application and is not used in the example of parsing the entry.

**a** A 1-character action code:
- **I** Insert
- **D** Display
- **R** Remove
- **U** Update.

**add_parm**
Depends on the action code.

```
#include <tpfeq.h>                /* Include libraries                */
#include <tpfapi.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

#pragma map(qxp0_exm_tests,"QXP0")
extern void qxp0_exm_tests();
#pragma map(qxp1_insert,"QXP1")
extern void qxp1_insert();
#pragma map(qxp2_delete,"QXP2")
extern void qxp2_delete();
#pragma map(qxp3_update,"QXP3")
extern void qxp3_update();
#pragma map(qxp4_display,"QXP4")
extern void qxp4_display();


/*********************************************************************/
/* This is the root function of the SQL example segments.  It        */
/* will parse the input looking for the action code, then            */
/* depending on the action code, it will call the appropriate        */
/* routine.                                                          */
/*********************************************************************/
void qxp0_exm_tests()
{
  struct mi0mi *blk;              /* Pointer to the input message:  */
                                  /*  The input message is on D0    */
  char  action;                   /* Action code entered:           */
                                  /*  This is from the parsed       */
                                  /*  message.                      */
```

*Figure 21. Root Segment (Part 1 of 3)*

```
/*********************************************************************/
/*  The database name is placed in ebx000 so that any function      */
/*  that needs it can use it and if it is changed, it will only     */
/*  have to be changed in one spot.                                  */
/*********************************************************************/

 memset ((void *) &ecbptr()->ebx000, ' ', 15);
 memcpy ((void *) &ecbptr()->ebx000,
                  "DB23TST",
                  sizeof("DB23TST")-1);
                           /*  The size of the string includes the */
                           /*  null at the end which needs to be   */
                           /*  placed at byte 16.                  */
 ecbptr()->ebx016 = '\0';


/*********************************************************************/
/* Set up a pointer to the input message and replace the EOM        */
/* character with a NULL.                                           */
/*                                                                  */
/*********************************************************************/

 blk = ecbptr()->ce1cr0;
 *strchr (blk->mi0acc,_EOM) = '\0';

/*********************************************************************/
/* Parse the message block.  The first parameter is unused, the     */
/* second parameter is the action.  The rest of the parameters      */
/* depend on what the first parameter is.  They will be scanned     */
/* individually based on the action. Notice that the action is      */
/* preceded by a /.                                                 */
/*                                                                  */
/*********************************************************************/
 sscanf(blk->mi0acc, "%*s /%1c",
                        &action);
```

*Figure 21. Root Segment (Part 2 of 3)*

```
/*********************************************************************/
/*  Based on the action entered, call the different routines.       */
/*  The action is converted to uppercase so that the case statement */
/*  can be made smaller.                                             */
/*                                                                   */
/*********************************************************************/

 switch (toupper(action))
 {
    case 'I':                    /* A new entry is to be inserted   */
          qxp1_insert();
          break;

    case 'D':                    /* A display is requested.         */
          qxp4_display();
          break;

    case 'R':                    /* A removal of an entry is needed. */
          qxp2_delete();
          break;

    case 'U':                    /* An update of an entry is needed. */
          qxp3_update();
          break;

    default:
          printf(
          "Action entered is incorrect.  Please enter a I, D, U or R");
          break;
 }
 exit(0);
}
```

*Figure 21. Root Segment (Part 3 of 3)*

## Inserting a Telephone Directory Entry

The program listed in Figure 22 on page 50 inserts a new employee entry into the corporate telephone directory.

The format of the parameters is:

**rc /I last_name/first_name/middle_initial/country/area/phone_number**

where:

**rc** A routing code. This code is used to route messages to the application and is not used in the example of parsing the entry.

**I** Insert action code.

**last_name**
The last name of the person to add to the database.

**first_name**
The first name of the person to add to the database.

**middle_initial**
The middle initial of the person to add to the database.

**country**
The country code of the person to add to the database.

**area**
The area code of the person to add to the database.

**phone_number**
The phone number of the person to add to the database.

**Notes:**

1. The / character separates the different parameters.
2. The employee number is not needed for an insert because a unique number is assigned by the program.

For example, to insert an entry for Juan Martinez into the PHONE_DIRECTORY table, you would type:

```
rc /I MARTINEZ/JUAN/C/52/748/2221923
```

```
#include <tpfeq.h>                /* Include the libraries           */
#include <tpfapi.h>
#include <tpfarapi.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

#pragma map(qxp1_insert,"QXP1")
extern void qxp1_insert();
static void Check();

EXEC SQL INCLUDE SQLCA;           /* Include the SQLCA                */


  /*********************************************************************/
  /* Check:                                                            */
  /*    This function verifies the SQLSTATE returned.  If it is not 0,*/
  /* a message is printed indicating what the SQLSTATE returned was.  */
  /*                                                                   */
  /* A customer implementing a function like this may want to include */
  /* a much more robust error handling and recovery.                   */
  /*********************************************************************/

static void Check()
{
  if (memcmp(sqlca.sqlstate,"00000",sizeof(sqlca.sqlstate)) != 0) {
    printf("FAILED\n %d %.5s\n",sqlca.sqlcode,sqlca.sqlstate);
    exit(5);
  }
}
```

*Figure 22. TPF Program to Insert an Employee into the PHONE_DIRECTORY Table (Part 1 of 6)*

```
 /*********************************************************************/
 /* qxp1_insert:                                                      */
 /*    This function inserts a new record into a database with        */
 /* the information passed from the user.                             */
 /*                                                                   */
 /*********************************************************************/
void qxp1_insert()
{
  short int num_input;           /* The number of variables sscanf   */
                                 /* has correctly set up.  This is   */
                                 /* used to check the validity of    */
                                 /* the parameters.                  */
  struct mi0mi *blk;             /* Pointer to the input message     */


 /*********************************************************************/
 /* Declare all the variables that SQL needs to know about.           */
 /*********************************************************************/


  EXEC SQL BEGIN DECLARE SECTION;

 /*********************************************************************/
 /*  Set up a structure for the directory record.  This structure     */
 /*  is set up in the same order as the CREATE TABLE parameters       */
 /*  when the table was created on DB2.                               */
 /*********************************************************************/
    struct {
      char last_name[18];
      char first_name[9];
      char middle_initial[2];
      char country_code[5];
      char area_code[6];
      char phone_number[13];
      short int employee_number;
      char timestamp[27];
    } dir_record;

    char buf[16];                /* This will be used to point to  */
                                 /* the database to connect to.    */

    short int indnull;           /* Used to indicate a null        */
                                 /* variable.                      */

  EXEC SQL END DECLARE SECTION;
```

*Figure 22. TPF Program to Insert an Employee into the PHONE_DIRECTORY Table (Part 2 of 6)*

```
 /*********************************************************************/
 /* Issue the connect with the name of the database to connect       */
 /* to.  When done, the check function will check the return code,   */
 /* and if invalid, exit.  Ebx000 was set up in the root segment     */
 /* with the name of the database to connect to.                     */
 /*********************************************************************/

  strcpy(buf, &ecbptr()->ebx000);
  EXEC SQL
    CONNECT TO :buf;
  Check();



 /*********************************************************************/
 /* Parse the message block.  The first parameter is unused, and     */
 /* the second was already parsed.  The rest of the parameters are   */
 /* all the information needed for the insert.                       */
 /* Notice that all of the parameters are separated by a /.          */
 /*                                                                  */
 /*********************************************************************/
  blk = ecbptr()->ce1cr0;
  num_input = sscanf(blk->mi0acc,
"%*s /%*1c %17[^/]/%8[^/]/%1[^/]/%4[^/]/%5[^/]/%12s",
           dir_record.last_name,
                   dir_record.first_name,
                          dir_record.middle_initial,
                                 dir_record.country_code,
                                        dir_record.area_code,
                                               dir_record.phone_number);

 /*********************************************************************/
 /* The number of conversions must be 6 which is the number of       */
 /* items needed for input.                                          */
 /*                                                                  */
 /*********************************************************************/

  if (num_input != 6)
  {
      printf("The input is invalid.  The format is:\n\
Last name/First name/MI/Country/Area/Phone number\n");
      exit(0);
  }
```

*Figure 22. TPF Program to Insert an Employee into the PHONE_DIRECTORY Table (Part 3 of 6)*

```
/*********************************************************************/
/* The new employee will automatically have the next employee       */
/* number assigned to them.  The table must be locked first so that */
/* anyone else making an update will not get the same employee      */
/* number automatically assigned.                                   */
/*                                                                  */
/* The new employee number is one more than the current maximum     */
/* employee number in the table.                                    */
/*                                                                  */
/*********************************************************************/

 EXEC SQL
       LOCK TABLE TPFNET.PHONE_DIRECTORY IN EXCLUSIVE MODE;
 Check();

 EXEC SQL
       SELECT MAX(EMPLOYEE_NUMBER)
        INTO :dir_record.employee_number:indnull
        FROM TPFNET.PHONE_DIRECTORY;
 Check();

/*********************************************************************/
/* Check the return to see if the answer was null.  If it is null,  */
/* then there are no entries in the table, so set the employee      */
/* number to zero, indicating no employee numbers yet assigned.     */
/*********************************************************************/

 if (indnull < 0)
 {
    dir_record.employee_number = 0;
 }

 dir_record.employee_number++;
```

*Figure 22. TPF Program to Insert an Employee into the PHONE_DIRECTORY Table (Part 4 of 6)*

```
/*********************************************************************/
/* Insert the record into the database.                              */
/*********************************************************************/

 EXEC SQL
       INSERT INTO TPFNET.PHONE_DIRECTORY
       (LAST_NAME, FIRST_NAME, MIDDLE_INITIAL, COUNTRY_CODE,
        AREA_CODE, PHONE_NUMBER, EMPLOYEE_NUMBER,
        TIME_STAMP)
       VALUES
       (:dir_record.last_name,
        :dir_record.first_name,
        :dir_record.middle_initial,
        :dir_record.country_code,
        :dir_record.area_code,
        :dir_record.phone_number,
        :dir_record.employee_number,
        CURRENT TIMESTAMP);
 Check();


/*********************************************************************/
/* The insert has completed.  We now want to commit the work so that*/
/* others can see our updates.                                       */
/*                                                                   */
/*********************************************************************/

 EXEC SQL
       COMMIT;
 Check();
```

*Figure 22. TPF Program to Insert an Employee into the PHONE_DIRECTORY Table (Part 5 of 6)*

```
 /*********************************************************************/
 /* Let us now double check to show the user what is actually out    */
 /* on the database.                                                  */
 /*                                                                   */
 /*********************************************************************/

  EXEC SQL
        SELECT *
        INTO
         :dir_record
        FROM TPFNET.PHONE_DIRECTORY WHERE EMPLOYEE_NUMBER =
         :dir_record.employee_number;
  Check();

 /*********************************************************************/
 /* Show the results and exit.                                        */
 /*                                                                   */
 /*********************************************************************/

  printf(
"Employee added to database\n\
    last name :      %s\n\
    first name :     %s\n\
    middle initial : %s\n\
    employee number: %hd\n",
           dir_record.last_name,
           dir_record.first_name,
           dir_record.middle_initial,
           dir_record.employee_number);


  exit(0);
}
```

*Figure 22. TPF Program to Insert an Employee into the PHONE_DIRECTORY Table (Part 6 of 6)*

## Removing a Telephone Directory Entry

The program listed in Figure 23 on page 57 removes an employee entry from the corporate telephone directory.

The format of the parameters is:

**rc /R last_name/first_name[/employee_number]**

where:

**rc** A routing code. This code is used to route messages to the application and is not used in the example of parsing the entry.

**R** Remove action code.

**last_name**
The last name of the person to be removed.

**first_name**
The first name of the person to be removed.

**employee_number**
An optional parameter that indicates a specific employee number. If this parameter is included, only the person with this last name, first name, and employee number is removed. If this parameter is omitted, all people with the same last and first names are removed.

**Notes:**

1. The / character separates the different parameters.
2. The [ and ] characters denote an optional parameter.

For example, to remove an entry for Robert Durr from the PHONE_DIRECTORY table, you would type:

```
rc /R DURR/ROBERT/1
```

```
#include <tpfeq.h>                /* Include libraries               */
#include <tpfapi.h>
#include <tpfarapi.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

/*********************************************************************/
/*                                                                   */
/*  Declare the internal and external functions                      */
/*                                                                   */
/*********************************************************************/
static void Check();
#pragma map(qxp2_delete,"QXP2")
extern void qxp2_delete();

EXEC SQL INCLUDE SQLCA;           /* Include the SQLCA               */


  /*********************************************************************/
  /* Check:                                                            */
  /*    This function verifies the SQLSTATE returned.  If it is not 0,*/
  /* a message is printed indicating what the SQLSTATE returned was.  */
  /*                                                                   */
  /* A customer implementing a function like this may want to include */
  /* a much more robust error handling and recovery.                   */
  /*********************************************************************/

static void Check()
{
  if (memcmp(sqlca.sqlstate,"00000",sizeof(sqlca.sqlstate)) != 0) {
    printf("FAILED\n %d %.5s\n",sqlca.sqlcode,sqlca.sqlstate);
    exit(5);
  }
}
```

*Figure 23. TPF Program to Remove a Specific Entry in the PHONE_DIRECTORY Table (Part 1 of 4)*

```
 /*********************************************************************/
 /*  This function will delete a entry from the database.             */
 /*                                                                   */
 /*********************************************************************/
void qxp2_delete()
{
  short int num_input;             /* The number of variables sscanf */
                                   /* has correctly set up.  This is */
                                   /* used to check the validity of  */
                                   /* the parameters.                */
  struct mi0mi *blk;               /* Pointer to the input message   */


 /*********************************************************************/
 /* Declare all the variables that SQL needs to know about.           */
 /*********************************************************************/


  EXEC SQL BEGIN DECLARE SECTION;

 /*********************************************************************/
 /*  Set up a structure for the directory record.  This structure     */
 /*  is set up in the same order as the CREATE TABLE parameters       */
 /*  were when the table was created on DB2.                          */
 /*********************************************************************/
    struct {
      char last_name[18];
      char first_name[9];
      char middle_initial[2];
      char country_code[5];
      char area_code[6];
      char phone_number[13];
      short int employee_number;
      char timestamp[27];
    } dir_record;

    char buf[16];                 /* This will be used to point to   */
                                   /* the database to connect to.     */
  EXEC SQL END DECLARE SECTION;
```

*Figure 23. TPF Program to Remove a Specific Entry in the PHONE_DIRECTORY Table (Part 2 of 4)*

```
 /*********************************************************************/
 /* Issue a connect with the name of the database to connect         */
 /* to.  When done, the check function will check the return code,   */
 /* and if invalid, exit.                                            */
 /*********************************************************************/

  strcpy(buf, &ecbptr()->ebx000);
  EXEC SQL
    CONNECT TO :buf;
  Check();



/*********************************************************************/
 /* Parse the message block.  The first parameter is unused, and     */
 /* the second was already parsed.  The next two must be the         */
 /* last name/first name combination.  The employee number is        */
 /* optional.                                                        */
 /*                                                                  */
 /*********************************************************************/
  blk = ecbptr()->ce1cr0;
  num_input = sscanf(blk->mi0acc,
     "%*s /%*1c %17[^/]/%8[^/]/%hd",
                dir_record.last_name,
                        dir_record.first_name,
                               &dir_record.employee_number);
```

*Figure 23. TPF Program to Remove a Specific Entry in the PHONE_DIRECTORY Table (Part 3 of 4)*

```
 switch (num_input)
 {
   /*****************************************************************/
   /*  We only have the last name and first name.  The delete will  */
   /*  be based on just this information.                           */
   /*****************************************************************/

    case 2:
    {
       EXEC SQL
         DELETE FROM TPFNET.PHONE_DIRECTORY
         WHERE LAST_NAME = :dir_record.last_name AND
               FIRST_NAME = :dir_record.first_name;

       Check();
       break;
    }
   /*****************************************************************/
   /*  We have the last name and first name and employee number.    */
   /*  This will guarantee a unique identification.                 */
   /*****************************************************************/
    case 3:
    {
       EXEC SQL
         DELETE FROM TPFNET.PHONE_DIRECTORY
         WHERE LAST_NAME = :dir_record.last_name AND
               FIRST_NAME = :dir_record.first_name AND
               EMPLOYEE_NUMBER = :dir_record.employee_number;

       Check();
       break;
    }
    default:
    {
       printf("The input for the delete was invalid. Please check.");
       exit(0);
    }
 }


/*******************************************************************/
/*  Commit the work so others may continue.                        */
/*******************************************************************/

 EXEC SQL
       COMMIT;
 Check();

/*******************************************************************/
/*  Tell the user that we have completed.                          */
/*******************************************************************/

 printf("Employee %s %s was removed from the database.",
                  dir_record.first_name,
                      dir_record.last_name);
 exit(0);
}
```

*Figure 23. TPF Program to Remove a Specific Entry in the PHONE_DIRECTORY Table (Part 4 of 4)*

## Updating a Telephone Directory Entry

The program listed in Figure 24 on page 62 updates an employee's entry in the corporate telephone directory.

The format of the parameters is:

**rc /U field_to_update/employee_number/new_value**

where:

**rc** A routing code. This code is used to route messages to the application and is not used in the example of parsing the entry.

**U** Update action code.

**field_to_update**
One of the following employee entry fields to be updated:

- **L** Last name
- **F** First name
- **M** Middle initial
- **C** Country code
- **A** Area code
- **P** Phone number

**employee_number**
The employee number of the entry to be updated.

**new_value**
The new value for the field indicated in the **field_to_update** parameter.

**Note:** The / character separates the different parameters.

For example, to update the country in the PHONE_DIRECTORY table entry for Mary Stewart, you would type:

```
rc /U C/3/1
```

```
#include <tpfeq.h>                /* Include Libraries              */
#include <tpfapi.h>
#include <tpfarapi.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>


/*********************************************************************/
/*                                                                   */
/*  Declare the internal and external functions.                     */
/*                                                                   */
/*********************************************************************/
static void Check();
#pragma map(qxp3_update,"QXP3")
extern void qxp3_update();

EXEC SQL INCLUDE SQLCA;           /* Include the SQLCA               */


  /*******************************************************************/
  /* Check:                                                          */
  /*    This function verifies the SQLSTATE returned.  If it is not 0,*/
  /* a message is printed indicating what the SQLSTATE returned was.  */
  /*                                                                 */
  /* A customer implementing a function like this may want to include */
  /* a much more robust error handling and recovery.                 */
  /*******************************************************************/

static void Check()
{
  if (memcmp(sqlca.sqlstate,"00000",sizeof(sqlca.sqlstate)) != 0) {
    printf("FAILED\n %d %.5s\n",sqlca.sqlcode,sqlca.sqlstate);
    exit(5);
  }
}


/*********************************************************************/
/* This routine updates the employee record requested.              */
/*********************************************************************/
void qxp3_update()
{
  short int num_input;             /* The number of variables sscanf */
                                   /* has correctly set up.  This is */
                                   /* used to check the validity of  */
                                   /* the parameters.                */
  struct mi0mi *blk;               /* Pointer to the input message   */
  char field;                      /* The field in the record that   */
                                   /* is being updated.              */
  char change[18];                /* The new value needed for the   */
                                   /* field.  The largest value is   */
                                   /* 18 for the last name.          */
```

*Figure 24. TPF Program to Update a Specific Entry in the PHONE_DIRECTORY Table (Part 1 of 8)*

```
/*********************************************************************/
/* Declare all the variables that SQL needs to know about.           */
/*********************************************************************/


 EXEC SQL BEGIN DECLARE SECTION;

/*********************************************************************/
/*  Set up a structure for the directory record.  This structure     */
/*  is set up in the same order as the CREATE TABLE parameters       */
/*  were when the table was created on DB2.  Two copies of this      */
/*  structure are needed, one for what the new record looks like,    */
/*  and one for what the old record looked like.                     */
/*********************************************************************/
   struct {
     char last_name[18];
     char first_name[9];
     char middle_initial[2];
     char country_code[5];
     char area_code[6];
     char phone_number[13];
     short int employee_number;
     char timestamp[27];
   } dir_record;

  struct {
     char last_name[18];
     char first_name[9];
     char middle_initial[2];
     char country_code[5];
     char area_code[6];
     char phone_number[13];
     short int employee_number;
     char timestamp[27];
  } dir_up_record;

   char buf[16];                /* This will be used to point to   */
                                /* the database to connect to.     */
   char column[10];
 EXEC SQL END DECLARE SECTION;

 EXEC SQL DECLARE U1 CURSOR FOR
       SELECT *
        FROM TPFNET.PHONE_DIRECTORY
        WHERE EMPLOYEE_NUMBER = :dir_record.employee_number
        FOR UPDATE OF LAST_NAME, FIRST_NAME, MIDDLE_INITIAL,
        COUNTRY_CODE, AREA_CODE, PHONE_NUMBER, TIME_STAMP;



/*********************************************************************/
/* Issue a connect with the name of the database to connect          */
/* to.  When done, the check function will check the return code,    */
/* and if invalid, exit.                                             */
/*********************************************************************/

 strcpy(buf, &ecbptr()->ebx000);
 EXEC SQL
   CONNECT TO :buf;
 Check();
```

*Figure 24. TPF Program to Update a Specific Entry in the PHONE_DIRECTORY Table (Part 2 of 8)*

```
 /*********************************************************************/
 /* Parse the message block.  The first parameter is unused, and      */
 /* the second was already parsed.  The third parameter is the field */
 /* to be updated, the fourth, the employee number to be updated,     */
 /* and the last, the new value of the field.                         */
 /*                                                                   */
 /*********************************************************************/
  blk = ecbptr()->ce1cr0;
  num_input = sscanf(blk->mi0acc,
   "%*s /%*1c %1c/%hd/%s",
                &field,
                   &dir_record.employee_number,
                      change);

 /*********************************************************************/
 /* All parameters are required.  If any are missing, then tell       */
 /* the user and exit.                                                */
 /*                                                                   */
 /*********************************************************************/

  if (num_input != 3)
  {
      printf("The input is invalid.  The format is:\n\
Field to change/employee number/new field");
      exit(0);
  }

  /*****************************************************************/
  /* Open the cursor for input.                                    */
  /*****************************************************************/
  EXEC SQL OPEN U1;
  Check();
```

*Figure 24. TPF Program to Update a Specific Entry in the PHONE_DIRECTORY Table (Part 3 of 8)*

```
/*********************************************************************/
/* Find the record that matches this request.                        */
/*                                                                   */
/*********************************************************************/


 EXEC SQL
       FETCH U1
        INTO :dir_record;

 if (memcmp(sqlca.sqlstate,"02000",sizeof(sqlca.sqlstate)) == 0)
 {
    printf("Employee Number %d not found.",dir_record.employee_number);
    exit(0);
 }
 Check();


/*********************************************************************/
/* Copy the request to the update record.                            */
/*                                                                   */
/*********************************************************************/

 memcpy(dir_up_record.last_name,
        dir_record.last_name, sizeof(dir_record));

 /*****************************************************************/
 /*                                                               */
 /*  Set up the update field based on what needs to be updated.   */
 /*  The first bytes of the string that are needed are used.      */
 /*                                                               */
 /*                                                               */
 /*****************************************************************/


 switch (toupper(field))
 {
```

*Figure 24. TPF Program to Update a Specific Entry in the PHONE_DIRECTORY Table (Part 4 of 8)*

```
 case 'L':
 {
   memcpy(dir_up_record.last_name, change,
            sizeof(dir_up_record.last_name)-1);

   /************************************************************/
   /*  The last character must be the null.  If the input      */
   /*  was less than the length, then the null is already      */
   /*  in place.  This is only for the case when the change    */
   /*  string is greater than or equal to the length of the    */
   /*  record.                                                 */
   /*                                                          */
   /************************************************************/

   dir_up_record.last_name[sizeof(dir_up_record.last_name)-1]
      = '\0';
   break;
 }

case 'F':
 {
   memcpy(dir_up_record.first_name, change,
            sizeof(dir_up_record.first_name)-1);

   /************************************************************/
   /*  The last character must be the null.  If the input      */
   /*  was less than the length, then the null is already      */
   /*  in place.  This is only for the case when the change    */
   /*  string is greater than or equal to the length of the    */
   /*  record.                                                 */
   /*                                                          */
   /************************************************************/

   dir_up_record.first_name[sizeof(dir_up_record.first_name)-1]
     = '\0';
   break;
 }
```

*Figure 24. TPF Program to Update a Specific Entry in the PHONE_DIRECTORY Table (Part 5 of 8)*

```
  case 'M':
  {
    memcpy(dir_up_record.middle_initial, change,
             sizeof(dir_up_record.middle_initial)-1);

    /*************************************************************/
    /*  The last character must be the null.  If the input       */
    /*  was less than the length, then the null is already       */
    /*  in place.  This is only for the case when the change     */
    /*  string is greater than or equal to the length of the     */
    /*  record.                                                  */
    /*                                                           */
    /*************************************************************/

dir_up_record.middle_initial
      [sizeof(dir_up_record.middle_initial)-1]
      = '\0';
    break;
  }

 case 'C':
  {
    memcpy(dir_up_record.country_code, change,
             sizeof(dir_up_record.country_code)-1);

    /*************************************************************/
    /*  The last character must be the null.  If the input       */
    /*  was less than the length, then the null is already       */
    /*  in place.  This is only for the case when the change     */
    /*  string is greater than or equal to the length of the     */
    /*  record.                                                  */
    /*                                                           */
    /*************************************************************/

    dir_up_record.country_code[sizeof(dir_up_record.country_code)-1]
      = '\0';
    break;
  }
```

*Figure 24. TPF Program to Update a Specific Entry in the PHONE_DIRECTORY Table (Part 6 of 8)*

```
      case 'A':
      {
        memcpy(dir_up_record.area_code, change,
                 sizeof(dir_up_record.area_code)-1);

        /*************************************************************/
        /*  The last character must be the null.  If the input       */
        /*  was less than the length, then the null is already       */
        /*  in place.  This is only for the case when the change     */
        /*  string is greater than or equal to the length of the     */
        /*  record.                                                  */
        /*                                                           */
        /*************************************************************/

        dir_up_record.area_code[sizeof(dir_up_record.area_code)-1]
          = '\0';
        break;
      }

      case 'P':
      {
        memcpy(dir_up_record.phone_number, change,
                 sizeof(dir_up_record.phone_number)-1);

        /*************************************************************/
        /*  The last character must be the null.  If the input       */
        /*  was less than the length, then the null is already       */
        /*  in place.  This is only for the case when the change     */
        /*  string is greater than or equal to the length of the     */
        /*  record.                                                  */
        /*                                                           */
        /*************************************************************/

       dir_up_record.phone_number
            [sizeof(dir_up_record.phone_number)-1]
          = '\0';
        break;
      }

      default:
        printf("Invalid type to update.");
        printf("Valid types are L, F, M, C, A, or P");
        exit(0);

  }
```

*Figure 24. TPF Program to Update a Specific Entry in the PHONE_DIRECTORY Table (Part 7 of 8)*

```
  EXEC SQL
        UPDATE TPFNET.PHONE_DIRECTORY
        SET LAST_NAME    = :dir_up_record.last_name,
         FIRST_NAME      = :dir_up_record.first_name,
         MIDDLE_INITIAL  = :dir_up_record.middle_initial,
         COUNTRY_CODE    = :dir_up_record.country_code,
         AREA_CODE       = :dir_up_record.area_code,
         PHONE_NUMBER    = :dir_up_record.phone_number,
         TIME_STAMP      = CURRENT TIMESTAMP
         WHERE CURRENT OF U1;

  Check();

  EXEC SQL
        COMMIT;
  Check();


  EXEC SQL
        SELECT *
        INTO
         :dir_up_record
        FROM TPFNET.PHONE_DIRECTORY WHERE EMPLOYEE_NUMBER =
         :dir_record.employee_number;

  Check();

   printf("Employee record updated");
   printf(
     "OLD %s %s %s %s %s %s %hd\n",
          dir_record.last_name,
             dir_record.first_name,
                dir_record.middle_initial,
                   dir_record.phone_number,
                      dir_record.country_code,
                         dir_record.area_code,
                            dir_record.employee_number);
   printf(
     "NEW %s %s %s %s %s %s %hd\n",
          dir_up_record.last_name,
             dir_up_record.first_name,
                dir_up_record.middle_initial,
                   dir_up_record.phone_number,
                      dir_up_record.country_code,
                         dir_up_record.area_code,
                            dir_up_record.employee_number);

  exit(0);
}
```

*Figure 24. TPF Program to Update a Specific Entry in the PHONE_DIRECTORY Table (Part 8 of 8)*

## Displaying Entries in the Telephone Directory

The program listed in Figure 25 on page 71 displays an employee entry in the corporate telephone directory.

The format of the parameter is:

**rc /D last_name[%][/first_name]**

where:

**rc** A routing code. This code is used to route messages to the application and is not used in the example of parsing the entry.

**D** Display action code.

**last_name**
The last name of the employee entry to be displayed.

**%** The % is an optional parameter used to retrieve information sharing common characteristics, such as similar names. See the examples that follow.

**first_name**
The first name of the employee entry to be displayed.

**Notes:**

1. The / character separates the different parameters.
2. The % can only be used when the **first_name** parameter has been omitted.
3. The [ and ] characters denote an optional parameter.

For example, to display the entry for Takao Chiba, you would type:

```
rc /D CHIBA/TAKAO
```

To display the entries for everyone who has a last name that begins with S, you would type:

```
rc /D S%
```

To display the entries for everyone in the database, you would type:

```
rc /D %
```

```
#include <tpfeq.h>                /* Include libraries               */
#include <tpfapi.h>
#include <tpfarapi.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

/*********************************************************************/
/*                                                                   */
/*  Declare the internal and external functions.                     */
/*                                                                   */
/*********************************************************************/
static void Check();
#pragma map(qxp4_display,"QXP4")
extern void qxp4_display();

EXEC SQL INCLUDE SQLCA;           /* Include the SQLCA               */


  /*******************************************************************/
  /* Check:                                                          */
  /*    This function verifies the SQLSTATE returned.  If it is not 0,*/
  /* a message is printed indicating what the SQLSTATE returned was.  */
  /*                                                                 */
  /* A customer implementing a function like this may want to include */
  /* a much more robust error handling and recovery.                 */
  /*******************************************************************/

static void Check()
{
  if (memcmp(sqlca.sqlstate,"00000",sizeof(sqlca.sqlstate))!= 0) {
    printf("FAILED\n %d %.5s\n",sqlca.sqlcode,sqlca.sqlstate);
    exit(5);
  }
}
```

*Figure 25. TPF Program to Display a Specific Entry in the PHONE_DIRECTORY Table (Part 1 of 7)*

```
 /*********************************************************************/
 /* qxp4_display:                                                     */
 /* This routine will display the data asked for from DB2. It will    */
 /* parse the rest of the message and issue a query depending on      */
 /* what information was passed.                                      */
 /*********************************************************************/
void qxp4_display()
{
  short int num_input;           /* The number of variables sscanf   */
                                 /* has correctly set up.  This is   */
                                 /* used to check the validity of    */
                                 /* the parameters.                  */
  struct mi0mi *blk;             /* Pointer to the input message     */


 /*********************************************************************/
 /* Declare all the variables that SQL needs to know about.           */
 /*********************************************************************/


  EXEC SQL BEGIN DECLARE SECTION;

 /*********************************************************************/
 /*  Set up a structure for the directory record.  This structure     */
 /*  is set up in the same order as the CREATE TABLE parameters       */
 /*  were when the table was created on DB2.                          */
 /*********************************************************************/
    struct {
      char last_name[18];
      char first_name[9];
      char middle_initial[2];
      char country_code[5];
      char area_code[6];
      char phone_number[13];
      short int employee_number;
      char timestamp[27];
    } dir_record;

    char buf[16];                /* This will be used to point to    */
                                  /* the database to connect to.     */
  EXEC SQL END DECLARE SECTION;
```

*Figure 25. TPF Program to Display a Specific Entry in the PHONE_DIRECTORY Table (Part 2 of 7)*

```
/*********************************************************************/
/* There are two possible cursors needed.  The first one is used     */
/* when only the last name has been given.  The second one is        */
/* used when the last name and first name are given.                 */
/*                                                                   */
/*********************************************************************/


/*********************************************************************/
/* Declare a cursor D1 for use when only the last name is given.     */
/*********************************************************************/

  EXEC SQL DECLARE D1 CURSOR FOR
       SELECT LAST_NAME,FIRST_NAME,
        PHONE_NUMBER, EMPLOYEE_NUMBER FROM TPFNET.PHONE_DIRECTORY
        WHERE LAST_NAME LIKE :dir_record.last_name
        ORDER BY LAST_NAME,FIRST_NAME;

/*********************************************************************/
/* Declare a cursor D2 for use when the last name and the first      */
/* name are given.                                                   */
/*********************************************************************/

  EXEC SQL DECLARE D2 CURSOR FOR
       SELECT *
        FROM TPFNET.PHONE_DIRECTORY
        WHERE LAST_NAME = :dir_record.last_name AND
              FIRST_NAME = :dir_record.first_name;


 /*********************************************************************/
 /* Issue the connect with the name of the database to connect        */
 /* to.  When done, the check function will check the return code,    */
 /* and if invalid, exit.  Ebx000 was set up in the root segment with*/
 /* the name of the database to connect to.                           */
 /*********************************************************************/

 strcpy(buf, &ecbptr()->ebx000);
 EXEC SQL
   CONNECT TO :buf;
 Check();


 /*********************************************************************/
 /* Parse the message block.  The first parameter is unused, and      */
 /* the second was already parsed.  If there is a first name, it is   */
 /* separated from the last name by a /.                              */
 /*                                                                   */
 /*********************************************************************/
 blk = ecbptr()->ce1cr0;
 num_input = sscanf(blk->mi0acc,
             "%*s /%*1c %17[^/]/%8s",
                        dir_record.last_name,
                               dir_record.first_name);


 /*********************************************************************/
 /*  Based on the number of successful conversions, we can tell       */
 /*  which query is requested.                                        */
 /*********************************************************************/

 switch (num_input)
 {
```

*Figure 25. TPF Program to Display a Specific Entry in the PHONE_DIRECTORY Table (Part 3 of 7)*

```
case 1:
{
 /*************************************************************/
 /*                                                           */
 /*  Since we are using the LIKE attribute on the WHERE clause, */
 /*  we must append a "%" at the end of the last_name.         */
 /*                                                           */
 /*************************************************************/

 strcpy((strchr (dir_record.last_name,'\0')),"%");

 /*************************************************************/
 /* Only the last name is given.  Give back all the names for   */
 /* all the people with a last name like the one entered.       */
 /* The first thing that needs to be done is the cursor opened. */
 /*************************************************************/
  EXEC SQL OPEN D1;
  Check();

 /*************************************************************/
 /* Now that the cursor is opened, get the first row of data.   */
 /*************************************************************/

  EXEC SQL FETCH D1 INTO
     :dir_record.last_name,
     :dir_record.first_name,
     :dir_record.phone_number,
     :dir_record.employee_number;


 /*************************************************************/
 /* Before we call check, check for end of table.  If it is,    */
 /* there were no entries for this query, so respond back with  */
 /* this information.                                           */
 /*************************************************************/

  if (memcmp(sqlca.sqlstate,"02000",sizeof(sqlca.sqlstate))== 0)
  {
     printf("No entries found for name %s",dir_record.last_name);
  }
  else
  {
```

*Figure 25. TPF Program to Display a Specific Entry in the PHONE_DIRECTORY Table (Part 4 of 7)*

```
    short int count;         /* Count has the number of rows    */
                             /* returned from the query.        */

    count = 0;               /* No rows printed yet.            */
    Check();                 /* Check the results of the first  */
                             /* fetch.                          */

    printf(
    "Last Name         Fst Name Phone        ENum");

    /*************************************************************/
    /* While there is more information, send it all back to the */
    /* person who issued the query.                              */
    /*************************************************************/

    while (memcmp(sqlca.sqlstate,"00000",sizeof(sqlca.sqlstate))
          ==0)
    {
     printf("%s %s %s %hd\n",
             dir_record.last_name,
                dir_record.first_name,
                   dir_record.phone_number,
                      dir_record.employee_number);
    count++;

    /*************************************************************/
    /* Get the next row of data from the query.                  */
    /*************************************************************/

     EXEC SQL FETCH D1 INTO
     :dir_record.last_name,
     :dir_record.first_name,
     :dir_record.phone_number,
     :dir_record.employee_number;

    /*************************************************************/
    /* Again, before we call check, we must see if this is the  */
    /* last record.                                              */
    /*************************************************************/

     if (memcmp(sqlca.sqlstate,"02000",sizeof(sqlca.sqlstate))==0)
     {
       printf("%1.0d rows found.",
               count);
     }
     else
     {
       Check();
     }
   }
  }
  break;
}
```

*Figure 25. TPF Program to Display a Specific Entry in the PHONE_DIRECTORY Table (Part 5 of 7)*

```
case 2:
{
 /*************************************************************/
 /* The last name and the first name were given.   Return     */
 /* the entire record(s) of the person with this name.        */
 /* The first thing that needs to be done is the cursor opened. */
 /*************************************************************/
  EXEC SQL OPEN D2;
  Check();

 /*************************************************************/
 /* Now that the cursor is opened, get the first row of data.   */
 /*************************************************************/

  EXEC SQL FETCH D2 INTO
     :dir_record;

 /*************************************************************/
 /* Before we call check, check for end of table.  If it is,    */
 /* there were no entries for this query, so respond back with  */
 /* this information.                                         */
 /*************************************************************/

  if (memcmp(sqlca.sqlstate,"02000",sizeof(sqlca.sqlstate))== 0)
  {
     printf("No entries found for name %s",dir_record.last_name);
  }
  else
  {
```

*Figure 25. TPF Program to Display a Specific Entry in the PHONE_DIRECTORY Table (Part 6 of 7)*

```
         short int count;         /* Count has the number of rows    */
                                  /* returned from the query.        */

         count = 0;               /* No rows printed yet.            */
         Check();                 /* Check the results of the first  */
                                  /* fetch.                          */

         printf(
"Last Name         Fst Name MI Phone        Country Area  ENum");

         /*************************************************************/
         /* While there is more information, send it all back to the */
         /* person who issued the query.                              */
         /*************************************************************/

         while (memcmp(sqlca.sqlstate,"00000",sizeof(sqlca.sqlstate))
            ==0)
         {
          printf(
          "%s %s %s  %s %s    %s %hd\n",
           dir_record.last_name,
              dir_record.first_name,
                 dir_record.middle_initial,
                     dir_record.phone_number,
                        dir_record.country_code,
                              dir_record.area_code,
                                 dir_record.employee_number);
         count++;

         /*************************************************************/
         /* Get the next row of data from the query.                  */
         /*************************************************************/

          EXEC SQL FETCH D2 INTO
          :dir_record;


         /*************************************************************/
         /* Again, before we call check, we must see if this is the  */
         /* last record.                                              */
         /*************************************************************/

          if (memcmp(sqlca.sqlstate,"02000",sizeof(sqlca.sqlstate))==0)
          {
            printf("%1.0d rows found.",
                    count);
          }
          else
          {
            Check();
          }
        }
       }
       break;
     }
     default:
       printf("Invalid input for display option.  Please re-enter\n");
  }
  exit(0);
}
```

*Figure 25. TPF Program to Display a Specific Entry in the PHONE_DIRECTORY Table (Part 7 of 7)*

# Using Assembler Language with TPFAR

This section contains examples of TPF applications written in assembler language that use TPFAR to offload data from TPF.

## Offloading Data from the TPF System

Another type of application using TPFAR is a back end program to move the data from TPF to DB2. A back end program is most appropriately used when many different ECBs are inserting single rows into a DB2 table that does not need to be current with the TPF database. An example of this type of application is an application server table (AS) containing flight reservations used for statistical purposes only. (The example is shown coded in assembler language).

Instead of issuing SQL CONNECT, INSERT, and COMMIT commands every time a request comes in, the application queues the data in a file-chained area. The back end application then issues the SQL CONNECT command, issues any number of SQL INSERT commands, and then issues an SQL COMMIT command. Because an SQL CONNECT and COMMIT command are not issued for each SQL INSERT command, processing is significantly more efficient.

**Note:** This type of back end processing can be used only when the data on the remote AS does not need to be current with the TPF data.

Another benefit of offloading is that if the connection to the AS is lost, the TPF applications can continue to queue the messages and, when the connection is brought back, the data can be offloaded.

## Setting Up the Application Server

For the assembler examples to work, two tables, INSERT_DRIVER and LOG_DRIVER, must be created on the remote DB2 subsystem. You can do this in the DB2 system through the use of interactive SQL. On MVS, the mechanism used is called SQL Processor Using File Input (SPUFI) application. SPUFI is a way to execute SQL commands from a TSO terminal.See the *DB2 Application Programming and SQL Guide* for more information on SPUFI. Figure 26 on page 80 shows the two SQL CREATE TABLE commands needed to create these tables.

```
CREATE TABLE TPFNET.INSERT_DRIVER
  (IDENTIFIER    CHAR(8),
   TIME_STAMP    TIMESTAMP NOT NULL,
   ECB_ADDR      INTEGER NOT NULL,
   CPU_ID        CHAR(1),
   NUM_LEFT      INTEGER NOT NULL,
   NUM_DONE      INTEGER,
   NUM_INSERT    INTEGER,
   NUM_FINDS     INTEGER,
   NUM_TO_LOG    INTEGER,
   PRIMARY KEY(ECB_ADDR, NUM_LEFT, TIME_STAMP));


CREATE TABLE TPFNET.LOG_DRIVER
  (IDENTIFIER    CHAR(8) NOT NULL,
   TIME_STAMP    TIMESTAMP NOT NULL,
   ECB_ADDR      INTEGER NOT NULL,
   CPU_ID        CHAR(1) NOT NULL,
   NUM_DONE      INTEGER NOT NULL,
   PRIMARY KEY(ECB_ADDR, NUM_DONE, TIME_STAMP));
```

*Figure 26. SQL CREATE TABLE Commands to Create the INSERT_DRIVER and LOG_DRIVER Tables*

These table commands have a primary key. Every primary key requires a primary index. These indexes must be created before the table definitions are complete. A primary key identifies a specific column or columns in a table that uniquely define each row in the table. Each row in the table must have a primary key value that is unique and not null. A primary index on a table allows access to a specific row of data without having to read every row in the table each time.

Figure 27 shows an example of the two SQL CREATE INDEX commands that need to be run for the CREATE TABLE definitions to be complete.

```
CREATE UNIQUE INDEX TPFNET.INSERT_D_INDEX ON TPFNET.INSERT_DRIVER
  (ECB_ADDR ASC,
   NUM_LEFT DESC,
   TIME_STAMP DESC);

CREATE UNIQUE INDEX TPFNET.LOG_D_INDEX ON TPFNET.LOG_DRIVER
  (ECB_ADDR ASC,
   NUM_DONE DESC,
   TIME_STAMP DESC);
```

*Figure 27. Example of SQL CREATE INDEX Commands to Create a Primary Index*

## Assembler Program QXRK

Segment QXRK is shown in Figure 28 on page 82. The beginning of this program is a simple command front end interface. Segment QXRK parses the input message and passes the parameters to the back end program (segment QXRL in Figure 29 on page 88). It also calculates the time it takes the back end program to complete by storing the time of day (TOD) clock before and after the call to the back end program. Before completing, this segment informs you if the function completed successfully, what the invocation parameters were, and how long it took to run. This application is started using a command, which can be ZTEST or any other command that points to this program. An example of a ZTEST command to start this application is:

**ZTEST num_insert num_find num_com [text] [RDB-rdbname]**

where:

**num_insert**
The decimal number of inserts for the application to perform.

**num_find**
The decimal number of finds to do before each insert. This can be used to simulate the average number of finds that are needed. The program repeatedly finds RRT record ordinal number 0.

**num_com**
The decimal number of inserts to do before a log record is inserted and an SQL COMMIT command is issued.

**text**
An 8-byte field used to uniquely define records.

**rbdname**
An 16-byte field that indicates the remote relational database name to connect to.

```
         PRINT NOGEN
*********************************************************************
*         THIS PRODUCT CONTAINS "RESTRICTED MATERIALS OF IBM "
*         COPYRIGHT = 5748-T13 (C) COPYRIGHT IBM CORP 1979,1989
*             LICENSED MATERIAL - PROGRAM PROPERTY OF IBM
*             REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083
*********************************************************************
         BEGIN NAME=QXRK,VERSION=ZZ,IBM=YES       CREATED    05/03/91
*
************************************************************
*                                                          *
* MODULE NAME..... QXRK  (PDS NAME)                        *
* RELATED MODULE.. NONE                                    *
* DOCUMENT NAME... NONE                                    *
* DESCRIPTION..... ZTEST DRIVER TO START TPFAR INSERT PGM   *
* LEVEL........... VERSION 1  MODIFICATION 0               *
*                                                          *
* FUNCTION... THIS SEGMENT JUST CALLS THE TPFAR INSERT DRIVER *
*             WITH THE VALUES PASSED                       *
*                                                          *
* MODULE ATTRIBUTES..                                      *
*   TYPE.......... 'E'                                     *
*                                                          *
* ECB-CONVENTIONS.......NONE                               *
* REGISTER-CONVENTIONS..NONE                               *
*                                                          *
************************************************************
* INTERFACE REQUIREMENTS:                                  *
*                                                          *
* DEPENDENCIES... TPFAR MUST BE GENERATED IN SYSTEM.       *
* RESTRICTIONS... NONE                                     *
*                                                          *
*  ECB          * INPUT..              * OUTPUT..          *
* -------------*----------------------*----------------------*
* WORK AREA.... *                      *                     *
* DATA LEVELS.. * MESSAGE BLOCK        *                     *
* REGISTERS.... *                      *                     *
*                                                          *
************************************************************
*                                                          *
* EXTERNAL-REFERENCES..   NONE                             *
*                                                          *
* ACRONYMS... (& DEFINITIONS)                              *
*                                                          *
* EXIT-NORMAL.. PRINT OUT STATISTICS FOR INSERTS AND EXIT  *
*     -ERROR... PRINT OUT ERROR MESSAGE AND EXIT           *
*                                                          *
*                                                          *
************************************************************
         SPACE 5
************************************************************
*                                                          *
*  Parse the message block looking for the parameters.     *
*                                                          *
*  When finished, we can release the message block.        *
*                                                          *
************************************************************
         SPACE 2
         BPKDC EXECUTE=Y,BPKD=DB2IBPKD,HELP=QXRKHELP
```

*Figure 28. TPF Program to Call the Insert Driver with the Values Passed (Part 1 of 5)*

```
*************************************************************
*                                                           *
* The parameter area needs to be set up for the call to the *
* insert program.  The first parameter is the number of     *
* inserts the program is to do, the second is the number of *
* finds the program is to do between inserts, the third is  *
* the number of inserts to do before inserting a log record *
* and committing, the forth parameter is an optional field  *
* that is placed in the inserted record.  The last parameter*
* is a optional keyword parameter for the RDBNAME to be     *
* accessed.  If not given, it will default to DB23TST.      *
*                                                           *
*************************************************************
         SPACE 2
         DCTBPK REG=R1             Set up format of param list
         L      R4,BPKOPM1         Set up the first parameter
         L      R15,0(,R4)         (Number of inserts)
         ST     R15,EBX000         In EBX000
         L      R4,BPKOPM2         Set up the second parameter
         L      R15,0(,R4)         (Number of finds)
         ST     R15,EBX004         In EBX004
         L      R4,BPKOPM3         Set up the third parameter
         L      R15,0(,R4)         (Number before commit)
         ST     R15,EBX008         In EBX008
         L      R4,BPKOPM4         Set up the forth parameter
*                                  (Informational log)
         MVC    EBX012(8),=C'       ' Initialize field to blanks
         SR     R15,R15            Clear for insert
         ICM    R15,B'0001',0(R4)  Get length of field
         BZ     QXRKCONT           No input, leave blank
         BCTR   R15,0              Minus 1 for MVC
         EX     R15,QXRKMVC        Move the input field
QXRKCONT DS     0H
         L      R4,BPKOPM5         Set up the fifth parameter
*                                  (RDBNAME)
         SR     R15,R15            Clear for insert
         ICM    R15,B'0001',0(R4)  Get length of field
         BZ     QXRKCONT2          No input, use default
         MVC    EBX020(16),=C'               ' Init field to blnks
         BCTR   R15,0              Minus 1 for MVC
         EX     R15,QXRKMVC2       Move the input field
         B      QXRKCONT3          Continue
QXRKCONT2 DS    0H                 Move the default RDBNAME
         MVC    EBX020(16),=C'DB23TST         '
QXRKCONT3 DS    0H
```

*Figure 28. TPF Program to Call the Insert Driver with the Values Passed (Part 2 of 5)*

```
*****************************************************************
*                                                               *
* Issue the message indicating that the inserts have started. *
*                                                               *
*****************************************************************
         SPACE 2
         WTOPC PREFIX=QXRK,TIME=YES,NUM=02,LET=I,CHAIN=NO,ENDOFM=YES,  X
               TEXT='STARTING INSERTS'
*****************************************************************
*                                                               *
* Set up the interface register, R6, and release the block.   *
*                                                               *
*****************************************************************
         SPACE 2
         RELCC D0                  Release the parse block
         LA    R6,EBX000           Interface is in R6
*****************************************************************
*                                                               *
* Store the TOD clock before the call.  On return from the    *
* insert, the clock time will be again stored so that the     *
* time that it took for all the inserts to work can be        *
* calculated.                                                   *
*                                                               *
*****************************************************************
         SPACE 2
         STCK  EBX040
         ENTRC QXRL                CALL THE INSERT
         STCK  EBX048
```

*Figure 28. TPF Program to Call the Insert Driver with the Values Passed (Part 3 of 5)*

```
*************************************************************
*                                                           *
* In order to work with the STCK, we must change the format   *
* of the time to a long.  This is done by inserting a x'4D'   *
* at the start.                                             *
*                                                           *
*************************************************************
         SPACE 2
         MVC   EBX080(15),EBX040    Copy for move
         MVC   EBX041(15),EBX080    Copy back over one space
         MVI   EBX040,X'4D'         Insert a X'4D' in front of
         MVI   EBX048,X'4D'         both numbers
         LD    0,EBX048             Load double the ending time
         SD    0,EBX040             Subtract the starting time
         DD    0,=D'1000'           Divide by 1000 to get
*                                   milliseconds
         AD    0,DUBB               This add will shift the high
*                                   order fullword of a double to
*                                   last four bytes for usage as
*                                   an integer.
         STD   0,EBX040             Store the number.
         L     R3,EBX044            Load the last four bytes.
*                                   This is the number of
*                                   milliseconds that elapsed.
         SR    R2,R2                Clear R2 for divide.
         D     R2,=F'1000'          Divide by 1000 to get num. of
*                                   seconds.
         ST    R3,EBX056            Save number of seconds.
         CVD   R2,EBX080            Convert the remainder to printable
         UNPK  EBX060(4),EBX086(2)  characters.
         MVZ   EBX061(3),EBX060     Set up the zone for the characters.
         MVI   EBX060,C'.'          Add the decimal point.
*
         L     R3,EBX044            Number of milliseconds
         SR    R2,R2                Clear for divide
         D     R2,EBX000            Divide by the number of inserts
         SR    R2,R2                Clear for divide
         D     R2,=F'1000'          Divide to get seconds
         ST    R3,EBX064            Save the number of seconds
         CVD   R2,EBX080            Convert the remainder to printable
         UNPK  EBX068(4),EBX086(2)  characters.
         MVZ   EBX069(3),EBX068     Set up the zone for the characters.
         MVI   EBX068,C'.'          Add the decimal point.
         LTR   R6,R6                If the return was good,
         BZ    QXRKGOOD             issue the good message.
         WTOPC PREFIX=QXRK,TIME=YES,NUM=04,LET=E,CHAIN=NO,ENDOFM=YES,  X
               TEXTA=QXRKERR,SUB=(DECA,EBX056,CHARA,EBX060,            X
               DECA,EBX000,DECA,EBX004,                                X
               DECA,EBX008,CHARA,EBX012,CHARA,EBX020),COMP=YES
         EXITC
QXRKGOOD DS    0H
         WTOPC PREFIX=QXRK,TIME=YES,NUM=03,LET=I,CHAIN=NO,ENDOFM=YES,  X
               TEXTA=QXRKMSG1,SUB=(DECA,EBX056,CHARA,EBX060,           X
               DECA,EBX064,CHARA,EBX068,DECA,EBX000,DECA,EBX004,       X
               DECA,EBX008,CHARA,EBX012,CHARA,EBX020),COMP=YES
         EXITC
QXRKHELP DS    0H
         WTOPC PREFIX=QXRK,TIME=YES,NUM=01,LET=E,CHAIN=NO,ENDOFM=YES,  X
               TEXTA=QXRKMSG
         EXITC
```

*Figure 28. TPF Program to Call the Insert Driver with the Values Passed (Part 4 of 5)*

```
QXRKMSG  DC  AL1(QXRKMSGE-QXRKMSG-1),AL1(#CAR)
         DC  C'ZDB2I INSERTS FINDS LOGGING R-RDBNAME',AL1(#CAR)
         DC  C'  WHERE INSERTS - NUMBER OF INSERTS TO DO',AL1(#CAR)
         DC  C'  FINDS - NUMBER OF FINDS TO DO BEFORE INSERT',AL1(#CAR)
         DC  C'  LOGGING - AMOUNT TO WAIT BEFORE COMMITTING',AL1(#CAR)
         DC  C'  RDBNAME - RDBNAME TO CONNECT TO'
QXRKMSGE EQU  *
QXRKMSG1 DC  AL1(QXRKMSG1E-QXRKMSG1-1)
         DC  C'DONE WITH INSERTS, TIME= .......... ....',AL1(#CAR)
         DC  C'AVERAGE PER INSERT= .......... ....',AL1(#CAR)
         DC  C'NUM INSERTS= ..........',AL1(#CAR)
         DC  C'NUM FINDS= ..........',AL1(#CAR)
         DC  C'LOGGING= ..........',AL1(#CAR)
         DC  C'IDENTIFIER = ........',AL1(#CAR)
         DC  C'RDBNAME= ..........'
QXRKMSG1E EQU  *
QXRKERR  DC  AL1(QXRKERRE-QXRKERR-1)
         DC  C'AN ERROR OCCURRED. '
         DC  C'TIME= .......... ....',AL1(#CAR)
         DC  C'NUM INSERTS= ..........',AL1(#CAR)
         DC  C'NUM FINDS= ..........',AL1(#CAR)
         DC  C'LOGGING= ..........',AL1(#CAR)
         DC  C'IDENTIFIER = ........',AL1(#CAR)
         DC  C'RDBNAME= ..........'
QXRKERRE EQU  *
QXRKMVC  MVC   EBX012(0),1(R4)
QXRKMVC2 MVC   EBX020(0),1(R4)
         DS    0D
DUBB     DC  X'4F08000000000000'
DB2IBPKD BPKDC (PRD,INSERTS,,8),                                    X
               (PRD,FINDS,,8),                                      X
               (PRD,LOGGING,,8),                                    X
               (P,IDENT,,8),                                        X
               (K,RDBNAME,1,16)
         LTORG
         FINIS QXRK
         END
```

*Figure 28. TPF Program to Call the Insert Driver with the Values Passed (Part 5 of 5)*

## Assembler Program QXRL

Segment QXRL does the SQL work of the assembler package. It takes as input the parameters that were parsed in the QXRK and does the requested inserts.

The commit log is used to indicate how far the program has gone, if a problem occurs. By inserting this record into the LOG_DRIVER table, you can be sure that, if the next issued SQL COMMIT command works, all the inserts done up to that point have been placed in the tables. For example, you want to insert 10,000 records into the table. You would issue an SQL COMMIT command at a given interval to prevent the entire table from locking for the whole duration. If, after inserting 5,000 records, a problem occurred on TPF or the remote AS or the link between the two, when the problem was resolved, you would restart the program from that point, rather than from the beginning. By looking at the LOG_DRIVER table, you would be able to tell what record was last committed, and you could then restart your program from this point. Because this program is just an example, the recovery logic is not included but must be considered when writing this type of application.

This program issues a find and wait for the number of times that was asked for in the **num_find** parameter. Because this program is only locating a randomly chosen record (RRT record 0), actual performance can differ.

```
         PRINT NOGEN
*********************************************************************
*         THIS PRODUCT CONTAINS "RESTRICTED MATERIALS OF IBM "
*         COPYRIGHT = 5748-T13 (C) COPYRIGHT IBM CORP 1979,1989
*             LICENSED MATERIAL - PROGRAM PROPERTY OF IBM
*             REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083
*********************************************************************
         BEGIN NAME=QXRL,VERSION=ZZ,IBM=YES
*
************************************************************
*                                                          *
* MODULE NAME..... QXRL                                    *
* RELATED MODULE.. None                                    *
* DOCUMENT NAME... None                                    *
* DESCRIPTION..... Issues inserts into a remote database   *
* LEVEL........... VERSION 1  MODIFICATION 0               *
*                                                          *
* FUNCTION... This driver issues inserts into a remote     *
*             database.  Depending on the input values, this *
*             could be used to simulate a real environment.  *
*                                                          *
* MODULE ATTRIBUTES..                                      *
*   TYPE.......... 'E' (ECB CONTROLLED)                    *
*                                                          *
* ECB-CONVENTIONS.......NONE USED                          *
*                                                          *
************************************************************
* INTERFACE REQUIREMENTS:                                  *
*                                                          *
* DEPENDENCIES... This will only work with the TPFAR function *
* RESTRICTIONS... NONE                                     *
*                                                          *
*  ECB          * INPUT..              * OUTPUT..          *
* ************************************************************
* WORK AREA.... *                      *                   *
* DATA LEVELS.. * D1, D2, and DF free  * D1, D2, and DF free  *
* REGISTERS.... * R6 POINT TO INPUT    * R6 = 0 - GOOD RETURN *
*                 PARAMETER            * R6 != 0 - BAD RETURN *
************************************************************
*                                                          *
* EXTERNAL-REFERENCES..                                    *
*      ROUTINES....  NONE                                  *
*      DATA AREAS..  NONE                                  *
*                                                          *
* ACRONYMS...                                              *
*                                                          *
* EXIT-NORMAL.. R6 = 0                                     *
*     -ERROR... R6 != 0                                    *
*                                                          *
* INTERFACE REGISTER R6 -                                  *
*                                                          *
*    R6 MUST POINT TO A AREA THAT LOOKS LIKE THE FOLLOWING:  *
*                                                          *
*  F    - The number of inserts this call is to do         *
*  F    - The number of find and waits to be issued        *
*         before an insert.                                *
*  F    - The number of inserts to do before a log record is *
*         inserted into the log table and a commit issued  *
*  CL8  - A 8 byte token used to identify different inserts  *
*         into the table.  This is included on both the    *
*         inserts and the log records.                     *
*  CL16 - A 16 byte RDBNAME used to connect to.            *
*                                                          *
************************************************************
```

*Figure 29. TPF Program to Insert Multiple Records into a Table (Part 1 of 8)*

```
*                                                               *
*  Pseudo Code                                                  *
*                                                               *
* 1. Issue the SQL WHENEVER command for SQLERROR and            *
*    SQLWARING to go to the error handler on any bad return.    *
* 2. Set up addressability to the SQL area.                     *
* 3. Set up addressability to the save area and save            *
*    the calling programs R0 through R7.                        *
* 4. Set up addressability to the SQLCA area.                   *
* 5. Save the ECB address and the CPUID in the save area.       *
*      (These are needed in a local variable for SQL)           *
* 6. Issue the SQL CONNECT to the RDB name passed in the        *
*    parameter area.                                            *
* 7. Clear the total number of inserts completed.               *
* 8. Set up the total number of inserts to do from the          *
*    parameter area.                                            *
* 9. While the total number of inserts to do is > 0             *
*    A. Set the number of inserts completed since the last      *
*       commit to 0.                                            *
*    B. While the total number of inserts to do is > 0 and      *
*       (Number of inserts before logging is 0 or               *
*        the number of inserts completed since the last         *
*        commit is < the number of inserts to do before         *
*        logging)                                               *
*       1. Set up the number of find and waits before           *
*          an insert from the parameter area.                   *
*       2. While the number of find and waits before an insert*
*          is > 0                                               *
*          a. Find RRT record 0 on level F                      *
*          b. Release the record on level F                     *
*          c. Decrement the number of find and waits before     *
*             an insert.                                        *
*       3. Issue the SQL INSERT to insert a record into the     *
*          INSERT_DRIVER table.                                 *
*       4. Decrement the total number of inserts to do.         *
*       5. Increment the number of inserts completed since      *
*          the last commit.                                     *
*    C. If the number of inserts before logging is NOT 0        *
*       1. Issue the SQL INSERT to insert a record into the     *
*          LOG_DRIVER table.                                    *
*    D. Issue the SQL COMMIT command to commit all inserts.     *
* 10.Restore the calling programs R6 and R7.                    *
* 11.Set R6 to 0 to indicate that this is a good return.        *
* 12.Release the Blocks used for the SQL area and the           *
*    save/SQLCA area.                                           *
* 13.Return to the calling application.                         *
*                                                               *
* On any error from a SQL call, the following will be done:     *
*                                                               *
* 1. Issue the SQL WHENEVER command for SQLERROR and            *
*    SQLWARING to continue on any bad return.  This will        *
*    stop an infinite loop if the SQL ROLLBACK we are           *
*    about to issue fails.                                      *
* 2. Issue a SQL ROLLBACK command to rollback any work that     *
*    has been completed.                                        *
* 3. Restore the calling programs R6 and R7.                    *
* 4. Set R6 to 4 to indicate that this is a error return.       *
* 5. Release the Blocks used for the SQL area and the           *
*    save/SQLCA area.                                           *
* 6. Return to the calling application.                         *
*                                                               *
*****************************************************************
```

*Figure 29. TPF Program to Insert Multiple Records into a Table (Part 2 of 8)*

```
         EJECT
*************************************************************************
* If any error occurs, go to the bad return routine to issue a        *
* rollback.                                                           *
*************************************************************************
         EXEC SQL WHENEVER SQLERROR GO TO QXRL_BAD
         EXEC SQL WHENEVER SQLWARNING GO TO QXRL_BAD
         SPACE 1
*************************************************************************
* Get and set up a block for the SQL area                             *
*************************************************************************
         GETCC     D1,L4             Get a 4k block on level 1.
         L         R2,CE1CR1         Set up addressability.
         LA        R3,4095           Load the length of the block
         XR        R5,R5             Clear length so the MVCL will
*                                    clear storage.
         MVCL      R2,R4             Clear the work block.
         L         R2,CE1CR1         Re-establish addressability.
         USING     SQLDSECT,R2
         SPACE 1
*************************************************************************
* Get and set up a block for the SQLCA and a save area.  The save     *
* area will be the first 100 bytes.                                   *
*************************************************************************
         GETCC     D2,L4             Get a 4K block on level 2.
         USING     SAVEAREA,R1
         L         R1,CE1CR2         Addressability to the block.
         LA        R3,100(,R1)       Bump past the save area.
         STM       R0,R7,REG_SAVE1   Save the registers
         ST        R9,ECB_ADDR       Put the ECB address in the save
*                                    area.
         MVC       CPU_ID,CE1CPD     Move the CPUID into the save area
         USING     DSQLCA,R3         Access to the SQLCA
         USING     PARAM,R6          Access to the parameter area
         SPACE 1
```

*Figure 29. TPF Program to Insert Multiple Records into a Table (Part 3 of 8)*

```
*********************************************************************
* Connect to the remote database.  The RDBNAME is passed as a       *
* parameter.                                                        *
*********************************************************************
         STM       R0,R7,REG_SAVE    Save registers over SQL call
         EXEC SQL CONNECT TO :RDBNAME
         L         R1,CE1CR2         Reload save area
         LM        R0,R7,REG_SAVE    Reload registers
         SPACE 1
*********************************************************************
* Set up R4 for the total number of inserts completed.              *
*********************************************************************
         SPACE 1
         XC        TOT_COMPLETE,TOT_COMPLETE  None completed so far.
         L         R4,NUM_INSERT     Number of inserts to do
*********************************************************************
* Do while # of inserts is > 0                                      *
*********************************************************************
QXRLWH1  DS    0H
         LTR   R4,R4                Are there more inserts to do?
         BZ    QXRLWH1D             No all finished.
*********************************************************************
* R0 will contain the current number of inserts completed since the  *
* last commit.  So far we haven't done any.                         *
*********************************************************************
         SR        R7,R7             No inserts completed yet.
*********************************************************************
* Do while # of inserts is > 0 and                                  *
*          (Number to log = 0 or # of inserts done < number to log)  *
*********************************************************************
QXRLWH2  DS        0H
         LTR       R4,R4             Number of inserts left to do
         BZ        QXRLWH2D          None left, log if necessary
         OC        NUM_TO_LOG,NUM_TO_LOG  Are we logging?
         BZ        QXRLWH2C          No, keep inserting.
         C         R7,NUM_TO_LOG     Numb. of inserts done since last
*                                    commit < number to log?
         BNL       QXRLWH2D          Yes, log and commit.
QXRLWH2C DS        0H
```

*Figure 29. TPF Program to Insert Multiple Records into a Table (Part 4 of 8)*

```
*************************************************************************
* While # of finds is > 0, issues the find and waits.                   *
*   This code will issue a find and wait for the number of times        *
*   that was passed in the parameter of the RRT ordinal # 0 record.     *
*************************************************************************
         L         R5,NUM_FINDS      Number of finds to do
         LTR       R5,R5             Are there any?
         BZ        QXRLWH3D          No, go do the insert.
         STM       R0,R7,REG_SAVE    Save registers over find code
         CRUSA     S0=F              Insure level F is clear
QXRLWH3  DS        0H
         SR        R0,R0             Set ordinal number to zero
         LA        R6,=CL8'#RRTRI'   Record type of RRT
         LA        R7,CE1FAF         Location for storage of addr.
         ENTRC     FACS              Get file addr of RRT.
         LTR       R0,R0             Has an error been detected?
         BZ        QXRLFERR          Yes, process the error.
         XC        CE1FAF,CE1FAF     Clear RCC and check field
         LH        R2,QXRLRRTI       Set up record id for find
         STH       R2,CE1FAF         in CE1FAF.
         FINWC     DF,QXRLFERR       Initiate record retrieval
*                                    Handle error if it occurs
         RELCC     DF                Just wanted to do the find
*                                    so we can release the block.
         BCT       R5,QXRLWH3        If there are more finds to do
*                                    then continue.
         L         R1,CE1CR2         Restore base of save area.
         LM        R0,R7,REG_SAVE    Reload registers.
```

*Figure 29. TPF Program to Insert Multiple Records into a Table (Part 5 of 8)*

```
*********************************************************************
* Save the number left and the number done into the host variables    *
* for the SQL INSERT command, and then issue the command.             *
*********************************************************************
QXRLWH3D DS        0H
         ST        R4,NUM_LEFT       The number left and the number
         ST        R7,NUM_DONE       done must be saved in host
*                                    variables for use by SQL.
         STM       R0,R7,REG_SAVE    Save regs over SQL call.
         EXEC SQL INSERT INTO INSERT_DRIVER                          X
                (IDENTIFIER,TIME_STAMP,ECB_ADDR,CPU_ID,              X
                 NUM_LEFT,NUM_DONE,NUM_INSERT,NUM_FINDS,NUM_TO_LOG)  X
                  VALUES                                             X
                (:IDENTIFIER,CURRENT TIMESTAMP,:ECB_ADDR,:CPU_ID,    X
                 :NUM_LEFT,:NUM_DONE,:NUM_INSERT,:NUM_FINDS,         X
                 :NUM_TO_LOG)
         L         R1,CE1CR2         Reload base of save area.
         LM        R0,R7,REG_SAVE    Restore regs.
*********************************************************************
* Decrement the number of inserts left to do.                       *
* Remember that the SQL WHENEVER statements at the top will cause   *
* Control to be switched to the error routine whenever an error occurs*
*********************************************************************
         BCTR      R4,0              Another insert completed
*                                    successfully.
*********************************************************************
* Increment the number of inserts that have completed               *
*********************************************************************
         LA        R7,1(,R7)         Increment # done
*********************************************************************
* Increment the total number of inserts that have completed         *
*********************************************************************
         L         R5,TOT_COMPLETE   Total number of inserts completed.
         LA        R5,1(,R5)         Add one.
         ST        R5,TOT_COMPLETE   Save this value.
         B         QXRLWH2           GO AND CHECK FOR MORE INSERTS.
QXRLWH2D DS        0H
```

*Figure 29. TPF Program to Insert Multiple Records into a Table (Part 6 of 8)*

```
*********************************************************************
* We have left the inner while loop.  This means that it is either  *
* time to log or we are at the end of our inserts.  We must now check *
* to see if logging is to occur.                                    *
*********************************************************************
         OC        NUM_TO_LOG,NUM_TO_LOG  Are we logging?
         BZ        QXLRIFN           No, go commit.
         STM       R0,R7,REG_SAVE    Save regs over SQL call.
         EXEC SQL INSERT INTO LOG_DRIVER                              X
               (IDENTIFIER,TIME_STAMP,ECB_ADDR,CPU_ID,NUM_DONE) VALUES X
               (:IDENTIFIER,CURRENT TIMESTAMP,:ECB_ADDR,:CPU_ID,      X
               :TOT_COMPLETE)
         L         R1,CE1CR2         Restore save area base.
         LM        R0,R7,REG_SAVE    Restore registers.
QXLRIFN  DS        0H
*********************************************************************
* Commit the work completed                                         *
*********************************************************************
         STM       R0,R7,REG_SAVE    Save regs over SQL call.
         EXEC SQL COMMIT
         L         R1,CE1CR2         Restore save area base.
         LM        R0,R7,REG_SAVE    Restore registers.
         B         QXRLWH1           Go back and check for more to do.
*********************************************************************
*  We have now completed the number of inserts asked.  Restore the  *
*  callers registers, setting R6 to 0 to indicate that this         *
*  is a good return.                                                *
*********************************************************************
QXRLWH1D DS        0H
         LM        R0,R7,REG_SAVE1   Restore callers registers.
         LA        R6,0              Indicate good return.
QXRLRET  DS        0H
         RELCC     D1                Release used blocks on levels
         RELCC     D2                D1 and D2.
         BACKC
         EJECT
```

*Figure 29. TPF Program to Insert Multiple Records into a Table (Part 7 of 8)*

```
*********************************************************************
* An error was returned on a SQL command.  Whenever this occurs,     *
* we want to rollback the work completed so far.  Before doing this, *
* we have to make sure that if another error occurs on the rollback  *
* we have set the SQL WHENEVER to continue so we do not get into an  *
* infinite loop.                                                     *
* On entry to this loop, the registers need to be restored from the  *
* save area, as they were saved there before the SQL call.           *
*********************************************************************
QXRLFERR DS        0H
         L         R1,CE1CR2         Restore save area base.
         LM        R0,R7,REG_SAVE    Restore registers.
QXRL_BAD DS        0H
         EXEC SQL WHENEVER SQLERROR CONTINUE
         EXEC SQL WHENEVER SQLWARNING CONTINUE
         EXEC SQL ROLLBACK
         L         R1,CE1CR2        Restore save area base.
         LM        R0,R7,REG_SAVE1  Restore callers registers.
         LA        R6,4             Indicate error.
         B         QXRLRET          Return to caller.
         EJECT
*********************************************************************
* The SQLCA dsect must be included in all assembler programs with SQL*
* The actual storage is on data level 1.  This is used just to       *
* include the dsect.                                                 *
*********************************************************************
DSQLCA   DSECT
         EXEC SQL INCLUDE SQLCA
$IS$     CSECT
         SPACE 5
*********************************************************************
* The PARAM dsect maps to the parameter area passed by the calling   *
* application.                                                       *
*********************************************************************
PARAM    DSECT
NUM_INSERT    DS   F               Number of inserts to do.
NUM_FINDS     DS   F               Number of finds before insert.
NUM_TO_LOG    DS   F               Number of inserts before logging.
IDENTIFIER    DS   CL8             Identifier inserted into rows.
RDBNAME       DS   CL16            RDBNAME to connect to.
$IS$     CSECT
         SPACE 2
*********************************************************************
* The SAVEAREA dsect maps to the save area on level 2.               *
*********************************************************************
SAVEAREA DSECT
REG_SAVE1     DS   8F              Save area for users R0 through R7.
REG_SAVE      DS   8F              Reg save area over SQL calls.
NUM_LEFT      DS   F               # of inserts left to do.
NUM_DONE      DS   F               # of inserts completed since last
*                                  commit.
ECB_ADDR      DS   F               Address of ECB (used for insert)
CPU_ID        DS   CL1             CPUID (Used for inserts)
TOT_COMPLETE  DS   F               Total number inserts completed.
$IS$     CSECT
         SPACE 5
QXRLRRTI      DC   CL2'S2'         Record ID of RRT for finds.
         LTORG ,
         FINIS ,
         END   ,
```

*Figure 29. TPF Program to Insert Multiple Records into a Table (Part 8 of 8)*

# Performance and Tuning for TPFAR

This section discusses performance and tuning considerations that are specific to the TPFAR environment. For related information on DB2 performance, see the *DB2 Administration Guide*.

## Considerations That Are the Same

TPFAR is subject to the same performance and tuning considerations that apply to an application running on the same platform as the relational database manager. Therefore, performance studies must begin with the expected performance of the relational database that serves as a target for the request: the application server (AS). The considerations involved include locking, I/O, CPU speed, and type of command. If the application would not perform well as an host application on the remote AS, it will not perform well as a remote application on TPFAR.

## Considerations That Are Different

### Communications Overhead and Hotcons

In standard processing, when a transaction requires access to the application server (AS), an LU 6.2 conversation is allocated between the application requester (AR) and the AS. When this transaction ends, the conversation is deallocated. If the duration of the transaction request is short, for example, a single SQL INSERT command, setting up and tearing down conversations can cause a significant amount of overhead. Setting up and tearing down conversations can account for over half of the data flow between the AR and AS.

The *hotcon* method reduces this overhead. If hotcons are defined for a particular AS, conversations or connections to that AS are no longer deallocated when the transactions are over. Instead, they are saved in hotcon table entries and attached to the appropriate SQL DBMS directory (SDD) entry. The next request directed to that AS automatically gets a ready-to-use conversation or connection bypassing the overhead of allocating a new conversation or connection.

The use of hotcons is not without cost; namely, there is a loss of diagnostic information at the AS, as well as the holding of resources on TPF, VTAM, and DB2 systems for each conversation in the hotcon table. Because the logical unit of work identifier (LUWID) is associated with each conversation, one LUWID is now shared by many transactions. To DB2, it appears as a long running thread. The LUWID consists of a fully qualified LU name, a logical unit of work instance number, and logical unit of work sequence number that uniquely identifies a logical unit of work within the network.

Performance trace information cannot be gathered for a single transaction while hotcons are in use. If information needs to be gathered at the AS for a TPF application, hotcons should be turned off. See *TPF ACF/SNA Network Generation* for information about using hotcons.

When using TCP/IP, the hotcon method similarly reduces overhead. If hotcons are defined for a particular AS, connections to the application server are no longer closed when the transactions are over. Instead, they are saved in hotcon table (HCT) entries and attached to the appropriate SQL DBMS (SDD) entry. The next

request directed to that AS automatically gets a ready-to-use connection, bypassing the overhead of starting a new TCP/IP connection.

## Application Overhead

The design of the application programs themselves can reduce the amount of communications traffic and database activity. Each application instance, or entry control block (ECB), that issues a remote SQL INSERT command requires a subsequent SQL COMMIT command. An installation can batch the SQL INSERT commands through an intermediate queue and then issue the SQL INSERT commands from a single ECB followed by a single SQL COMMIT command. This method of implementing an application as described in “Offloading Data from the TPF System” on page 79, may cut in half the time it would have taken if each SQL INSERT command was followed by an SQL COMMIT command.

**Note:** This method should not be used for applications requiring the data be current with other database updates.

# CTC Considerations for LU 6.2

The communications line throughput needs to be monitored. As a rule of thumb, LU 6.2 running on an SNA CTC link can probably handle the throughput requirements of the TPFAR connected to DB2, but application profiles vary widely. The number of the CTC read buffers defined in keypoint 2 through the use of the SNAKEY macro is one place to check on TPF if the amount of data returned on queries is large. Increasing the number of read buffers may cause better throughput if the bottleneck is on the return side of the CTC link from VTAM. A performance study of each application or group of applications should be done to determine the impact to system resources. You have to consider both sides of the interface. For ALS connections, you can increase the number of read buffers by using the MAXBFRU parameter on the SNAKEY macro.

For detailed information about the SNAKEY considerations, see *TPF ACF/SNA Network Generation*.

# Specific Performance Considerations

## Methods of Calculating Response Time

This section describes methods for calculating TPFAR response times to SQL requests.

### ZSTTD and the SQL Trace Table

Using ZSTTD, you can display the SQL trace table, which contains the response time for SQL requests. At the start of each SQL request, TPFAR saves the original clock value. When the return is made to the application, TPFAR saves the current clock value and subtracts this from the original clock value. The difference, which represents the response time, is stored in the SQL trace table. See *TPF Operations* for more information about ZSTTD.

The SQL trace table is processed in wrap-around mode. Only the most recent SQL commands are saved in the table and can be examined via ZSTTD. In order to save information from older SQL commands, the user exit in segment UAR1 can be used to process the information in the SQL trace table before it is overwritten.

### Data Reduction Reports

A data reduction report, called a system summary report, is calculated for TPFAR. The system summary report has 2 fields:

- The `SQL REQUESTS PER SECOND` field calculates the total number of SQL requests.
- The `ACTIVE SQL ECBS` field counts the number of ECBs that are making SQL requests at a given time.

Message stream data is available for data reduction for the TPFAR system by using the SNA DDM option. The STREAM DDM report shows the DDM message traffic, including the existence time. The existence time is the time difference between the SQL request being sent out from TPF to DB2 and the response coming back into TPF from DB2. By using the existence time and the ZSTTD information, you can determine the SQL request time within TPFAR and TPF/APPC code.

## Segment Allocation

Because most TPFAR segments are one-time calls and returns, it is recommended that the segments be core-resident to reduce the enter/back overhead. This greatly improves TPFAR performance.

## TPF Utilization Impact

The following list contains items in TPF that can affect TPF processor utilization when using TPFAR. All users need to evaluate their own requirements for utilization and TPFAR in order to adjust their system accordingly.

1. The format of the data type.

   When the table is created on DB2, information about the column types is supplied. When a user program asks for the data back in a different form (for instance the table column was created as a floating point number, but the application wants the data in an integer format), this causes additional TPF overhead to convert the data to the requested type.

2. The number of affected columns.

   The larger the number of columns returned on an SQL SELECT or the larger the number of columns used on an SQL INSERT, the more the processing time is affected. The affected columns can have a large impact on processing time, especially on SQL SELECTS. With SQL, you can specify the specific columns that you want returned to your application. An application program that issues an SQL SELECT should only request the data that is required for the application. This way, the amount of processing to return a row to the application can be reduced.

3. The cumulative size of host variables.

   TPF/APPC limits the size of an RU. This limit affects the number of host variables that can fit in a single RU. See “Request Unit Size Considerations” on page 28 for more on the effect this limit has on the system. TCP/IP similarly limits the size of a send message.

4. The select conditions specified.

   When a large number of host variables are used in the condition statement of an SQL command, these variables need to be transmitted to DB2. The more host variables there are on a condition statement, the larger the utilization impact on TPF.

5. The SNA pacing for LU 6.2.

   When sending a large amount of SQL traffic over SNA, SNA pacing limits need to be evaluated. If the pacing value is too low, the processing of a pacing

request or response slows system performance. SNA pacing includes session as well as virtual route (with PU 5 support) pacing.

6. The SQL command mix.

   Of all the SQL commands, the cursor-related, blocked-SQL fetch especially affects system performance. When using the blocked-SQL fetch, multiple rows of the answer set are sent back to TPF from DB2. When the application requests the next row, TPF can use the row information already returned by DB2, rather then having to go back to DB2.

7. Run-time binding versus static binding.

   Run-time binding can be a convenient method for managing DBRMs across multiple platforms. Run-time binding incurs the overhead of the time required to perform the bind. This overhead depends on the system load and configuration. Once a DBRM is bound for a given program on a DB2 system, there is no performance difference between run-time binding and static binding.

# SQL Commands Supported by TPFAR

This table lists all of SQL commands and whether they are supported by TPFAR. See the *DB2 SQL Reference* for more information on the use of each command.

*Table 3. TPFAR SQL Command Subset*

| SQL Command | Supported | Comments |
|---|---|---|
| ALTER DATABASE | Yes | |
| ALTER INDEX | Yes | |
| ALTER STOGROUP | Yes | |
| ALTER TABLE | Yes | |
| ALTER TABLESPACE | Yes | |
| BEGIN DECLARE SECTION | Yes | |
| CLOSE | Yes | |
| COMMENT ON | Yes | |
| COMMIT | Yes | |
| CONNECT | Yes | TPF supports only the TO option because it does not have a default database manager. |
| CREATE ALIAS | Yes | |
| CREATE DATABASE | Yes | |
| CREATE INDEX | Yes | |
| CREATE STOGROUP | Yes | |
| CREATE SYNONYM | Yes | |
| CREATE TABLE | Yes | |
| CREATE TABLESPACE | Yes | |
| CREATE VIEW | Yes | |
| DECLARE CURSOR | Yes | |
| DECLARE STATEMENT | Yes | |
| DECLARE TABLE | Yes | |
| DELETE | Yes | |
| DESCRIBE | Yes | |
| DROP | Yes | |
| END DECLARE SECTION | Yes | |
| EXECUTE | Yes | |
| EXECUTE IMMEDIATE | Yes | |
| EXPLAIN | Yes | |
| FETCH | Yes | |
| GRANT | Yes | |
| INCLUDE | Yes | |
| INSERT | Yes | |
| LABEL ON | Yes | |
| LOCK TABLE | Yes | |

*Table 3. TPFAR SQL Command Subset (continued)*

| SQL Command | Supported | Comments |
|---|---|---|
| OPEN | Yes | |
| PREPARE | Yes | |
| REVOKE | Yes | |
| ROLLBACK | Yes | |
| SELECT INTO | Yes | |
| SET CURRENT PACKAGESET | No | CURRENT PACKAGESET register is not supported. |
| SET CURRENT SQLID | No | TPF uses a fixed SQLID. |
| SET HOST VAR | No | TPF does not support special registers. |
| UPDATE | Yes | |
| WHENEVER | Yes | |

# Appendix. TPFAR SQLCODEs

This appendix lists all SQLCODEs that are set by TPF Application Requester (TPFAR), as well as their corresponding SQLSTATEs. For SQLCODEs set by the application server (AS), for example DB2, see the messages and codes manual for that particular AS. The originator of the SQLCODE (the product identifier) is indicated in the SQLERRP field of the Structured Query Language communications area (SQLCA).

SQLCODE is a signed integer value representing the disposition of the executed SQL statement. Normally, SQLSTATE should be used to check the execution of an SQL statement. The application program checks for different classes of errors by examining the first character of the SQLSTATE. Because the SQLSTATE is product independent, the explanations are common for all platforms. The SQLCODEs are product specific and generally provide more detailed information than SQLSTATE.

When more than one SQLSTATE is listed for a particular SQLCODE, examine the SQLSTATE field in the SQLCA to identify the specific reason for the SQLCODE.

When the system action indicates that the program has been put in a connectable state because of a system error, the LU 6.2 conversation to the AS has been deallocated and an implied rollback has been done on the current unit of work.

**0**

**SQLSTATE:** 00000

**Explanation:** The last SQL statement executed without any errors.

**System Action:** Processing continues.

**System Programmer Response:** None.

**SQLSTATE:** 01501

**Explanation:** The value of a string was truncated when assigned to a host variable. The column size on the AS may have changed since the application was written and the size of the host variable may need to be increased.

**System Action:** SQLWARN1 in the SQLCA is set. The truncated value was placed in the host variable.

**System Programmer Response:** Check the definitions on the AS with the host variable length.

**SQLSTATE:** 01503

**Explanation:** The number of result columns is greater than the number of host variables provided. Columns may have been added to the table on the AS since the application was written.

**System Action:** SQLWARN3 in the SQLCA is set. The columns that had host variables defined are returned.

**System Programmer Response:** Ensure that the application is still compatible with the table on the AS.

**+100**

**SQLSTATE:** 02000

**Explanation:** No data. The statement was executed but no data was found.

**System Action:** No data is returned.

**System Programmer Response:** None.

**+180**

**SQLSTATE:** 01534

**Explanation:** The string representation of a datetime value returned by the AS is invalid. The application had a host variable defined as a time, timestamp, or date, but the data returned by the AS was the wrong size to fit in this type of host variable.

**System Action:** The host variable is set to null.

**System Programmer Response:** Check the definitions in the application and on the AS to correlate the data types.

**+183**

**SQLSTATE:** 01535

**Explanation:** The string representation set up as a host variable is too short to hold the datetime value returned by the AS. The AS returned a variable defined as a time, timestamp, or date, but the host variable defined in the application was too small to fit the data in.

**System Action:** The host variable is set to null.

**System Programmer Response:** Check the definitions in the application and on the AS to correlate the data types.

**+802**

**SQLSTATE:** 01519

**Explanation:** The numeric value received from the AS was not in the valid range.

**System Action:** The host variable is set to null.

**System Programmer Response:** Check the AS to see the value sent and verify its size.

**+863**

**SQLSTATE:** 01539

**Explanation:** In response to TPFAR's CONNECT request, the AS sent coded character set identifiers (CCSIDs) for double or mixed-byte character sets. Because TPFAR does not support double- or mixed-byte character sets, a warning is given. If actual double or mixed-byte data is received in subsequent processing, a dump is taken.

**System Action:** The connection is established.

**System Programmer Response:** Verify that the AS should return double or mixed-byte character sets.

**-180**

**SQLSTATE:** 22007

**Explanation:** The string representation of a datetime value returned by the AS is invalid. The application had a host variable defined as a time, timestamp, or date, but the data returned by the AS was the wrong size to fit in this type of host variable.

**System Action:** The host variable is not set.

**System Programmer Response:** Check the definitions in the application and on the AS to correlate the data types.

**-183**

**SQLSTATE:** 22008

**Explanation:** The string representation set up as a host variable is too short to hold the datetime value returned by the AS. The AS returned a variable defined as a time, timestamp, or date, but the host variable defined in the application was too small to fit the data in.

**System Action:** The host variable is not set up.

**System Programmer Response:** Check the definitions in the application and on the AS to correlate the data types.

**-302**

**SQLSTATE:** 22510

**Explanation:** A NULL terminated input host variable did not contain a NULL. This error probably occurred because a host variable did not contain a NULL character within the first table column size number of characters of the host variable.

**System Action:** The statement cannot be executed. A dump is taken. The program state is not changed.

**System Programmer Response:** Investigate why there is no NULL character in the host variable.

**-303**

**SQLSTATE:** 22509

**Explanation:** A value could not be assigned to a host variable because the data types are incompatible.

**System Action:** The value is set to null, if it is a nullable value.

**System Programmer Response:** Ensure that the host data type and the data type on the AS are compatible.

**-305**

**SQLSTATE:** 22002

**Explanation:** A null value could not be assigned to a host variable because no indicator variable was specified.

**System Action:** The variable is not set up.

**System Programmer Response:** Set up an indicator variable for the variable in question, or change the definition on the AS to NOT NULL.

**-332**

**SQLSTATE:** 57017

**Explanation:** Character translation is not defined. The data returned by the AS is of a different code character set identifier (CCSID) than specified in the SQL Database Management System Directory (SDD). TPFAR does not support character translation.

**System Action:** The statement cannot be executed. A dump is taken. If the failing statement was a CONNECT, the program is left in an unconnected but connectable state, otherwise the program remains in a connected state.

**System Programmer Response:** Determine why the data is a different CCSID. If all data from this AS is the same single-byte CCSID, and the application is capable of handling this CCSID, then update the SDD entry for this AS.

**-501**

**SQLSTATE:** 24501

**Explanation:** The cursor specified by the last SQL statement is not open. COMMIT closes all cursors that did not specify the HOLD option and ROLLBACK closes all cursors. Error conditions on prior SQL statements may also close cursors.

**System Action:** Statement cannot be executed.

**System Programmer Response:** Determine why the cursor is not open.

**-502**

**SQLSTATE:** 24502

**Explanation:** The cursor specified by the OPEN statement is already open.

**System Action:** Statement cannot be executed. The cursor state is unchanged.

**System Programmer Response:** Determine why the application attempted to open a cursor that is already open.

**-522**

**SQLSTATE:** 54014

**Explanation:** An OPEN statement was executed when the maximum number of concurrent cursors have already been opened by this ECB.

**System Action:** The cursor was not opened.

**System Programmer Response:** Examine the application to determine why multiple cursors are open.

**-752**

**SQLSTATE:** 51011

**Explanation:** A CONNECT statement was executed while the ECB is not in a connectable state. The ECB is connectable if no prior SQL statements have been executed or if the last SQL statement was a CONNECT, COMMIT, or ROLLBACK. All other SQL statements take the ECB out of a connectable state, even if the statement fails.

**System Action:** Statement cannot be executed. The ECB remains not connectable.

**System Programmer Response:** Determine by the ECB is not in a connectable state.

**-802**

**SQLSTATE:** 22003

**Explanation:** The numeric value received from the AS was out of the supported range.

**System Action:** The host variable was not set up.

**System Programmer Response:** Determine why the mismatch occurred. The program may be out of date.

**-809**

**SQLSTATE:** 54018

**Explanation:** The data length would exceed the maximum block size. The data cannot be sent.

**System Action:** The statement cannot be executed. A dump is taken. The program state is not changed.

**System Programmer Response:** Try to reduce the number of columns or the size of the fields. Use multiple statements to send the data.

**-901**

**SQLSTATE:** 58004

**Explanation:** An internal error occurred.

**System Action:** The statement cannot be executed. A dump is taken. The program is left in a connected state.

**System Programmer Response:** Check the dump for the exact cause of the error.

**-902**

**SQLSTATE:** 58005

**Explanation:** An internal error occurred.

**System Action:** The statement cannot be executed. A dump is taken. The program is left in an unconnected but connectable state.

**System Programmer Response:** Check the dump for the exact cause of the error. Check the primary and secondary TPF/APPC return codes if a communications error occurred. (See TPF General Macros for more information about return codes for the TPPCC macro.)

**-922**

**SQLSTATE:** 42505

**Explanation:** The remote AS did not have the communication database set up to be accessed by TPFAR.

**System Action:** The statement cannot be executed. A dump is taken. The program is left in an unconnected but connectable state.

**System Programmer Response:** Check the SYSIBM.SYSLUNAMES and SYSIBM.SYSUSERNAMES table on the remote AS for the definition needed for TPFAR. See the TPF Application Requester User's Guide for information about definitions needed.

**-923**

**SQLSTATE:** 57015

**Explanation:** The remote AS did not like a parameter in the TPPCC ALLOCATE issued by TPFAR.

**System Action:** The statement cannot be executed. A dump is taken. The program is left in an unconnected but connectable state.

**System Programmer Response:** Check the secondary return code in the dump for the specific reason for the problem. (See TPF General Macros for more information about return codes for the TPPCC macro .)

**-949**

**SQLSTATE:** 58024

**Explanation:** A TPF/APPC error with an unknown primary or secondary return code was received.

**System Action:** The statement cannot be executed. A dump is taken. The program is left in an unconnected but connectable state.

**System Programmer Response:** Check the dump for the exact cause of the error. Check the primary and secondary TPF/APPC return codes. (See TPF General Macros for more information about return codes for the TPF/APPC macros.)

**-951**

**SQLSTATE:** 58024

**Explanation:** The conversation has failed because of a LU 6.2 protocol error.

**System Action:** The statement cannot be executed. The program is left in an unconnected but connectable state.

**System Programmer Response:** Check for a LU 6.2 dump relating to the protocol error.

**-953**

**SQLSTATE:** 58024

**Explanation:** A timeout or system error occurred on the LU6.2 session.

**System Action:** The statement cannot be executed. The program is left in an unconnected but connectable state.

**System Programmer Response:** Check the cause of the LU 6.2 timeout.

**-1013**

**SQLSTATE:** 52005

**Explanation:** The server name is undefined. The specified RDB name could not be found in the SDD. An AS must be defined in the SDD before it can be accessed.

**System Action:** The CONNECT cannot be executed. The program is left in an unconnected but connectable state.

**System Programmer Response:** Determine why the AS is not in the SDD. The SDD is processor and subsystem unique. If the specified relational database (RDB) is valid, add it to the SDD of the appropriate processor and subsystem.

**-1024**

**SQLSTATE:** 51007

**Explanation:** A CONNECT must be the first SQL statement to be executed. TPFAR does not support a default connection or an implied CONNECT.

**System Action:** The statement cannot be executed. The program is left in an unconnected but connectable state.

**System Programmer Response:** Determine why a CONNECT was not issued previously.

**-30000**

**SQLSTATE:** 58008

**Explanation:** A distributed protocol error occurred.

**System Action:** The statement cannot be executed. A dump is taken. The program remains in a connected state.

**System Programmer Response:** Check the dump for the specific DDM command returned and any associated information to determine the cause of the error.

**-30020**

**SQLSTATE:** 58009

**Explanation:** A distributed protocol error occurred.

**System Action:** The statement cannot be executed. A dump is taken. The program is left in an unconnected but connectable state.

**System Programmer Response:** Check the dump for the specific DDM command returned and any associated information to determine the cause of the error.

**-30021**

**SQLSTATE:** 58010

**Explanation:** Manager levels between the AS and TPFAR are not compatible.

**System Action:** The statement cannot be executed. A dump is taken. The program is left in an unconnected but connectable state.

**System Programmer Response:** Check the dump for the manager levels of the AS. Update the manager levels on the AS to a level supported by TPFAR.

**-30024**

**SQLSTATE:** 58009

**Explanation:** An SQL communications area reply data (SQLCARD) was expected, but did not arrive.

**System Action:** The statement cannot be executed. A dump is taken. The program is left in an unconnected but connectable state.

**System Programmer Response:** Check the dump to find out why the SQLCARD was not found.

**-30025**

**SQLSTATE:** 58009

**Explanation:** A premature end of the reply assembly block (RAB) was encountered.

**System Action:** The statement cannot be executed. A dump is taken. The program is left in an unconnected but connectable state.

**System Programmer Response:** Check the dump to see what replies were missing from the RAB by comparing the RAB with the command assembly block (CAB).

**-30035**

**SQLSTATE:** 58009

**Explanation:** A request correlation ID (RQSCRR) was missing, out of order, or invalid.

**System Action:** The statement cannot be executed. A dump is taken. The program is left in an unconnected but connectable state.

**System Programmer Response:** Check the dump to find out what was wrong with the RQSCRR in error.

**-30040**

**SQLSTATE:** 57012

**Explanation:** The last SQL statement failed because of insufficient nondatabase resources. This does not affect the successful execution of subsequent SQL statements. TPFAR has attempted to issue a TPPCC ALLOCATE 5 times, but has received a retry indication every time. The error probably occurred because all of the TPF/APPC sessions with the remote AS are in use.

**System Action:** The statement cannot be executed. The program is left in an unconnected but connectable state.

**System Programmer Response:** Establish sufficient LU 6.2 sessions to the remote AS to handle the maximum number of concurrent requests to the AS.

**-30041**

**SQLSTATE:** 57013

**Explanation:** The last SQL statement failed because of insufficient nondatabase resources. This affects the successful execution of subsequent SQL statements. TPFAR has received a no-retry return code while trying to issue a TPPCC ALLOCATE. The error probably occurred because there were no LU 6.2 sessions active to the remote AS LU defined in the SDD.

**System Action:** The statement cannot be executed. The program is left in an unconnected but connectable state.

**System Programmer Response:** Start the LU 6.2 sessions with the remote AS before retrying.

**-30042**

**SQLSTATE:** 57012

**Explanation:** TPFAR issued a TPPCC ALLOCATE to begin a session, but the AS has rejected the request because of lack of resources.

**System Action:** The statement cannot be executed. The program is left in an unconnected but connectable state.

**System Programmer Response:** Investigate the lack of resources on the remote side.

**-30043**

**SQLSTATE:** 57013

**Explanation:** A permanent lack of resources has been detected on the remote AS side. No other SQL commands will work.

**System Action:** The statement cannot be executed. The program is left in an unconnected but connectable state.

**System Programmer Response:** Investigate the lack of resources on the remote side.

**-30044**

**SQLSTATE:** 57013

**Explanation:** The TPPCC ALLOCATE was issued with an invalid LU 6.2 mode name. This error probably occurred because of a mismatch between the LU 6.2 mode name defined in the SDD and the active available mode names, previously set up with the ZNCNS command.

**System Action:** The statement cannot be executed. The program is left in an unconnected but connectable state.

**System Programmer Response:** Investigate why there is a difference between the mode name defined in the SDD with the ZSQLD command and the one defined using the ZNCNS command.

**-30045**

**SQLSTATE:** 22003

**Explanation:** TPFAR received an RAB size from the application server (AS) that was not valid. The value is greater than the maximum socket read buffer specified in TPFAR or the value equals zero.

**System Action:** The statement cannot be executed. A dump is taken. The program is left in a connected state.

**System Programmer Response:** Check the dump to determine the rejected value.

**-30046**

**SQLSTATE:** 57013

**Explanation:** An SQL connect failed because of a function that returned an error, preventing subsequent SQL statements from being completed successfully. One of the following occurred:

- TPFAR was unable to acquire a TCP/IP socket. The `socket` function returned an error.
- TPFAR was unable to resolve a host name to an Internet Protocol (IP) address. The `gethostbyname` function returned an error.
- TPFAR was unable to connect to the application server (AS). The `connect` function returned an error.

**System Action:** The statement cannot be executed. A dump is taken. The program is left in a unconnected but connectable state.

**System Programmer Response:** Determine which function caused the error and correct the problem.

**-30047**

**SQLSTATE:** 57012

**Explanation:** TPFAR issued a TCP/IP `socket` function that reurned an error.

**System Action:** The statement cannot be executed. A dump is taken with the specific function and `errno` value. The program is left in a connected state.

**System Programmer Response:** Check the `errno` value returned in the dump to determine why the function failed.

**-30060**

**SQLSTATE:** 42507

**Explanation:** An AS reply message indicated that TPFAR is not authorized for the specified RDB. Because TPF uses the complex name as the TPF application's AUTHID, authorize this complex name in CTKI for the commands that TPF is to issue.

**System Action:** The statement cannot be executed. The program is left in an unconnected but connectable state.

**System Programmer Response:** Grant TPFAR access from DB2.

**-30061**

**SQLSTATE:** 52017

**Explanation:** An AS reply message indicated that the RDB name sent by TPFAR does not exist. The RDB name at the AS needs to match the name specified in the SDD.

**System Action:** The statement cannot be executed. The program is left in an unconnected but connectable state.

**System Programmer Response:** Update the SDD entry to reflect the correct AS RDB name.

**-30070**

**SQLSTATE:** 58014

**Explanation:** Either the AS or TPFAR detected an unsupported SQL command.

**System Action:** The statement cannot be executed. A dump is taken. The program is left in a connected state.

**System Programmer Response:** Examine the dump to determine the rejected command. Also, check the service level of the AS and TPFAR to ensure that they are up to date. Check the DDM table ensure there was no core corruption.

**-30071**

**SQLSTATE:** 58015

**Explanation:** An AS reply message indicated that an invalid DDM object was sent by TPFAR.

**System Action:** The statement cannot be executed. A dump is taken. The program is left in a connected state.

**System Programmer Response:** Examine the dump to determine the rejected object. Also, check the service level of the AS and TPFAR to ensure that they are up to date. Check the DDM table to ensure there was no core corruption.

**-30072**

**SQLSTATE:** 58016

**Explanation:** An AS reply message indicated that an invalid DDM parameter was sent by TPFAR.

**System Action:** The statement cannot be executed. A dump is taken. The program is left in a connected state.

**System Programmer Response:** Check the dump to determine the rejected parameter. Also, check the service level of the AS and TPFAR to ensure that they are up to date.

**-30073**

**SQLSTATE:** 58017

**Explanation:** An AS reply message indicated that an invalid DDM parameter value was sent by TPFAR.

**System Action:** The statement cannot be executed. A dump is taken. The program is left in a connected state.

**System Programmer Response:** Check the dump to determine the rejected parameter value. Check the service level of the AS and TPFAR to ensure that they are up to date. When the rejected parameter is identified, check the DDM

encode routine to see how this parameter was built.

---

**-30074**

**SQLSTATE:** 58018

**Explanation:** An invalid severity code or reply message was received from the remote AS.

**System Action:** The statement cannot be executed. A dump is taken. The program is left in a connected state.

**System Programmer Response:** Check for a possible mismatch in the AS or TPFAR manager levels.

---

**-30080**

**SQLSTATE:** 58019

**Explanation:** A communications error occurred. The current conversation has failed because of a failure of the underlying session. This can be cause by operator action or hardware failure.

**System Action:** The statement cannot be executed. The program is left in an unconnected but connectable state.

**System Programmer Response:** The communications console should be checked to determine the reason for the session failure.

# Glossary of Terms Related to TPFAR

This glossary defines terms and concepts related to TPFAR in this book. If you do not find the term you are looking for, refer to the Index, the *TPF Library Guide*, or the *Dictionary of Computing*, ZC20-1699.

## A

**application requester (AR) .** A DRDA component that transforms a database request into communication protocols for a distributed relational database system.

**application server (AS) .** A DRDA component that receives and processes database requests from an application requester.

## B

**bind process .** A process in which a relational database management system transforms the specification of an SQL statement into a sequence of internal operations to optimize data retrieval.

## C

**column .** A vertical arrangement of information in a table.

**cursor .** A control structure used by an application to retrieve and or update multiple rows in a table or to point to a row of interest within a table.

## D

**database management system (DBMS).** A software system that has a catalog describing the data it manages. The DBMS controls access to the data stored within it.

**database request module (DBRM) .** A DB2 data set member containing information about SQL commands. The DBRM is created by the precompiler and used in the bind process.

This file contains information on all SQL commands in the user application program. Included is information about how the SQL commands are executed and DB2's access strategy.

**DB2.** An IBM relational database management system for MVS operating systems.

**Distributed Relational Database Architecture (DRDA).** A relational database connection protocol consisting of protocols for communication between an application and a remote database, and communications between databases.

## H

**HCT .** Hotcon table.

**hotcon .** A hot conversation or hot connection, depending on the communication protocol used.

In LU 6.2, a TPF Advanced Program-to-Program Communications (TPF/APPC) conversation that remains allocated and active past the completion of the transaction. The TPF/APPC conversation parameters between the TPF Application Requester (TPFAR) and the DB2 system are saved in an entry in the hotcon table (HCT). When another entry control block (ECB) requests a conversation with the same remote application server, TPFAR reuses the active conversation.

In Transmission Control Protocol/Internet Protocol (TCP/IP), a TCP/IP connection that remains active past the completion of the transaction. The socket descriptors are saved in an entry in the hotcon table. When another entry control block (ECB) requests a connection with the same remote application server, the TPFAR reuses the active connection.

## L

**logical unit of work .** A sequence of SQL commands that DB2 treats as a single entity.

**logical unit of work identifier (LUWID) .** An ID consisting of the fully qualified LU name, logical unit of work instance number, and logical unit of work sequence number that uniquely identifies a logical unit of work within the network.

## P

**package .** An object containing a set of SQL statements that have been bound statically and are available for processing.

## R

**relational database (RDB).** A database in which the data are organized and accessed according to relations.

**remote unit of work.** A method of accessing distributed relational data in which users or applications can, within a single unit of work, read and update one system using multiple SQL statements.

**row.** A horizontal arrangement of information in a table.

# S

**SQLCODE .** A system-dependent SQL return code.

**SQLSTATE .** A system-independent SQL return code field for the outcome of the last executed SQL command.

**structured query language (SQL) .** The programming language used to define relational data, access relational data, and control access to relational database resources.

**SQL database management system directory (SDD).** A directory used by TPF to keep relational database (RDB) information. Most of this information is set up with the ZSQLD command.

**structured query language communications area (SQLCA).** A structure that contains information about the execution of the SQL commands.

**Systems Application Architecture (SAA).** A set of software interfaces, conventions, and protocols that provide a foundation for designing and developing consistent applications across systems.

# T

**table .** A named data object consisting of a specific number of columns and some number of unordered rows.

# Index

## Special Characters



## Numerics



## A



## B



## C

## D



## E



## F



## H

## I



## J



## K



## L



## M



## N



## O



## P



## R

# S

# T

## V



## Z

File Number: S370/30XX-40
Program Number: 5748-T14

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SH31-0133-04