## Contents

Send Us Your Comments ................................................................................................................... xi

**Preface**............................................................................................................................................... xiii

- Intended Audience ........................................................................................................................... xiii
- Related Publications ........................................................................................................................ xiii
- Conventions........................................................................................................................................ xiv
- Documentation Accessibility .......................................................................................................... xvi

### 1 Introduction to Oracle Procedural Gateway for APPC

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>1-2</td>
</tr>
<tr>
<td>Terms</td>
<td>1-2</td>
</tr>
<tr>
<td>Examples and Sample Files</td>
<td>1-5</td>
</tr>
<tr>
<td>Features</td>
<td>1-6</td>
</tr>
<tr>
<td>Architecture</td>
<td>1-7</td>
</tr>
<tr>
<td>Starting the Gateway</td>
<td>1-9</td>
</tr>
<tr>
<td>Communication</td>
<td>1-9</td>
</tr>
<tr>
<td>Remote Procedural Call Functions</td>
<td>1-9</td>
</tr>
<tr>
<td>Remote Transaction Initiation</td>
<td>1-10</td>
</tr>
<tr>
<td>Data Exchange</td>
<td>1-10</td>
</tr>
<tr>
<td>Remote Transaction Termination</td>
<td>1-10</td>
</tr>
<tr>
<td>Transaction Types</td>
<td>1-11</td>
</tr>
<tr>
<td>Simple Communication Between Mainframe and Oracle Integrating Server</td>
<td>1-11</td>
</tr>
<tr>
<td>Steps to Communication Between Oracle Server and Mainframe</td>
<td>1-12</td>
</tr>
<tr>
<td>Writing TIPs to Generate PL/SQL Programs</td>
<td>1-13</td>
</tr>
</tbody>
</table>
2 Procedural Gateway Administration Utility

Overview of PGAU ................................................................. 2-2

COMMIT/ROLLBACK Processing ............................................. 2-3
  COMMIT Processing ......................................................... 2-3
  ROLLBACK Processing ................................................... 2-3

Invoking PGAU ........................................................................ 2-3

Definitions and Generation .................................................. 2-4
  Process to Define and Test a TIP ....................................... 2-5
  Definition Names ............................................................ 2-5
  Definition Versioning ....................................................... 2-6
  Keywords ............................................................................ 2-7

PGAU Commands .................................................................... 2-7
  CONNECT ............................................................................... 2-8
  DEFINE CALL ................................................................. 2-9
  DEFINE DATA ............................................................... 2-10
  DEFINE TRANSACTION .................................................. 2-13
  DESCRIBE ......................................................................... 2-18
  DISCONNECT ................................................................. 2-19
  EXECUTE ............................................................................. 2-20
  EXIT .................................................................................... 2-21
  GENERATE .......................................................................... 2-22
  GROUP ................................................................................. 2-29
  HOST ................................................................................... 2-31
  PRINT .................................................................................. 2-32
  REDEFINE DATA ............................................................ 2-33
  REM .................................................................................... 2-38
  REPORT ................................................................................ 2-39
  SET ..................................................................................... 2-42
  SHOW .................................................................................. 2-45
  SPOOL ................................................................................ 2-47
  UNDEFINE CALL .............................................................. 2-48
  UNDEFINE DATA ............................................................ 2-49
  UNDEFINE TRANSACTION ............................................... 2-50
  VARIABLE ........................................................................... 2-51
3  Creating a TIP

Granting Privileges for TIP Creators ................................................................. 3-2
Evaluating the RHT ......................................................................................... 3-2
  Identify the Remote Host Transaction......................................................... 3-2
  PGAU DEFINE CALL Command................................................................. 3-3
  PGAU DEFINE DATA Command................................................................. 3-4
  PGAU DEFINE TRANSACTION Command.................................................. 3-4
Writing the PGAU Statements ..................................................................... 3-5
  Writing a PGAU Script File...................................................................... 3-6
Defining and Generating the TIP ................................................................. 3-7
Compiling the TIP ......................................................................................... 3-8
TIP Content Documentation (tipname.doc) .................................................. 3-9

4  Client Application Development

Overview of Client Application ..................................................................... 4-2
Preparing the Client Application ............................................................... 4-3
  Understanding the Remote Host Transaction Requirements .................. 4-4
  TIP Content and Purpose ......................................................................... 4-4
Remote Host Transaction Types .................................................................... 4-5
  One-Shot Transactions ............................................................................. 4-5
  Persistent Transactions .............................................................................. 4-6
  Multi-Conversational Transactions ......................................................... 4-7
Customized TIPs for Each Remote Host Transaction .................................... 4-8
Client Application Requirements ............................................................... 4-9
Ensuring TIP and Remote Transaction Program Correspondence ............... 4-14
  DATA Correspondence ............................................................................. 4-14
  CALL Correspondence ............................................................................. 4-16
  Flexible Call Sequence ............................................................................. 4-17
  Call Correspondence Order Restrictions .................................................. 4-18
  TRANSACTION Correspondence ............................................................. 4-18
Calling the TIP from the Client Application ................................................... 4-20
  Declaring TIP Variables ............................................................................ 4-20
  Initializing the Conversation ..................................................................... 4-22
  Transaction Instance Parameter .............................................................. 4-23
  Overriding TIP Initializations ................................................................... 4-24
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security Considerations</td>
<td>4-26</td>
</tr>
<tr>
<td>Exchanging Data</td>
<td>4-27</td>
</tr>
<tr>
<td>Terminating the Conversation</td>
<td>4-27</td>
</tr>
<tr>
<td>Error Handling</td>
<td>4-27</td>
</tr>
<tr>
<td>Granting Execute Authority</td>
<td>4-28</td>
</tr>
<tr>
<td>Executing the Application</td>
<td>4-28</td>
</tr>
<tr>
<td>APPC Conversation Sharing</td>
<td>4-28</td>
</tr>
<tr>
<td>APPC Conversation Sharing Concepts</td>
<td>4-29</td>
</tr>
<tr>
<td>APPC Conversation Sharing Usage</td>
<td>4-30</td>
</tr>
<tr>
<td>APPC Conversation Sharing TIP Compatibility</td>
<td>4-30</td>
</tr>
<tr>
<td>APPC Conversation Sharing for TIPs That Are Too Large</td>
<td>4-31</td>
</tr>
<tr>
<td>APPC Conversation Sharing Example</td>
<td>4-32</td>
</tr>
<tr>
<td>APPC Conversation Sharing Overrides and Diagnostics</td>
<td>4-35</td>
</tr>
<tr>
<td>Application Development with Multi-Byte Character Set Support</td>
<td>4-35</td>
</tr>
<tr>
<td>Modifying a Terminal-Oriented Transaction to Use APPC</td>
<td>4-36</td>
</tr>
<tr>
<td>Privileges Needed to Use TIPs</td>
<td>4-37</td>
</tr>
</tbody>
</table>

## 5 Implementing Commit-Confirm

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview of Commit-Confirm</td>
<td>5-2</td>
</tr>
<tr>
<td>Supported OLTPs</td>
<td>5-2</td>
</tr>
<tr>
<td>Required Components</td>
<td>5-3</td>
</tr>
<tr>
<td>Configuring Commit-Confirm</td>
<td>5-5</td>
</tr>
<tr>
<td>Configuring the Oracle Integrating Server</td>
<td>5-5</td>
</tr>
<tr>
<td>Configuring the Gateway Initialization Parameters</td>
<td>5-6</td>
</tr>
<tr>
<td>Configuring the OLTP</td>
<td>5-7</td>
</tr>
<tr>
<td>Application Design Requirements</td>
<td>5-8</td>
</tr>
<tr>
<td>Commit-Confirm Architecture</td>
<td>5-9</td>
</tr>
<tr>
<td>Components</td>
<td>5-9</td>
</tr>
<tr>
<td>Interactions</td>
<td>5-10</td>
</tr>
<tr>
<td>Commit-Confirm Flow</td>
<td>5-10</td>
</tr>
<tr>
<td>Commit-Confirm Logic Flow, Step by Step</td>
<td>5-10</td>
</tr>
<tr>
<td>Gateway Server Commit-Confirm Transaction Log</td>
<td>5-13</td>
</tr>
<tr>
<td>Sample Applications</td>
<td>5-14</td>
</tr>
</tbody>
</table>
6 Problem Determination

TIP Definition Errors ................................................................. 6-2
Problem Analysis with PG DD Diagnostic References ..................... 6-3
Problem Analysis with PG DD Select Scripts ................................ 6-4
Data Conversion Errors ................................................................. 6-5
Problem Analysis of Data Conversion and Truncation Errors ............... 6-6
Problem Analysis with TIP Runtime Traces .................................... 6-8
TIP Runtime Trace Controls .......................................................... 6-10
  Generating Runtime Data Conversion Trace and Warning Support .......... 6-10
  Controlling TIP Runtime Conversion Warnings ................................ 6-11
  Controlling TIP Runtime Function Entry/Exit Tracing ......................... 6-11
  Controlling TIP Runtime Data Conversion Tracing ......................... 6-11
  Controlling TIP Runtime Gateway Exchange Tracing ......................... 6-12
Suppressing TIP Warnings and Tracing ........................................ 6-12
Gateway Server Tracing ................................................................. 6-13
  Defining the Gateway Trace Destination ......................................... 6-13
  Enabling the Gateway Trace ........................................................ 6-14
    Enabling the APPC Trace Using Initialization Parameters .............. 6-15
    Enabling the APPC Trace Dynamically from PL/SQL ....................... 6-15
Sample Gateway Server Initialization and Trace Output Files ............... 6-16

A Procedural Gateway for APPC Data Dictionary

PG DD Environment Dictionary ...................................................... A-2
  Environment Dictionary Sequence Numbers ..................................... A-2
  Environment Dictionary Tables .................................................. A-3
PG DD Active Dictionary ............................................................ A-4
  Active Dictionary Versioning ..................................................... A-4
  Active Dictionary Sequence Numbers ........................................... A-5
  Active Dictionary Tables .......................................................... A-5

B TIP Internals

  Background Reading ................................................................. B-2
  PL/SQL Package and TIP File Separation ...................................... B-2
    Independent TIP Body Changes ............................................... B-3
Dependent TIP Body or Specification Changes ................................................................. B-4
Inadvertent Alteration of TIP Specification ................................................................. B-5
TIP Call to APPC Call Relationship ........................................................................... B-5
  TIP Function Calls and Remote Transaction Program APPC Calls ....................... B-6
  Parameter Datatype Conversion .............................................................................. B-6
  APPC Send/Receive Synchronization .................................................................... B-7
Oracle Procedural Gateway for APPC Restrictions ................................................ B-9
TIP Specification ........................................................................................................ B-10
TIP Body ..................................................................................................................... B-11
TIP Initialization ........................................................................................................ B-18
TIP Init and Term Functions ..................................................................................... B-19
  TIP Initialization Function .................................................................................. B-19
  TIP Terminate Function ..................................................................................... B-20
C  Gateway RPC Interface
  Calling Gateway Functions to Execute Transaction Programs ......................... C-2
    PGAINIT and PGAINIT_SEC ................................................................. C-2
    PGAXFER ................................................................................................. C-5
    PGATERM ................................................................................................. C-8
    PGATCTL ................................................................................................. C-8
    PGATRAC ................................................................................................. C-9
D  The UTL_PG and UTL_RAW Interfaces
  UTL_RAW Functions .......................................................................................... D-2
    BIT_AND .................................................................................................. D-2
    BIT_COMPLEMENT ................................................................................. D-3
    BIT_OR .................................................................................................... D-3
    BIT_XOR ............................................................................................... D-4
    CAST_TO_RAW ..................................................................................... D-5
    CAST_TO_VARCHAR2 ................................................................. D-6
    COMPARE ............................................................................................. D-6
    CONCAT ................................................................................................. D-7
    CONVERT .............................................................................................. D-8
    COPIES ................................................................................................. D-9
    LENGTH ................................................................................................. D-10
OVERLAY.......................................................................................................................... D-10
REVERSE.......................................................................................................................... D-12
SUBSTR .................................................................................................................................. D-12
TRANSLATE....................................................................................................................... D-13
TRANSLITERATE............................................................................................................ D-14
X RANGE.............................................................................................................................. D-16

UTL_P G Functions ............................................................................................................. D-17
Common Parameters........................................................................................................... D-17
Input Parameters ................................................................................................................ D-18
Output Parameters ............................................................................................................. D-19
RAW_TO_NUMBER.............................................................................................................. D-19
NUMBER_TO_RAW.............................................................................................................. D-23
MAKE_RAW_TO_NUMBER_FORMAT ................................................................................ D-26
MAKE_NUMBER_TO_RAW_FORMAT .................................................................................. D-29
RAW_TO_NUMBER_FORMAT............................................................................................. D-32
NUMBER_TO_RAW_FORMAT.............................................................................................. D-33
W M S GCNT ...................................................................................................................... D-34
W MSG .................................................................................................................................. D-35

NUMBER_TO_RAW and RAW_TO_NUMBER Argument Values ........................................ D-37

E Administration Utility Samples
Sample PGAU DEFINE DATA Statements ........................................................................... E-2
Sample PGAU DEFINE CALL Statements ........................................................................... E-3
Sample PGAU DEFINE TRANSACTION Statement ............................................................ E-3
Sample PGAU GENERATE Statement ................................................................................ E-3
Sample Implicit Versioning Definitions .............................................................................. E-4
Sample PGAU REDEFINE DATA Statements .................................................................... E-8
Sample PGAU UNDEFINE Statements ................................................................................. E-9
Sample PGAU REPORT Output .......................................................................................... E-9
Sample PGAU Script File (pgadb2i.ctl) ............................................................................. E-14
Sample TIP Content Documentation ................................................................................ E-16
Sample TIP Trace Output .................................................................................................... E-19
Sample TIP Output ............................................................................................................. E-21
F  CICS APPC Transaction Samples
Description of Sample CICS APPC Transaction ................................................................. F-2
COBOL Source Code ........................................................................................................... F-2
Sample pgacics.sql File ..................................................................................................... F-13

G  Datatype Conversions
Procedural Gateway for APPC Components ..................................................................... G-2
Length Checking .................................................................................................................. G-3
Parameters Over 32K in Length ......................................................................................... G-3
Conversion ......................................................................................................................... G-4
  USAGE(PASS) .................................................................................................................... G-4
    Datatype Conversion ....................................................................................................... G-4
    Alphanumeric and DBCS Editing Field Positions ........................................................ G-6
    Format Conversion ........................................................................................................ G-8
  USAGE(ASIS) .................................................................................................................. G-13
  USAGE(SKIP) .................................................................................................................. G-13
PL/SQL Naming Algorithms .............................................................................................. G-13
  Delimiters ....................................................................................................................... G-13
    Qualified Compound Names ......................................................................................... G-14
    Truncated and Non-Unique Names ............................................................................ G-14
    Duplicate Names .......................................................................................................... G-15

H  National Language Support
Overview ....................................................................................................................... H-2
Languages Supported for Messages ................................................................................ H-2
Languages Supported for Data Conversion ...................................................................... H-3

Index
Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and the chapter, section, and page number (if available). You can send comments to us at the following email address:

    infoibm_us@oracle.com

Please include your name, address, and telephone number.

If you have problems with the software, please contact your local Oracle Support Services.
Read this guide if you are responsible for tasks such as:

- determining hardware and software requirements
- installing, configuring, or administering the Oracle Procedural Gateway for APPC
- developing applications
- determining security requirements
- determining and resolving problems

Before using this guide to administer the gateway, you should understand the fundamentals of the operating system for your platform and procedural gateways.

**Intended Audience**

This guide is intended for anyone responsible for administering the gateway, and also for developers writing applications that access remote host databases through the gateway.

**Related Publications**

The *Oracle Procedural Gateway for APPC User’s Guide* is included as part of your product shipment. Also included is:

- *Oracle Procedural Gateway for APPC Messages Guide*
- *Oracle Procedural Gateway for APPC Installation and Configuration Guide for UNIX*
You might also need Oracle server and Oracle Net documentation. The following Oracle9i publications are referenced in this guide:

- Oracle 9i Server Installation Guide
- Oracle 9i Server Administrator’s Guide
- Oracle9i Server Application Developer’s Guide
- Oracle9i Server Concepts
- Oracle9i Administrator’s Guide

Refer to the Oracle Technical Publications Catalog and Price Guide for a complete list of documentation provided for Oracle products.

Conventions

Examples of input and output for the gateway and Oracle environment are shown in a special font:

```
$ mkdir /ORACLE/your_name
```

All output is shown as it appears. For input, refer to the list of conventions and their meanings in the following table:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>example text</td>
<td>Words or phrases, such as <code>mkdir</code> and <code>ORACLE</code>, must be entered exactly as spelled and in the letter case shown. In this example, <code>mkdir</code> must be entered in lowercase letters and <code>ORACLE</code> in uppercase letters.</td>
</tr>
<tr>
<td>italic text</td>
<td>Italicized uppercase or lowercase, such as <code>your_name</code>, indicates that you must substitute a word or phrase, such as the actual directory name.</td>
</tr>
<tr>
<td>BOLD text or bold italic TEXT</td>
<td>Bold words or phrases refer to a file or directory structure, such as a directory, path, or file name.</td>
</tr>
<tr>
<td>...</td>
<td>Ellipses indicate that the preceding item can be repeated. You can enter an arbitrary number of similar items.</td>
</tr>
<tr>
<td>[]</td>
<td>Curly braces indicate that one of the enclosed arguments is required. Do not enter the braces themselves.</td>
</tr>
<tr>
<td></td>
<td>Vertical lines separate choices.</td>
</tr>
</tbody>
</table>
Other punctuation, such as commas, quotation marks or the pipe symbol (|) must be entered as shown unless otherwise specified. Directory names, file IDs and so on appear in the required letter case in examples. The same convention is used when these names appear in text, and the names are highlighted in **bold**. The use of *italics* indicates that those portions of a file ID that appear in *italics* can vary.

Gateway commands, file IDs, reserved words and keywords appear in uppercase in examples and text. UNIX commands, environment variables, and keywords appear in the required letter case in examples and text. Reserved words and keywords must always be entered as shown; they have reserved meanings within the Oracle system.

### SQL*Plus Prompts
The SQL*Plus prompt, **SQL>**, appears in SQL statement and SQL*Plus command examples. Enter your response at the prompt. Do not enter the text of the prompt, **SQL>**, in your response.

### UNIX Prompts
The UNIX prompt, $, appears in UNIX command examples. Enter your response at the prompt. Do not enter the text of the prompt, $, in your response. A dollar sign, $, is also part of some UNIX directory names and should not be mistaken for a prompt character.

### PGAU Prompts
The PGAU prompt, **PGAU>**, appears in PGAU command examples. Enter your response at the prompt. Do not enter the text of the prompt, **PGAU>**, in your response.

### Directory Names
Throughout this document, there are references to the directories in which product-related files reside. **$ORACLE_HOME** is used to represent the Oracle home directory. This is the default location for Oracle products. If you have installed into a location other than **$ORACLE_HOME**, replace all references to **$ORACLE_HOME** with the drive and path specification you have used.
Storage Measurements

Storage measurements use these abbreviations:

- K, for kilobyte which equals 1024 bytes
- M, for megabyte which equals 1 048 576 bytes
- G, for gigabyte which equals 1 073 741 824 bytes

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle Corporation is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at: http://www.oracle.com/accessibility/

Accessibility of Code Examples in Documentation  JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation  This documentation may contain links to Web sites of other companies or organizations that Oracle Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.
This guide is intended for users of the following UNIX-based platforms:

- HP9000 Series 700/800
- Sun Solaris, and
- IBM RS/6000

The Oracle Procedural Gateway for APPC allows you to initiate transaction program execution on remote online transaction processors (OLTPs). The Oracle Procedural Gateway for APPC provides Oracle applications seamless access to IBM mainframe data and services through remote procedural call (RPC) processing.

Read this chapter to learn more about the architecture, uses, and features of the Oracle Procedural Gateway for APPC.

This chapter contains the following sections:

- **Overview** on page 1-2
- **Terms** on page 1-2
- **Examples and Sample Files** on page 1-5
- **Features** on page 1-6
- **Architecture** on page 1-7
- **Transaction Types** on page 1-11
Overview

The Oracle Procedural Gateway for APPC extends the remote procedural call (RPC) facilities available with the Oracle server. The gateway enables any client application to use PL/SQL to request execution of a remote transaction program residing on a host. The gateway provides RPC (remote procedural call) processing to systems using the SNA APPC (Advanced Program-to-Program Communication) protocol. This architecture allows efficient access to invaluable data and transactions available on the IBM mainframe.

The use of a generic and standard protocol, APPC, allows the gateway to access a multitude of systems. The gateway can communicate with virtually any APPC-enabled system, including IBM Corporation’s CICS on any platform, IBM Corporation’s IMS and APPC/MVS, and Compute Associates’ IDMS. These transaction monitors provide access to a broad range of systems, allowing the gateway to access many datastores, including VSAM, DB2 (static SQL), IMS, IDMS, ADABAS, and others.

The gateway requires no Oracle software on the remote system. Thus, the gateway uses existing transactions with little or no programming effort on the remote system. The gateway helps leverage your investment in existing applications and gives you the ability to move toward new distributed environments.

Terms

The following terms are used throughout this guide.

gateway initialization file
This file is known as `initsid.ora` and it contains parameters that govern the operation of the gateway. Refer to Appendix A of the Oracle Procedural Gateway for APPC Installation and Configuration Guide for your platform more information.

gateway remote procedure
Oracle Procedural Gateway for APPC provides pre-built remote procedures. In general, the following three remote procedures are used:

- PGAINIT, which initializes transactions,
- PGAXFER, which transfers data
- PGATERM, which terminates transactions
Refer to "Remote Procedural Call Functions" on page 1-9, as well as to Appendix C, "Gateway RPC Interface" for more information about gateway remote procedures.

**Oracle integrating server**
This is any Oracle server instance that communicates with the Oracle Procedural Gateway for APPC to perform remote procedural calls that execute remote transaction programs (RTP). The Oracle integrating server can be on the same system as the gateway or on a different system. If it is on a different system, then Oracle Net is required on both systems. Refer to Figure 1–1, "Relationship of Gateway and Oracle Integrating Server on UNIX Host" and Figure 1–2, "Oracle Procedural Gateway for APPC Architecture" for a view of the gateway architecture.

**OLTP (Online Transaction Processor)**
OLTP is any of a number of online transaction processors available from other vendors, including CICS, IMS/TM, and IDMS-DC.

**PGA (Procedural Gateway Administration)**
PGA is a general reference within this guide to all or most components comprising the Oracle Procedural Gateway for APPC. This term is used when references to a specific product or component are too narrow.

**PGAU (Procedural Gateway Administration Utility)**
PGAU is the tool that is used to define and generate PL/SQL transaction interface packages (TIPs). Refer to Chapter 2, "Procedural Gateway Administration Utility" for more information about PGAU.

**PG DD (Data Dictionary)**
This component of the Oracle Procedural Gateway for APPC is a repository of remote host transaction definitions and data definitions. PGAU accesses definitions in the PG DD when generating TIPs. The PG DD has datatype dependencies because it supports the PGAU and is not intended to be directly accessed by the customer. Refer to Appendix A, "Procedural Gateway for APPC Data Dictionary" for a list of PG DD tables.

**PGDL (Procedural Gateway Definition Language)**
PGDL is the collection of statements used to define transactions and data to the PGAU.
PL/SQL stored procedure specification (PL/SQL package)
This is a precompiled PL/SQL procedure that is stored in the Oracle integrating server.

remote procedural call (RPC)
RPC is a programming call that executes program logic on one system in response to a request from another system. See "gateway remote procedure" on page 1-2 for more information, and refer to Appendix C, "Gateway RPC Interface" as well.

remote transaction program (RTP)
A remote transaction program is a customer-written transaction, running under the control of an OLTP, which the user invokes remotely using a PL/SQL procedure. To execute a remote transaction program through the gateway, you must use RPC to execute a PL/SQL program to call the gateway functions.

transaction interface package (TIP)
A TIP is an Oracle PL/SQL package that exists between your application and the remote transaction program. The transaction interface package, or TIP, is a set of PL/SQL stored procedures that invoke the remote transaction program through the gateway. TIPs perform the conversion and reformating of remote host data using PL/SQL and UTL_RAW/UTL_PG functions.

UTL_RAW PL/SQL package (UTL_RAW functions)
This component of the Oracle Procedural Gateway for APPC is a series of data conversion functions for PL/SQL RAW variables and remote host data. The types of conversions performed depend on the language of the remote host data. Refer to "UTL_RAW Functions" in Appendix D for more information.

UTL_PG PL/SQL package (UTL_PG functions)
This component of the Oracle Procedural Gateway for APPC is a series of COBOL numeric data conversion functions.

Refer to "NUMBER_TO_RAW and RAW_TO_NUMBER Argument Values" in Appendix D for supported numeric datatype conversions.

Figure 1–1 illustrates where the terminology discussed above applies to the gateway’s architecture.
Examples and Sample Files

This guide uses a CICS-DB2 inquiry as an example. Transaction Interface Package (TIP) PGADB2I sends an employee number, `empno`, to a DB2 application and receives an employee record, `emprec`.

The CICS-DB2 inquiry sample and its associated PGAU commands are documented in Appendix F, "CICS APPC Transaction Samples" and Appendix E, "Administration Utility Samples" respectively. These samples are also available in the `$ORACLE_HOME/pg4appc/demo/CICS` directory. The sample CICS-DB2 inquiry used as an example in this chapter is in files `pgadb2i.pkh` and `pgadb2i.pkb`.

Introduction to Oracle Procedural Gateway for APPC  1-5
Features

Refer to the README.doc file in the same directory for information about installing and using the samples.

Features

Using the Oracle Procedural Gateway for APPC to access data has the following advantages:

- Fast interface
  The gateway is optimized so that remote execution of a program is achieved with minimum network traffic. The interface to the gateway is an optimized PL/SQL stored procedure specification (called the "PL/SQL package") precompiled in the Oracle integrating server. Because there are no additional software layers on the remote system, overhead occurs only when your program executes.

- Location transparency
  Client applications need not be operating system-specific. For example, your application can call a program in a CICS region on MVS. If you move the program to a CICS region on OS/2, then you need not change the application.

- Application transparency
  Users calling applications that execute a remote transaction program are unaware that a request is sent to a host.

- Flexible interface
  You can use the gateway to interface with existing procedural logic or to integrate new procedural logic into an Oracle integrating server environment.

- Oracle server integration
  The integration of the Oracle server with the gateway enables the gateway to benefit from existing and future Oracle features. For example, the gateway can be called from an Oracle stored procedure or database trigger.

- Transactional support
  The gateway and the Oracle integrating server allow remote transaction updates and Oracle server updates to be performed in a coordinated fashion. Oracle two-phase commit protection is extended to remote transactions without any special programming in the Oracle application.
Wide selection of tools
The gateway supports any tool or application that supports PL/SQL.

PL/SQL code generator
The Oracle Procedural Gateway for APPC provides a powerful development environment, including:
- a dictionary to store information relevant to the remote transaction
- a tool to generate the PL/SQL Transaction Interface Package, or TIP
- a report utility to view the information stored in the gateway dictionary
- a complete set of tracing and debugging facilities
- a wide set of samples to demonstrate the use of the product against datastores such as DB2, IMS, IDMS, CICS, and ADABAS

Site autonomy and security
The Oracle Procedural Gateway for APPC provides site autonomy, allowing you to do such things as authenticate users. It also provides role-based security compatible with any security package running on your mainframe computer.

Automatic conversion
Through the TIP, the following conversions are performed:
- ASCII to and from EBCDIC
- remote transaction program datatypes to and from PL/SQL datatypes
- National Language Support for many languages

Architecture
The architecture of the Oracle Procedural Gateway for APPC consists of three components:

1. The Oracle integrating server
This server should include the distributed option. The Oracle integrating server is usually installed on the same UNIX system as the Oracle Procedural Gateway for APPC.

If you install the Oracle integrating server on a system other than the UNIX system on which the gateway is installed, then you must install Oracle Net with the Oracle integrating server and with the gateway. The Oracle integrating
server must be capable of connecting to the gateway through any supported Oracle Net protocol. The Oracle integrating server can also be used for non-gateway applications. Refer to the Oracle Procedural Gateway for APPC Installation and Configuration Guide for a list of Oracle Net protocols supported by the gateway and tools.

2. The gateway

The Oracle Procedural Gateway for APPC must be installed on a UNIX operating system.

3. An OLTP

The OLTP must be on a system accessible to UNIX using the SNA APPC protocol. Multiple Oracle integrating servers can access the same gateway. A single UNIX gateway installation can be configured to access more than one OLTP.

Figure 1–2 illustrates the architecture of the Oracle Procedural Gateway for APPC.

Figure 1–2 Oracle Procedural Gateway for APPC Architecture
Starting the Gateway

The gateway is not started in the same way as an Oracle database instance. It has no background processes and does not need a management utility such as Oracle Enterprise Manager. Each Oracle integrating server user session that accesses a particular gateway creates an independent process on the UNIX system that runs the gateway server and executes SNA functions to communicate with an OLTP.

Communication

All of the communication between the Oracle integrating server, the gateway, and the target system are handled through a Transaction Interface Package (TIP). The TIP is a standard PL/SQL procedural gateway program (PL/SQL package) that provides the following functions:

- declares the PL/SQL variables that can be exchanged with a remote transaction program
- calls the gateway packages that handle the APPC communication for starting the conversation, exchanging data, and terminating the conversation
- handles all datatype conversions between PL/SQL datatypes and the target program datatypes

The Procedural Gateway Administration Utility (PGAU), provided with the gateway, automatically generates the TIP specification.

The gateway is identified to the Oracle integrating server using a database link. The database link is the same construct used to identify other Oracle server databases. The functions in the gateway are referenced in PL/SQL as:

`function_name@dblink_name`

Remote Procedural Call Functions

The Oracle Procedural Gateway for APPC provides a set of functions that are invoked by the client through remote procedural call (RPC). These functions direct the gateway to initiate, transfer data with, and terminate remote transaction programs running under an OLTP on another system.
The remote procedural call functions and the correlating commands invoked in the gateway and remote system are shown in the following table:

<table>
<thead>
<tr>
<th>Application</th>
<th>Oracle TIP</th>
<th>Gateway</th>
<th>Remote System</th>
</tr>
</thead>
<tbody>
<tr>
<td>call tip_init</td>
<td>tip_init call pgainit@gateway</td>
<td>PGAINIT</td>
<td>Initiate program</td>
</tr>
<tr>
<td>call tip_main</td>
<td>tip_main call pgaxfer@gateway</td>
<td>PGAXFER</td>
<td>Exchange data</td>
</tr>
<tr>
<td>call tip_term</td>
<td>tip_term call pgaterm@gateway</td>
<td>PGATERM</td>
<td>Terminate program</td>
</tr>
</tbody>
</table>

**Remote Transaction Initiation**

The TIP initiates a connection to the target system using one of the gateway functions, PGAINIT. PGAINIT provides, as input, the required SNA parameters to start a conversation with the target transaction program. These parameters are sent across the SNA network, which returns a conversation identifier to PGAINIT. Any future calls to the target program use the conversation identifier as an INPUT parameter.

**Data Exchange**

After the conversation is established, a procedural gateway function called PGAXFER can exchange data in the form of input and output variables. PGAXFER sends and receives buffers to and from the target transaction program. The gateway sees a buffer as only a RAW stream of bytes. The TIP that is residing in the Oracle integrating server is responsible for converting the application’s PL/SQL datatypes to RAW before sending the buffer to the gateway. It is also responsible for converting RAW to the PL/SQL datatypes before returning the results to the application.

**Remote Transaction Termination**

When communication with the remote program is complete, the gateway function PGATERM terminates the conversation between the gateway and the target system. PGATERM uses the conversation identifier as an INPUT parameter to request conversation termination.
Transaction Types

The Oracle Procedural Gateway for APPC supports three types of transactions that read data from and write data to remote systems:

- **One-shot**
  
  In a one-shot transaction, the application executes initialization, exchanges data, and terminates the connection, all in a single call.

- **Persistent**
  
  In a persistent transaction, multiple calls to exchange data with the remote transaction can be executed before terminating the conversation.

- **Multi-conversational**
  
  In a multi-conversation transaction, the procedural gateway server can be used to exchange multiple records in one call to the remote transaction program.

Refer to "Remote Host Transaction Types" in Chapter 4, "Client Application Development" for more information about transaction types.

The following examples demonstrate the power of the Oracle Procedural Gateway for APPC:

- You can initiate a CICS transaction on the mainframe to retrieve data from a VSAM file for a PC application.
- You can modify and monitor the operation of a remote process control computer.
- You can initiate an IMS/TM transaction that executes static SQL in DB2.
- You can initiate a CICS transaction that returns a large number of records in a single call.

Simple Communication Between Mainframe and Oracle Integrating Server

The Oracle Procedural Gateway for APPC lets you write your own procedures to begin transferring information between the Oracle integrating server and a variety of programs on an IBM mainframe, including IBM CICS, IMS, and APPC/MVS.

For an illustration of the communications function of the Oracle Procedural Gateway for APPC, refer to “Sample pgacics.sql File” in Appendix F, which is a simple sample communication between the Oracle server and CICS on an IBM mainframe. Executing a simple PL/SQL procedure (like pgacics.sql) causes the
Oracle server to invoke the procedural gateway, which uses APPC to converse with the **PGAFLIP RTP** in the mainframe.

---

**Note:** You will already have created the following stored procedure when you configured the gateway. If you are unsure whether communication has been established, execute the sample `pgacics.sql` file located in the `$ORACLE_HOME/pg4appc/demo/CICS` directory.

---

**Steps to Communication Between Oracle Server and Mainframe**

The following steps describe the UNIX-to-mainframe communications process illustrated in Figure 1–3, "Communication Between the Oracle Server and the Mainframe":

1. From the SQL*Plus, execute `pgacics`; this invokes the PL/SQL stored procedure in the Oracle integrating server.

   ```
   sqlplus <userid>/<password>@<database_specification_string>
   SQL> execute pgacics('==< .SCIC htiw gntacinummoc si yawetag ruoy ,snoitalutargnoC >==');
   ```

2. The `pgacics` PL/SQL stored procedure will start up the gateway. The gateway will start up communication with the mainframe through APPC and will call **PGAFLIP**.

3. **PGAFLIP** processes the input, generates the output and sends the output back to the procedural gateway.

4. Finally, the procedural gateway will send the output back to the PL/SQL stored procedure in the Oracle integrating server. The result is displayed in SQL*Plus:

   ```
   ==> Congratulations, your gateway is communicating with CICS. ===
   PL/SQL procedure successfully completed.
   ```

*Figure 1–3, "Communication Between the Oracle Server and the Mainframe"* illustrates the communications process described in steps 1 through 4, above.
Writing TIPs to Generate PL/SQL Programs

Most transactions are much larger and more complex than the simple sample `pgacics.sql` file referred to above and communication with a normal-sized RTP (remote transaction program) would require you to create an extremely long PL/SQL file. Oracle Procedural Gateway for APPC’s TIP function generates the PL/SQL procedure for you.

The following is a brief description of the four steps necessary for you to generate a TIP. Refer to Chapter 3, “Creating a TIP” for detailed information about this procedure, and refer to Chapter 2, “Procedural Gateway Administration Utility” for more information about PGAU.

All parameter names in this section are taken from a file called `pgadb2i.ctl` in the `$ORACLE_HOME/pga4appc/demo/CICS` directory. Refer to “Sample PGAU Script
1. Create a control file:

The user writes the control files. The control file has four main types of PGAU commands:

a. **DEFINE DATA.** This is used to define PL/SQL functions calls to be generated as part of the package.

   Sample define data:

   ```
   define data empno plsdname(empno) usage(pass) language(ibmvscobolii)
   infile("empno.cob");
   ```

b. **DEFINE CALL.** This is used to define input and output fields, using COBOL data definitions.

   Sample define call:

   ```
   define call db2imain pkgcall(pgadb2i_main)
   parms((empno in), (emprec out));
   ```

c. **DEFINE TRANSACTION.** This is used to group the above functions and specify other parameters on which the TIP depends.

   Sample define transaction:

   ```
   define transaction db2i call(db2imain, db2idiag)
   sideprofile(CICSPGA)
   tpname(DB2I)
   logmode(oraplu62)
   synclevel(0)
   nls_language("american_america.we8ebcdic37c");
   ```

d. **GENERATE.** This is used to generate the TIP specification files from the previously stored transaction, call and data definitions.

   Sample generate transaction:

   ```
   generate db2i pkgname(pgadb2i) pganode(pga) outfile("pgadb2i");
   ```
2. Run the control file within PGAU to create PG DD entries for the data, call, and transaction definitions, and to generate the specification files (for example, `pgadb2i.pkh` and `pgadb2i.pkb`):

```bash
$ pgau
PGAU> CONNECT<userid>/<password>@<database>_specification_string
PGAU> @pgadb2i.ctl
```

3. Run the specification files to create the PL/SQL stored procedures. Note that the header specification file (for example, `pgadb2i.pkh`) must be run first.

```sql
$ sqlplus<userid>/<password>@<database_specification_string>
SQL> @pgadb2i.pkh;
SQL> @pgadb2i.pkb;
```

4. The TIP is now ready for use. For convenience, it will usually be called using a driver procedure (for example, `db2idriv`), which will call the individual stored procedures in the correct order. Create the driver procedure and run it:

```sql
$ sqlplus <userid>/<password>@<database_specification_string>
SQL> @pgadb2id.sql
SQL> execute db2idriv('000320');
```
The Procedural Gateway Administration Utility (PGAU) is a utility that assists the PGA administrator or user to define the data which is to be exchanged with remote transaction programs. It generates the PL/SQL Transaction Interface Packages (TIPs) described in Chapter 3, "Creating a TIP", Chapter 4, "Client Application Development" and Appendix B, "TIP Internals".

This chapter contains the following sections:

- Overview of PGAU on page 2-2
- COMMIT/ROLLBACK Processing on page 2-3
- Invoking PGAU on page 2-3
- Definitions and Generation on page 2-4
- PGAU Commands on page 2-7
Overview of PGAU

PGAU maintains a data dictionary, PG DD, which is a collection of tables in an Oracle database. These tables hold the definitions of the remote transaction data and how that data is to be exchanged with the remote transaction program. Refer to "Ensuring TIP and Remote Transaction Program Correspondence" on page 4-14 for a discussion of the correlation between TIPs and their respective remote transaction programs. The PG DD contents define this correlation.

The PGA administrator or user defines the correlation between TIPs and the remote transaction program using the following PGAU commands (also called "statements"):

- PGAU DEFINE DATA statements, which describe the data to be exchanged.
- PGAU DEFINE CALL statements, which describe the exchange sequences.
- PGAU DEFINE TRANSACTION statements, which group the preceding CALL and DATA commands together and describe certain aspects unique to the remote transaction program, such as its network name or location.
- PGAU GENERATE statement, which the PGA administrator or user uses to specify and create the TIP specifications, after the TIP/transaction correlation has been defined in the PG DD. Additional PGAU commands needed to alter and delete definitions in the PG DD are described in "PGAU Commands" later in this chapter.

The PGAU commands are known collectively as Procedural Gateway Definition Language (PGDL). Any references to PGDL are to the collection of PGAU commands defined in this chapter.

PGAU provides editing and spooling facilities and the ability to issue SQL commands.

---

**Caution:** Do not use PGAU instead of SQL*Plus for general database administration.

---

Alternatively, PGAU commands can be supplied in a control file. The control file contains one or more PGAU commands for manipulating the PG DD or generating TIP specifications.

PGAU issues status messages on each operation. The message text is provided through Oracle NLS message support. PGAU processes each command in sequence. An error on a single command causes PGAU to skip that command.
To run PGAU, the PG Data Dictionary tables must already have been created. Refer to "Configuring the Oracle Integrating Server for First Time Installations" in Chapter 4 of the Oracle Procedural Gateway for APPC Installation and Configuration Guide.

**COMMIT/ROLLBACK Processing**

The following sections provide information on COMMIT/ROLLBACK processing.

**COMMIT Processing**

PGAU never issues COMMIT commands. As the user, it is your responsibility to COMMIT PG DD changes when all the changes are implemented. Otherwise Oracle issues a COMMIT command by default when you exit the PGAU session. If PG DD changes are not to be committed, you must run a ROLLBACK command before exiting.

**ROLLBACK Processing**

PGAU sets a savepoint at the beginning of each PGAU command that alters the PG DD and at the beginning of a PGAU GROUP. PGAU rolls back to the savepoint upon any PGAU command or group failure.

You can code COMMIT or ROLLBACK commands within PGAU scripts, or interactively in PGAU, but not within a GROUP.

Neither COMMIT nor ROLLBACK is issued for PGAU GENERATE or REPORT commands.

For information about grouping PGAU commands together to roll back changes in case of failure, refer to the discussion of the PGAU "GROUP" command later in this chapter.

**Invoking PGAU**

Before you can invoke PGAU, your Oracle server should already be set up. If it is not, refer to the chapter on configuring your Procedural Gateway for APPC, in the Oracle Procedural Gateway for APPC Installation and Configuration Guide for your platform.

Before executing PGAU, you must set the ORACLE_HOME environment variable to the directory into which the gateway server was installed.
If you want to receive PGAU messages in a language other than English, set the LANGUAGE environment variable to the appropriate value. For a list of supported languages and the syntax for the LANGUAGE setting, refer to Appendix H, "National Language Support".

PGAU is invoked by entering the `pgau` command. You can run prepared scripts of PGAU commands directly from the operating system prompt by specifying a command string on the command line using the following syntax:

```
$ pgau @command_file
$ pgau command=@command_file
$ pgau command="@command_file"
```

The default extension is `.sql`. Use the last form if the command filename contains non-alphanumeric characters.

To perform PG DD maintenance and PL/SQL package generation, you must connect to the Oracle integrating server from PGAU as user PGAADMIN, using the CONNECT command. The "PGAU Commands" section below discusses how to use the "CONNECT" command.

---

**Definitions and Generation**

This version of PGAU supports definition of remote transaction data in IBM VS/COBOL II, entered interactively or in a file. File input is supported for the DEFINE and REDEFINE DATA commands, and standard COBOL data division macros or "copybooks" can be supplied.

PGAU and the PG DD support different versions of user data and remote transaction definitions. This facilitates alteration and testing of data formats and transactions without affecting production usage.

Multiple versions of any data or transaction definitions might exist. It is your responsibility to ensure that versions stored and used in the PG DD are synchronized with the remote transactions. Neither the gateway, PGAU, nor generated TIPs provide this synchronization, but they will issue messages as error conditions are detected.

Data definitions must exist before being referenced by call definitions. Call definitions must exist before being referenced by transaction definitions.
Process to Define and Test a TIP

The general process for defining and testing a TIP for a given transaction is as follows:

1. Define input and output using COBOL data definitions.
2. Redefine the default datanames and PL/SQL variable names created by the above process (optional).
3. Define PL/SQL FUNCTION calls to be generated as part of the PL/SQL package.
4. Define a transaction that groups the above functions.
5. Generate the TIP specifications from the previously stored transaction, call, and data definitions.
6. Generate the TIP PL/SQL stored procedures.
7. Test the TIP by calling it from a high-level application.

Refer to Chapter 3, "Creating a TIP" for more information.

Definition Names

Definition names are unique identifiers that you designate through PGAU. The name is a string of 1 to 30 bytes. If punctuation or white space is included, the name must be specified within double quotes.

Names are assumed to be unique within the PG DD, except when duplicate names are intentionally distinguished by a unique version number. It is your responsibility to ensure name uniqueness.

Valid characters for PG DD definition names are:

- A through Z
- a through z
Definitions and Generation

- 0 through 9
- #
- $
- _ (underscore)

Note that unless defaults are overridden, transaction definition names might be PL/SQL package names, and transaction call names might be PL/SQL procedure names. Therefore, choose names that are syntactically correct for PL/SQL, making certain that they are also unique names within that system. As the user, it is your responsibility to ensure PL/SQL name compatibility.

Definition Versioning

The PG Data Dictionary tables contain the descriptions of transactions and data structures. There might be more than one version of a definition. Old versions are retained indefinitely.

In all PG DD operations, a definition or package is referred to by its name. That name can be qualified by a specific version number.

All version numbers:
- are supplied by Oracle Sequence Objects
- are purely numeric
- must be free from user alteration, suffixing, or prefixing

Refer to Appendix A, "Procedural Gateway for APPC Data Dictionary" and the pgdcr8.sql file in the $ORACLE_HOME/pg4appc/admin directory for the specific names of the Oracle Sequence Objects used for version number generation.

If an explicit version number is specified, it is presumed to be the version number of an existing definition, not a new definition. Such explicit references are used when:
- generating a TIP from a specific remote transaction version
- defining a remote transaction based on a specific data version

If no explicit version is specified:
- The latest (highest number) is assumed when a definition is being referenced. This is the MAX value selected from the VERSION column for all rows with the same definition name, not the CURRVAL number.
- The next (NEXTVAL number) is assumed when a definition is being added.
Version numbers might not be contiguous. Although version numbers are always increasing, multiple versions of a given definition might skip numbers. This is because the sequence object is shared for all definitions of the same type (Transaction, Call, or Data), and sequence object NEXTVAL is not restored in event of an Oracle database transaction ROLLBACK. Thus, NEXTVAL might be assigned to a different definition before the next version of the same definition.

Examples of valid definition names:

```
DEFINE TRANSACTION|CALL|DATA
  payroll     (new or latest definition)
  payroll_xaction (new or latest definition)
  payroll_xaction VERSION(3)... (an existing definition)
```

No attempt is made by PGAU to synchronize versions. Although the existence of dependent items is assured at definition time, deletion is done without reference to dependencies. For example, generating a TIP requires prior definition of the transaction, which requires prior definition of the calls, which require prior definition of the data. But nothing prevents PGAU from deleting an active data definition while a call definition still references it.

**Keywords**

All PGAU keywords can be specified in upper or lower case and are not reserved words. Reservation is not necessary because all keywords have known spelling and appear in predictable places, and because all data is delimited by parentheses, apostrophes, quotes, or blanks.

Note that all unquoted values specified by keywords are stored in the PG Data Dictionary in uppercase unless otherwise specified in the keyword description.

**PGAU Commands**

PGAU allows you to enter Procedural Gateway Administration commands (commands), such as DEFINE, UNDEFINE, REDEFINE, and GENERATE, in addition to normal SQL commands. The SET and SHOW commands are also implemented. In addition, the PGAU commands on the following pages are available to you.
CONNECT

Purpose
This command enables you to make a connection to PGAU. Use the CONNECT command to log on to an Oracle database, optionally specifying the user ID and password in addition to the Oracle instance. The CONNECT command has the following syntax:

Syntax
CONNECT [username|username/password|username@connect-string|username/password@connect-string]

Parameters
username/password is the username and password used to connect to PGAU, and
connect-string specifies the service name of the remote database. Refer to Oracle Net Administrator’s Guide for more information about specifying remote databases.

Examples
CONNECT
CONNECT SCOTT/TIGER
CONNECT SCOTT@OTHERSYS

CONNECT Usage Notes
■ Before connecting, you must set ORACLE_SID to the database SIDname.
■ If you want to connect to a remote database, you must set TNS_ADMIN to the full pathname of the directory in which the file tnsname is stored.
■ You do not need to place a semi colon (;) at the end of the command.
DEFINE CALL

Purpose
This command creates a new version of the PL/SQL call definition in the PG Data Dictionary.

Syntax
```
DEFINE CALL cname
    [PKGCALL(pcname)]
    [PARMS( (dname {IN | OUT | IN OUT} [VERSION(datavers)]), ...)];
```

Parameters
CALL  `cname` is a mandatory parameter. It is the name of the call definition to be created.

PKGCALL (`pcname`) is an optional parameter. It specifies the name of the PL/SQL package procedure or function by which the application might invoke the call. The default value, `cname`, is assumed if this operand is omitted, in which case `cname` must also be valid in PL/SQL syntax and unique within the transactions and TIPs referencing this call.

PARMS( (dname {IN | OUT | IN OUT} [VERSION(datavers)]), ... ) is an optional parameter. It specifies a list of previously defined data input to and output from this PL/SQL function call, and the type of parameter each (input to the call, output from, or both). The order in which the parameters are specified determines the order in which they must appear in subsequent calls to the TIP from an application. Each `dname` specifies a previously defined data item, and is mandatory. `{IN | OUT | IN OUT}` specifies the PL/SQL call mode of the parameter and indicates whether the `dname` data is sent, received, or both in the exchange with the remote transaction program. One must be chosen. VERS(`datavers`) is an optional specific version number of the `dname` data definition, if not the latest. If this operand is omitted, it is assumed that the call takes no parameters.

Examples
Refer to "Sample PGAU DEFINE CALL Statements" in Appendix E for examples of DEFINE CALL commands.
DEFINE CALL Usage Notes

- Version of the CALL definition is not specified and defaults to NEXTVAL of the Oracle Sequence Object for CALL.
- PKGCALL and PARMS can be specified in either order.
- You need to place a semi colon (;) at the end of the command.

DEFINE DATA

Purpose
This command creates a new version of the data definition in the PG Data Dictionary.

Syntax
DEFINE DATA dname
   [PLSDNAME(plsdvar)]
   [USAGE({PASS|ASIS|SKIP})]
   [COMPOPTS ('options')]
   LANGUAGE(language)
   { (definition) | INFILE("filespec") };

Parameters
The following table describes the DEFINE DATA parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA dname</td>
<td>A mandatory parameter. It is the name of the data definition to be created.</td>
</tr>
<tr>
<td>PLSDNAME (plsdvar)</td>
<td>An optional parameter. It is the name of the PL/SQL variable associated with dname. It becomes the name of a PL/SQL variable if the dname item is atomic data, or a PL/SQL record variable if the dname item is aggregate data (such as a record or structure), when the TIP is generated.</td>
</tr>
</tbody>
</table>
**Parameter** | **Description**  
---|---  
**USAGE**((PASS|ASIS|SKIP}) | is an optional parameter. It specifies the way the TIP handles the data items when exchanged in calls with the remote transaction.  
- PASS indicates that the item should be translated and exchanged with the transaction.  
- ASIS indicates the item is binary and, though exchanged, should not be translated.  
- SKIP indicates the item should be deleted from all exchanges.  
The default value, PASS, is assumed if this parameter is omitted.  
  
**COMPOPTS** ('options') | is an optional parameter. It specifies the compiler options used when compiling the data definition on the remote host.  
The only option currently supported is 'TRUNC(BIN)'. Note that the options must be enclosed in apostrophes ('') or quotes ("). TRUNC(BIN) is an IBM VS COBOL II option that affects the way halfword and fullword binary values are handled. Refer to "DEFINE DATA Usage Notes" on page 2-12 for further information on this option.  
  
**LANGUAGE** | is a mandatory parameter. It specifies the name of the programming language in the supplied definition. PGAU presently supports only IBMVS COBOL II.  
  
**definition** | is mutually exclusive with the INFILE parameter. It is an inline description of the data. The description must be provided in IBM VS COBOL II syntax, as indicated above. This inline description must begin with an opening parenthesis and end with a closing parenthesis. The opening parenthesis must be the last non-blank character on the line and the COBOL data definition must start on a new line, following the standard COBOL rules for column usage and continuations. The closing parenthesis and terminating semicolon must be on a separate line following the last line of the COBOL data definition. In COBOL, the specification is a COBOL data item or structure, defined in accordance with IBM VS COBOL II. Margins are assumed to be standard, and explicit or implicit continuation is supported. Datanames containing invalid characters (for example, ",") for PL/SQL use are translated to their closest equivalent and truncated as required.
Examples
Refer to “Sample PGAU DEFINE DATA Statements” in Appendix E for examples of DEFINE DATA commands.

### DEFINE DATA Usage Notes
- Version of the DATA definition is not specified and defaults to NEXTVAL of the Oracle Sequence Object for DATA.
- PLSDNAME, USAGE, and LANGUAGE can be specified in any order.
- INFILE ("filespec") is a platform-specific designation of a disk file.
- COMPOPTS ('TRUNC(BIN)') should be used only when the remote host transaction was compiled using IBM VS COBOL II with the TRUNC(BIN) compiler option specified. When this option is used, binary data items defined as PIC 9(4) or PIC S9(4) can actually contain values with 5 digits, and binary data items defined as PIC 9(9) or PIC S9(9) can actually contain values with 10 digits. Without COMPOPTS ('TRUNC(BIN)'), PGAU generates NUMBER(4,0) or NUMBER(9,0) fields for these data items, resulting in possible truncation of the values.

When COMPOPTS ('TRUNC(BIN)') is specified, PGAU generates NUMBER(5,0) or NUMBER(10, 0) fields for these data items, avoiding any truncation of the values. Care must be taken when writing the client application to ensure that invalid values are not sent to the remote host transaction.

For a PIC 9(4) the value must be within the range 0 to 32767, for a PIC S9(4) the value must be within the range -32767 to +32767, for a PIC 9(9) the value must be within the range 0 to 2,147,483,647, and for a PIC S9(9) the value must be within the range -2,147,483,647 to +2,147,483,647. VS COBOL II always reserves the high-order bit of binary fields for a sign, so the value ranges for unsigned fields are limited to the absolute values of the value ranges for signed fields. For further information, refer to the IBM manuals VS COBOL II Application Programming Guide, and VS COBOL II Application Programming Language Reference.
PGAU Commands

- Refer to "Alphanumeric and DBCS Editing Field Positions" in Appendix G, "Datatype Conversions" for information about how PGAU converts COBOL statements.
- You need to place a semi colon (;) at the end of the command.

DEFINE TRANSACTION

Purpose
This command creates a new version of the transaction definition in the PG Data Dictionary.

Syntax

```
DEFINE TRANSACTION tname
CALL(cname [VERS(callvers)], ...)
 [ENVIRONMENT(name)]
 {SIDEPROFILE(name) [LUNAME(name)] [TPNAME(name)]
   [LOGMODE(name)] |
   LUNAME(name) TPNAME(name) LOGMODE(name)}
 [SYNCLEVEL(0|1|2)]
 [NLS_LANGUAGE("nlsname")];
 [REMOTE_MBCS("nlsname")]
 [LOCAL_MBCS("nlsname")];
```

Parameters
The following table describes the DEFINE TRANSACTION parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSACTION tname</td>
<td>A mandatory parameter. It is the name of the transaction definition to be created. If you do not specify a package name (TIP name) in the GENERATE statement, the transaction name specified here will become the package name, by default. In that case, the tname must be unique and must be in valid PL/SQL syntax within the database containing the PL/SQL packages.</td>
</tr>
</tbody>
</table>
CALL (cname [VERS(callvers)], ...) A mandatory parameter. It specifies a list of previously defined calls (created with DEFINE CALL) which, taken together, comprise this transaction. The order in which the calls are specified here determines the order in which they are created by GENERATE, but not necessarily the order in which they might be called by an application. VERS(callvers) is an optional specific version number of the call definition, if not the latest.

The relative position of each cname in its left-to-right sequence is the seq# column in pga_trans_calls. For example:

CALL (cname1, cname2, cname3)

pga_trans_calls(seq#) = 1

2 3

ENVIRONMENT (name) Specifies the name of the host environment for this transaction, for example, "IBM370". If this parameter is omitted, IBM370 is assumed. IBM370 is the only environment supported by this version of PGAU.

SIDEPROFILE (name) Optional, but if omitted, the user must specify the parameters for LUNAME, TPNAME, and LOGMODE. It specifies the name of an SNA Side Information Profile which directs the APPC connection to the transaction manager. This name can be 1 to 8 characters in length. Name values can be alphanumeric with '@', '#', and '$' characters only if unquoted. Quoted values can contain any character, and delimited by quotes ('), or apostrophes ('). Case is preserved for all values.
### PGAU Commands

#### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| LUNAME (name) | Overrides the LUNAME within the Side Information Profile, if the Side Information Profile was specified. It specifies the SNA Logical Unit name of the transaction manager (OLTP).

This is either the fully-qualified LU name, 3 to 17 characters in length, or an LU alias 1 to 8 characters in length (when the SNA software on your gateway system supports LU aliases).

Name values can be alphanumeric with '@', '#', and '$' characters and a single period '.', to delimit the network from the LU, as in netname.luname, if fully qualified. Quoted values can contain any character, and delimited by quotes ('"), or apostrophes (''). Case is preserved for all values.

| TPNAME (name) | Overrides the TPNAME within the Side Profile, if the Side profile was specified. It specifies the APPC partner Transaction Program name to be invoked.

- For CICS, this must be the CICS Transaction ID and is 1 to 4 characters in length.
- For IDMS, this must be the IDMS Task Code and is 1 to 8 characters in length.
- For IMS, this must be the IMS Transaction Name and is 1 to 8 characters in length.
- For AS/400, this must be specified as "library/program" and cannot exceed 21 bytes.

Name values can be alphanumeric with '@', '#', and '$' characters only if unquoted. Quoted values can contain any character, and delimited by quotes ('"), or apostrophes (''). Case is preserved for all values.
LOGMODE(name)

Overrides the LOGMODE within the Side Information Profile, if the Side Information Profile was specified. It specifies the name of a VTAM logmode table entry to be used to communicate with this transaction, and is 1-8 characters in length.

Name values can be alphanumeric with '@', '#', and '$' characters only. Values cannot be quoted. Case is not preserved and always translated to upper case.

SYNCLEVEL (0|1|2)

An optional parameter. It specifies the APPC sync level of this transaction (0, 1, or 2). The default value of 0 is assumed if this operand is omitted, indicating the remote transaction program does not support synchronization. A value of 1 indicates that CONFIRM is supported, and a value of 2 indicates that SYNC is supported. SYNC support is required for two-phase commit.

Note that not all platforms support two-phase commit. Refer to the Oracle Procedural Gateway for APPC Installation and Configuration Guide for information on whether or not your platform supports two-phase commit.

NLS_LANGUAGE ("nlsname")

An optional parameter. The default value is "american_america.we8ebcdic37c". It is an Oracle NLS name in the language_territory.charset format. It specifies the Oracle NLS name in which the remote host data for all single-byte character set fields in the transaction are encoded. Refer to Appendix H, "National Language Support" for more information.
Examples

Refer to "Sample PGAU DEFINE TRANSACTION Statement" in Appendix E for examples of DEFINE TRANSACTIONS commands.

DEFINE TRANSACTION Usage Notes:

- NLS_LANGUAGE and the Oracle integrating server’s LANGUAGE specify default character sets to be used for conversion of all single-byte character fields for the entire transaction. These defaults can be overridden for each SBCS field by the REDEFINE DATA REMOTE_LANGUAGE or LOCAL_LANGUAGE parameters.

- The version of the TRANSACTION definition is not specified and defaults to NEXTVAL of the Oracle Sequence Object for TRANS.

- REMOTE_MBCS and LOCAL_MBCS specify the default multi-byte character sets to be used for conversion of all DBCS or MBCS fields for the entire transaction. This default can be overridden for each DBCS or MBCS field by the REDEFINE DATA REMOTE_LANGUAGE or LOCAL_LANGUAGE parameters.

- You must place ";" at the end of the command.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REMOTE_MBCS (&quot;nlsname&quot;)</td>
<td>This is an optional parameter. The default value is &quot;japanese_japan.jal6dbcs&quot;. It is an Oracle NLS name in the language_territory.charset format. It specifies the Oracle NLS name in which the remote host data for all multi-byte character set fields in the transaction are encoded. Refer to Appendix H, &quot;National Language Support&quot; for more information.</td>
</tr>
<tr>
<td>LOCAL_MBCS (&quot;nlsname&quot;)</td>
<td>This is an optional parameter. The default value is &quot;japanese_japan.jal6dbcs&quot;. It is an Oracle NLS name in the language_territory.charset format. It specifies the Oracle NLS name in which the local host data for all multi-byte character set fields in the transaction are encoded. Refer to Appendix H, &quot;National Language Support&quot; for more information.</td>
</tr>
</tbody>
</table>
DESCRIBE

Purpose
Use this command to describe a table, view, stored procedure, or function. If neither TABLE, VIEW, nor PROCEDURE are explicitly specified, the table or view with the specified name is described.

Syntax
The DESCRIBE command has the following syntax:

DESCRIBE [TABLE table|VIEW view|PROCEDURE proc|some_name]

Parameters
The following table describes the parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE</td>
<td>is the tablename</td>
</tr>
<tr>
<td>VIEW</td>
<td>is the viewname</td>
</tr>
<tr>
<td>PROC</td>
<td>is the procedurename</td>
</tr>
</tbody>
</table>

Examples
 DESCRIPTURE PROCEDURE SCOTT.ADDEMP
 DESCRIPTURE SYS.DUAL
 DESCRIPTURE TABLE SCOTT.PERSONNEL
 DESCRIPTURE VIEW SCOTT.PVIEW

DESCRIBE Usage Notes
- You do not need to place “;” at the end of the command.
DISCONNECT

Purpose
Use this command to disconnect from an Oracle database.

Syntax
The DISCONNECT command has the following syntax:
DISCONNECT

Parameters
None

Examples
None

DISCONNECT Usage Notes
- You do not need to place ";" at the end of the command.
EXECUTE

Purpose
Use this command to execute a one-line PL/SQL statement.

Syntax
The EXECUTE command has the following syntax:

```
EXECUTE pl/sql block
```

Parameters


Examples

```
EXECUTE :balance := get_balance(333)
```

EXECUTE Usage Notes

- You do not need to place ";" at the end of the command.
EXIT

**Purpose**
Use this command to terminate PGAU.

**Syntax**
The EXIT command has the syntax:

```
EXIT
```

**Parameters**
None

**Examples**
None

**EXIT Usage Notes**
- You do not need to place ";" at the end of the command.
- The quit command is not a valid statement in PGAU.
GENERATE

Purpose
A PL/SQL package is built and written to the indicated output files. The PG Data Dictionary is not updated by this command.

Syntax
GENERATE tname
    [VERSion(tranvers)]
    [PKGNAME(pname)]
    [PGANODE(dblink_name)]
    [OUTFILE("[specpath][specname][.{spectype}]")]
        {"[bodypath][bodyname][.{bodytype}]"
        [PKGEX([DC],[DR])]]

Parameters
The following table describes the GENERATE parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tname</td>
<td>is a mandatory parameter. It is the transaction name defined in a DEFINE TRANSACTION statement.</td>
</tr>
<tr>
<td>VERSion(tranvers)</td>
<td>is an optional parameter. It specifies which transaction definition is to be used. The VERsion parameter defaults to highest numbered transaction if not specified.</td>
</tr>
<tr>
<td>PKGNAME(pname)</td>
<td>is an optional parameter. It specifies the name of the PL/SQL package to be created. If this operand is omitted, the package name is assumed to be the same as the transaction name.</td>
</tr>
<tr>
<td>PGANODE(dblink_name)</td>
<td>is an optional parameter. It specifies the Oracle database link name to the gateway server. If this operand is omitted, &quot;PGA&quot; is assumed to be the dblink_name.</td>
</tr>
<tr>
<td>OUTFILE</td>
<td>is an optional parameter. If this parameter is specified, specname must also be specified.</td>
</tr>
</tbody>
</table>
**Parameter** | **Description**  
--- | ---  
`specpath` | is the optional directory path of the TIP specification and the TIP content documentation. It defaults to the current directory. The value must end with a forward slash (/).  
`specname` | is the filename of the TIP specification and the TIP content documentation. It defaults to `pname`, if specified, or else `pgau`.  
`spectype` | is the optional file extension of the TIP specification and defaults to `pkh`.  
`bodypath` | is the optional directory path of the TIP body. It defaults to `specpath`, if specified, or else the current directory. The value must end with a forward slash (/).  
`bodyname` | is the optional file name of the TIP body. It defaults to `specname`, if specified, or else `pname`, if specified, or else `pgau`. If `bodyname` defaults to `specname`, the leftmost period of `specname` is used to extract `bodyname` when `specname` contains multiple qualifiers.  
`bodytype` | is the optional file extension of the TIP body and defaults to `pkb`.  

The TIP Content output path defaults to `specpath` or else the current directory. The file id defaults to `specname`, if specified, or else `pname`, if specified, or else `pgau`, and always has an extension of `doc`.  

Refer to the "GENERATE Usage Notes:" on page 2-28 for more examples, and Appendix B, "TIP Internals" for more information.  

**DIAGNOSE** | is an optional parameter with two options, TRACE and PKGEX.
TRACE specifies that an internal trace of the execution of PGAU is written to output file pgau.trc in the user’s current directory.

TRACE suboptions are delimited by commas.

Trace messages are provided as a diagnostic tool to Oracle Support Services and other Oracle representatives to assist them in diagnosing customer problems when generating TIPs. They are part of an Oracle reserved function for which the usage, interface, and documentation might change without notice at Oracle’s sole discretion. This information is provided so customers might document problem symptoms.

- SE - Subroutine Entry/Exit
  Messages are written tracing subroutine name and arguments upon entry, and subroutine name and conditions at exit.

- IT - Initialization/Termination
  Messages are written tracing PGAU initialization and termination functions.

- QM - Queue Management
  Messages are written tracing control block allocation, queuing, searching, dequeuing, and deletion.

- IO - Input/Output
  Messages are written tracing input, output, and control operations for .dat input files and .wrk and package output files.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRACE</td>
<td>specifies that an internal trace of the execution of PGAU is written to output file pgau.trc in the user’s current directory. TRACE suboptions are delimited by commas. Trace messages are provided as a diagnostic tool to Oracle Support Services and other Oracle representatives to assist them in diagnosing customer problems when generating TIPs. They are part of an Oracle reserved function for which the usage, interface, and documentation might change without notice at Oracle’s sole discretion. This information is provided so customers might document problem symptoms.</td>
</tr>
<tr>
<td>SE</td>
<td>Subroutine Entry/Exit</td>
</tr>
<tr>
<td>IT</td>
<td>Initialization/Termination</td>
</tr>
<tr>
<td>QM</td>
<td>Queue Management</td>
</tr>
<tr>
<td>IO</td>
<td>Input/Output</td>
</tr>
</tbody>
</table>
DD - PG DD Definitions
Messages are written tracing the loading of transaction, call, data parameter, field, attribute, environment and compiler information from the PG DD.

OC - Oracle Calls
Messages are written tracing the Oracle UPI call results for SQL statement processing and SELECTs from the PG DD.

TG - TIP Generation
Messages are written tracing steps completed in TIP Generation, typically a record for each call, parameter, and data field for which a PL/SQL code segment has been generated.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DD</td>
<td>DD - PG DD Definitions</td>
</tr>
<tr>
<td>OC</td>
<td>OC - Oracle Calls</td>
</tr>
<tr>
<td>TG</td>
<td>TG - TIP Generation</td>
</tr>
</tbody>
</table>
PKGEX causes additional TIP execution time diagnostic logic to be included within the generated PL/SQL package.

PKGEX suboptions are delimited by commas.

- DC - Data Conversion

Enables runtime checking of repeating group limits and the raising of exceptions when such limits are exceeded.

Enables warning messages to be passed from the UTL_PG data conversion functions:

- NUMBER_TO_RAW
- RAW_TO_NUMBER
- MAKE_NUMBER_TO_RAW_FORMAT
- MAKE_RAW_TO_NUMBER_FORMAT

The additional logic checks for the existence of warnings and, if present, causes them to be displayed using DBMS_OUTPUT calls.

The TIP generation default is to suppress such warnings on the presumption that a TIP has been tested with production data and that data conversion anomalies either do not exist, or are known and to be ignored.

If errors occur which might be due to data conversion problems, regeneration of the TIP with PKGEX(DC) enabled might provide additional information.

**Note:** A runtime switch is also required to execute the warning logic. PKGEX(DC) only causes the warning logic to be included in the TIP. Refer to "Controlling TIP Runtime Conversion Warnings" on page 6-11.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PKGEX</td>
<td>causes additional TIP execution time diagnostic logic to be included within the generated PL/SQL package. PKGEX suboptions are delimited by commas. DC - Data Conversion Enables runtime checking of repeating group limits and the raising of exceptions when such limits are exceeded. Enables warning messages to be passed from the UTL_PG data conversion functions: NUMBER_TO_RAW RAW_TO_NUMBER MAKE_NUMBER_TO_RAW_FORMAT MAKE_RAW_TO_NUMBER_FORMAT The additional logic checks for the existence of warnings and, if present, causes them to be displayed using DBMS_OUTPUT calls. The TIP generation default is to suppress such warnings on the presumption that a TIP has been tested with production data and that data conversion anomalies either do not exist, or are known and to be ignored. If errors occur which might be due to data conversion problems, regeneration of the TIP with PKGEX(DC) enabled might provide additional information. <strong>Note:</strong> A runtime switch is also required to execute the warning logic. PKGEX(DC) only causes the warning logic to be included in the TIP. Refer to &quot;Controlling TIP Runtime Conversion Warnings&quot; on page 6-11.</td>
</tr>
</tbody>
</table>
Examples

Refer to "Sample PGAU GENERATE Statement" in Appendix E for examples of GENERATE commands.

The following list describes the TIP output file ids resulting from various combinations of GENERATE parameters:

```
GENERATE tran
   GENERATE tran OUTFILE("dirpath/")
   dirpath/pgau.pkh - TIP specification
   dirpath/pgau.doc - TIP Content
   dirpath/pgau.pkb - TIP body
GENERATE tran PKNAME tipname OUTFILE("dirpath/")
   dirpath/tipname.pkh - TIP specification
   dirpath/tipname.doc - TIP Content
   dirpath/tipname.pkb - TIP body
GENERATE tran PKNAME tipname OUTFILE("dirpath/",
   "bdir/path/"
   sdirpath/pgau.pkh - TIP specification
   sdirpath/pgau.doc - TIP Content
   bdir/path/pgau.pkb - TIP body
GENERATE tran PKNAME tipname OUTFILE("spath/sname","bpath/bname")
   spath/sname.pkh - TIP specification
   spath/sname.doc - TIP Content
   bpath/bname.pkb - TIP body
```

### Additional messages are written to a named pipe for tracing the data conversion steps performed by the TIP as it executes.

This option only causes the trace logic to be generated in the TIP. It must be enabled when the TIP is initialized. Refer to "Controlling TIP Runtime Conversion Warnings" on page 6-11 for more information.

- DR - Dictionary Reference

PL/SQL single line Comments are included in TIPs which reference the PG DD id numbers for the definitions causing the TIP function calls and conversions.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additional messages are written to a named pipe for tracing the data conversion steps performed by the TIP as it executes.</td>
<td></td>
</tr>
<tr>
<td>This option only causes the trace logic to be generated in the TIP. It must be enabled when the TIP is initialized. Refer to &quot;Controlling TIP Runtime Conversion Warnings&quot; on page 6-11 for more information.</td>
<td></td>
</tr>
<tr>
<td>DR - Dictionary Reference</td>
<td></td>
</tr>
<tr>
<td>PL/SQL single line Comments are included in TIPs which reference the PG DD id numbers for the definitions causing the TIP function calls and conversions.</td>
<td></td>
</tr>
</tbody>
</table>
PGAU Commands

GENERATE tran PKGNAME tipname
  tipname.pkh - TIP specification
  tipname.doc - TIP Content
  tipname.pkb - TIP body
GENERATE tran PKGNAME tipname OUTFILE("fileid")
  fileid.pkh - TIP specification
  fileid.doc - TIP Content
  fileid.pkb - TIP body
GENERATE tran PKGNAME tipname OUTFILE("specname","bodyname")
  specname.pkh - TIP specification
  specname.doc - TIP Content
  bodyname.pkb - TIP body
GENERATE tran PKGNAME tipname OUTFILE("specname.spectype")
  specname.spectype - TIP specification
  specname.doc - TIP Content
  specname.pkb - TIP body
GENERATE tran PKGNAME tipname OUTFILE("sname.stype","bname")
  sname.stype - TIP specification
  sname.doc - TIP Content
  bname.pkb - TIP body
GENERATE tran PKGNAME tipname
  OUTFILE("spath/sname.sext","bpath/bname.bext")
  spath/sname.sext - TIP specification
  spath/sname.doc - TIP Content
  bpath/bname.bext - TIP body

GENERATE Usage Notes:

■ All PGAU GENERATE trace messages are designated PGU-39nnn. Refer to the
  $ORACLE_HOME/pg4appc/mesg/pguus.msg file for further information on
  any given trace message.

■ The pgau.trc trace message output file is overwritten by the next invocation of
  GENERATE, regardless of the TRACE specification. A trace header record is
  always written to the pgau.trc file. If a particular trace file is to be saved, it must
  be copied to another file before the next invocation of GENERATE.

■ TRACE options can be specified in any order or combination, and can also be
  specified with PKGEX operand on the same GENERATE statement.
You must place ";" at the end of the command.

GROUP

Purpose
Multiple PGAU commands can be grouped together for purposes of updating the PG DD, and for rolling back all changes resulting from the commands in the group, if any one statement fails.

No COMMIT processing is performed, even if all commands within the group succeed. You perform the COMMIT either by coding COMMIT commands in the PGAU script, outside of GROUPs, or by issuing COMMIT interactively to PGAU.

PGAU issues a savepoint ROLLBACK to conditions before processing the group if any statement within the group fails.

Syntax
GROUP (pgaustmt1; pgaustmt2; ... pgaustmtN);

Parameters
pgaustmtN: is a PGAU DEFINE, REDEFINE, or UNDEFINE statement.

Examples
GROUP
  DEFINE DATA EMPNO
  PLSIDNAME (EMPNO)
  USAGE (PASS)
  LANGUAGE (IBMVSCOBOLII)
  (01 EMP-NO PIC X(6)).
);

DEFINE CALL DB2IMAIN
  PKGCALL (PGADB2I_MAIN)
  PARMS ( (EMPNO IN ),
            (EMPREC OUT) );

DEFINE TRANSACTION DB2I
  CALL ( DB2IMAIN,
         DB2IDIAG )
GROUP Usage Notes:

- No non-PGAU commands, such as ORACLE or SQL, can be placed inside the parentheses delimiting the group.

- A PGAU script can contain multiple GROUPs. Each GROUP can be interspersed with SQL commands, such as COMMIT or SELECT or with PGAU commands, such as GENERATE or REPORT.

- The first failing PGAU statement within the group causes a savepoint ROLLBACK to conditions at the beginning of the group. All subsequent commands within the group are flushed and not examined. PGAU execution resumes with the statement following the group. If that statement is a COMMIT, all PG DD changes made before the failing group are committed.

- You must place ";" at the end of the command.
HOST

Purpose
Use this command to execute an operating system command without exiting PGAU.

Syntax
The HOST command has the syntax:

```
HOST host_command
```

Parameters

- `host_command`: any valid UNIX command.

Examples

```
HOST vi log.out
HOST ls -la
HOST pwd
```

HOST Usage Notes
- Using the HOST command starts a new command shell under which to execute the specified operating system command. This means that any environment changes caused by the executed command affect only the new command shell started by PGAU, and not the command shell under which PGAU itself is executing. For example, a "cd" command executed by the HOST command does not change the current directory in the PGAU execution environment.
- You do not need to place ";" at the end of the command.
PRINT

Purpose
Use this command to print the value of a variable defined with the VARIABLE command.

Syntax
The PRINT command has the syntax:

\texttt{PRINT \textit{varname}}

Parameters
\textit{varname} is a variable name which is defined by a variable command

Examples
\begin{itemize}
  \item \texttt{PRINT ename}
  \item \texttt{PRINT balance}
\end{itemize}

PRINT Usage Notes
\begin{itemize}
  \item You do not need to place ";" at the end of the command.
\end{itemize}
REDEFINE DATA

Purpose
The existing data definition in the PG Data Dictionary is modified. PG DD column values for DATA#, FLD#, and POS remain the same for redefined data items. This permits existing CALL and DATA definitions to utilize the redefined data. REDEFINE does not create a different version of a data definition and the version number is not updated.

Syntax
REDEFINE DATA dname
  [VERSion(datavers)]
  [PLSDNAME(plsdvar)]
  [FIELD(fname) [PLSFNAME(plsfvar)]]
  [USAGE({PASS|ASIS|SKIP})]
  [COMPOPTS ('options')]
  [REMOTE_LANGUAGE("nlsname")]
  [LOCAL_LANGUAGE("nlsname")]
  LANGUAGE(language)
  <(definition) | INFILE("filespec")>;

Parameters
The following table describes the REDEFINE DATA parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA dname</td>
<td>is a mandatory parameter. It is the name of the data definition to be modified.</td>
</tr>
<tr>
<td>VERSion(datavers)</td>
<td>is an optional parameter. It specifies which version of dname is to be modified, and if specified, the updated dname information retains the same version number; a new version is not created. It defaults to the highest version if omitted.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PLSDNAME <em>(plsdvar)</em></td>
<td>is an optional parameter. It is the name of the PL/SQL variable associated with the <code>dname</code> above. It becomes the name of a PL/SQL variable if the <code>dname</code> item is atomic data, or a PL/SQL record variable if the <code>dname</code> item is aggregate data (such as a record or structure), when the TIP is generated. This name replaces any <code>plsdvar</code> name previously specified by DEFINE DATA into <code>pga_data(plsdvar)</code> of the PG DD.</td>
</tr>
<tr>
<td>FIELD <em>(fname)</em></td>
<td>is an optional parameter. It is the name of a field or group within the <code>dname</code> item, if aggregate data is being redefined (such as changing a field within a record).</td>
</tr>
<tr>
<td>PLSFNAME <em>(plsfvar)</em></td>
<td>is an optional parameter if FIELD is specified. It is the name of the PL/SQL variable associated with the <code>fname</code> above. It becomes the name of a PL/SQL field variable within a PL/SQL record variable when the TIP is generated. This name replaces any <code>plsfvar</code> name previously specified by REDEFINE DATA into <code>pga_data(plsfvar)</code> of the PG DD.</td>
</tr>
</tbody>
</table>
| USAGE({PASS|ASIS | SKIP}) | is optional. If omitted, the last usage specified is retained. It specifies the way the TIP handles the data items when exchanged in calls with the remote transaction:
  - PASS indicates that the item should be translated and exchanged with the transaction.
  - ASIS indicates the item is binary and, though exchanged, should not be translated.
  - SKIP indicates the item should be deleted from all exchanges.

If specified, all affected fields are updated with the same USAGE value. (Refer to the notes pertaining to single or multiple field redefinition, under FIELD).
### PGAU Commands

**Parameter** | **Description**  
--- | ---  
COMPOPTS ('options')  | is optional. If omitted, the last options specified are retained. If specified as a null string ('') then the last options specified are removed. If a non-null value is specified, then the last options specified are all replaced with the new options. The only option currently supported is 'TRUNC(BIN)'. Note that the options must be enclosed in apostrophes (') or quotes (''). TRUNC(BIN) is an IBM VS COBOL II option that affects the way halfword and fullword binary values are handled. Refer to "REDEFINE DATA Usage Notes:" on page 2-36 for further information on this option.  
REMOTE_LANGUAGE ("nlsname")  | is an optional parameter. The default value is "american_america.w8ebcdic37c" or as overridden by the NLS_LANGUAGE parameter of DEFINE TRANSACTION. It is an Oracle NLS name in the language_territory.charset format. It specifies the Oracle NLS name in which the remote host data for the specific character field being redefined is encoded. The field can be single byte or multi-byte character data. Refer to Appendix H, "National Language Support" for more information.  
LOCAL_LANGUAGE ("nlsname")  | is an optional parameter. The default value is initialized from the LANGUAGE variable of the local Oracle server when the TIP executes. It is an Oracle NLS name in the language_territory.charset format. It specifies the Oracle NLS name in which the local Oracle data for the specific character field being redefined is encoded. The field can be single byte or multi-byte character data. Refer to Appendix H, "National Language Support" for more information.  
LANGUAGE ("language")  | is a mandatory parameter if definition input is specified. It specifies the name of the programming language in the supplied definition. PGAU presently supports only IBM VSCOBOL II.
Examples
Refer to "Sample PGAU REDEFINE DATA Statements" in Appendix E for examples of REDEFINE commands.

REDEFINE DATA Usage Notes:
- Specification of either PLSDNAME, FIELD, or PLSFNAME allows redefinition of a single data item’s names while the (definition) parameter redefines the named data item’s content.
- The presence of FIELD denotes only a single data field (single PG DD row uniquely identified by dname, fname, and version) is updated. The absence
of FIELD denotes that multiple data fields (multiple PG DD rows identified by dname and version) are updated or replaced by the definition input.

- REMOTE_LANGUAGE and LOCAL_LANGUAGE override the character sets used for conversion of any individual SBCS, DBCS, or MBCS character data field.

- LANGUAGE (language) and (definition) | INFILE ("filespec") are mandatory as a group. If data definitions are to be supplied, then a LANGUAGE parameter must be specified and then either the inline definition or INFILE must also be specified.

- The presence of (definition) | INFILE ("filespec") denotes that multiple data fields (those PG DD rows identified by dname and version) are updated or replaced by the definition input. Fewer, equal, or greater numbers of fields might result from the replacement.

- INFILE ("filespec") is a platform-specific designation of a disk file.

- COMPOPTS ('TRUNC(BIN)') should be used only when the remote host transaction was compiled using IBM VS COBOL II with the TRUNC(BIN) compiler option specified. When this option is used, binary data items defined as PIC 9(4) or PIC S9(4) can actually contain values with 5 digits, and binary data items defined as PIC 9(9) or PIC S9(9) can actually contain values with 10 digits. Without COMPOPTS ('TRUNC(BIN)'), PGAU generates NUMBER(4,0) or NUMBER(9,0) fields for these data items, resulting in possible truncation of the values. When COMPOPTS ('TRUNC(BIN)') is specified, PGAU generates NUMBER(5,0) or NUMBER(10, 0) fields for these data items, avoiding any truncation of the values. Care must be taken when writing the client application to ensure that invalid values are not sent to the remote host transaction. For a PIC 9(4) the value must be within the range 0 to 32767, for a PIC S9(4) the value must be within the range -32767 to +32767, for a PIC 9(9) the value must be within the range -2,147,483,647 to +2,147,483,647, and for a PIC S9(9) the value must be within the range -2,147,483,647 to +2,147,483,647. VS COBOL II always reserves the high-order bit of binary fields for a sign, so the value ranges for unsigned fields are limited to the absolute values of the value ranges for signed fields. For further information, refer to the IBM manuals VS COBOL II Application Programming Guide, and VS COBOL II Application Programming Language Reference.

- Refer to "Alphanumeric and DBCS Editing Field Positions" in Appendix G, "Datatype Conversions" for information about how PGAU converts COBOL statements.

- You must place ";" at the end of the command.
REM

**Purpose**
Comments can either be introduced by the REM command or started with the two-character sequence `/*` and terminated with the two-character sequence `*/`.

Use the REM command to start a Comment line.

**Syntax**
The REM command has the syntax:

```
REM Comment
```

**Parameters**

- `Comment` any strings

**Examples**
REM This is a Comment....

**REM Usage Notes**
- You do not need to place `;` at the end of the command.
REPORT

Purpose
This command produces a report of selected data from the PG Data Dictionary. Selection criteria might determine that:

* a single TRANSACTION, CALL, or DATA entity (with or without an explicit version) is reported, or
* that all TRANSACTION, CALL, or DATA entities with a given name be reported or that all entities in the PG DD be reported, or
* that all invalid TRANSACTIONs or CALLs and all unreferenced CALLs, or DATA entities be reported.

Syntax
REPORT { { TRANSACTION tname | CALL cname | DATA dname } [VERSION(ver1...)] |
ALL { TRANSACTIONS [tname] | CALLS [cname] | DATA [dname] } |
{ WITH { CALLS | DATA | DEBUG } ... } |
ISOLATED; }

Parameters
The following table describes the REPORT parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSACTION tname</td>
<td>Reports the PG DD contents for the latest or selected versions of the transaction tname.</td>
</tr>
<tr>
<td>CALL cname</td>
<td>Reports the PG DD contents for the latest or selected versions of the call cname.</td>
</tr>
<tr>
<td>DATA dname</td>
<td>Reports the PG DD contents for the latest or selected versions of the data dname.</td>
</tr>
<tr>
<td>VERSION(ver1, [ver2 ...])</td>
<td>Reports selected versions of the indicated entry and is mutually exclusive with ALL.</td>
</tr>
<tr>
<td>ALL TRANSACTIONS [tname]</td>
<td>Reports the PG DD contents for all existing versions of every transaction entry or optionally a specific transaction tname, and is mutually exclusive with TRANSACTION.</td>
</tr>
</tbody>
</table>
**Examples**

Refer to “Sample PGAU REPORT Output” in Appendix E for sample REPORT commands.

**REPORT Usage Notes:**

- Report output is to the terminal and can be spooled, saved, and printed.
- Data reports are formatted according to their original compiler language, and preceded by a PGAU DEFINE DATA command which defines the data to the PG DD.
Call and Transaction reports are formatted as PGAU DEFINE CALL or TRANSACTION commands (also called "statements"), which effectively define the entry to the PG DD.

The following command reports the single most recent data definition specified by data name `dname`, or optionally, for those specific versions given.

```
REPORT DATA dname;
REPORT DATA dname VERSION(version#1,version#2);
```

This command reports all data definitions specified by data name `dname`:

```
REPORT ALL DATA dname;
```

The following command reports the single most recent call definitions specified by call name `cname`, or optionally for those specific versions given.

```
REPORT CALL cname;
REPORT CALL cname VERSION(version#1,version#2) WITH DATA;
```

This command reports all call definitions specified by call name `cname`:

```
REPORT ALL CALLS cname WITH DATA;
```

This command reports all call definitions in the PG DD:

```
REPORT ALL CALLS WITH DATA;
```

When WITH DATA is specified, all the data definitions associated with each selected call are also reported. The data definitions precede each corresponding selected call in the report output.

The following command reports the single most recent transaction definitions specified by transaction name `tname`, or optionally for those specific versions given.

```
REPORT TRANSACTION tname
REPORT TRANSACTION tname VERSION(version#1,version#2) WITH DATA WITH CALLS;
```

This command reports all transaction definitions specified by transaction name `tname`:

```
REPORT ALL TRANSACTIONS tname WITH DATA WITH CALLS;
```

This command reports all transaction definitions in the PG DD:

```
REPORT ALL TRANSACTIONS WITH DATA WITH CALLS;
```
When WITH CALLS option is specified, all call definitions associated with each selected transaction are also reported (the call definitions precede each corresponding selected transaction in the report output).

When WITH DATA is specified, all the data definitions associated with each selected call are also reported (the data definitions precede each corresponding selected call in the report output).

For transaction reports, specification of WITH DATA implies specification of WITH CALL.

- The following command reports any unreferenced CALL or DATA definitions. It also reports any TRANSACTION or CALL definitions that reference missing CALL or DATA definitions respectively.

```
REPORT ISOLATED;
```

- The following command reports all definitions in the PG DD.

```
REPORT ALL;
```

Data definitions are reported, followed by their associated call definitions, followed by the associated transaction definition.

This sequence is repeated for every defined call and transaction in the PG DD.

- You must place ";" at the end of the command.

## SET

### Parameters

The following table describes the SET parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARRAYSIZE [n]</td>
<td>Sets the number of rows fetched at a time from the database. The default is 20.</td>
</tr>
<tr>
<td>CHARWIDTH [n]</td>
<td>Sets the column display width for CHAR data. If entered with no argument, it returns the setting to 9, which is the default.</td>
</tr>
<tr>
<td>DATEWIDTH</td>
<td>Sets the column display width for DATE data. If entered with no argument, it returns the setting to 9, which is the default.</td>
</tr>
</tbody>
</table>
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECHO [ON</td>
<td>OFF]</td>
</tr>
<tr>
<td>FETCHROWS [n]</td>
<td>Sets the number of rows returned by a query. This is useful with ordered queries for finding a certain number of items in a category, the top ten items for example. It is also useful with unordered queries for finding the first n records that satisfy a certain criteria.</td>
</tr>
<tr>
<td>LONGWIDTH [n]</td>
<td>Sets the column display width for LONG data. If entered with no argument, it returns the setting to 80, which is the default.</td>
</tr>
<tr>
<td>MAXDATA [n]</td>
<td>Sets the maximum data size. It indicates the maximum data that can be received in a single fetch during a SELECT command. The default is 20480 bytes (20K).</td>
</tr>
<tr>
<td>NUMWIDTH [n]</td>
<td>Sets the column display width for NUMBER data. If entered with no argument, it returns the setting to 10, which is the default.</td>
</tr>
<tr>
<td>SERVEROUTPUT {OFF</td>
<td>ON [SIZE n</td>
</tr>
<tr>
<td>STOPONERROR [ON</td>
<td>OFF]</td>
</tr>
</tbody>
</table>
Examples

PGAU> set arraysize 30

PGAU> set CHARWIDTH

SET Usage Notes

- You do not need to place ";" at the end of the command.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TERMOOUT ON</td>
<td>OFF</td>
</tr>
<tr>
<td>TIMING ON</td>
<td>OFF</td>
</tr>
</tbody>
</table>
SHOW

Parameters
The following table describes the SHOW parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>Shows all valid SET parameters</td>
</tr>
<tr>
<td>ARRAYSIZE</td>
<td>Shows the number of rows fetched at a time from the database.</td>
</tr>
<tr>
<td>CHARWIDTH</td>
<td>Shows the column display width for CHAR data.</td>
</tr>
<tr>
<td>DATEWIDTH</td>
<td>Shows the column display width for DATE data.</td>
</tr>
<tr>
<td>ECHO</td>
<td>Shows echoing of commands entered from command files to ON or OFF.</td>
</tr>
<tr>
<td>FETCHROWS</td>
<td>Shows the number of rows returned by a query.</td>
</tr>
<tr>
<td>LONGWIDTH</td>
<td>Shows the column display width for LONG data.</td>
</tr>
<tr>
<td>MAXDATA</td>
<td>Shows the maximum data size.</td>
</tr>
<tr>
<td>NUMWIDTH</td>
<td>Shows the column display width for NUMBER data.</td>
</tr>
<tr>
<td>SERVEROUTPUT</td>
<td>Shows debugging output from stored procedures that use DBMS_OUTPUT PUT and PUT_LINE commands.</td>
</tr>
<tr>
<td>STOPONERROR</td>
<td>Indicates whether execution of a command file should stop if an error occurs.</td>
</tr>
<tr>
<td>TERMOUT</td>
<td>Shows whether the terminal output for SQL commands is enabled or disabled.</td>
</tr>
<tr>
<td>TIMING</td>
<td>Shows whether display of parse, execute, and fetch times (both CPU and elapsed) for each executed SQL statement is enabled or disabled.</td>
</tr>
<tr>
<td>VAR</td>
<td>Is the same as the PRINT command; in addition, it shows all variables and their datatypes.</td>
</tr>
</tbody>
</table>
Examples

Note that when you issue a SET command, there will be no output if it is successful. If you want to check whether your statement was executed successfully, issue a SHOW command like the following:

```
PGAU> show arraysize
Arraysize 30

PGAU> show CHARWIDTH
Charwidth 80

PGAU> show all
Instance local
Spool OFF
Timing OFF
Termout ON
Echo OFF
Stoponerror OFF
Maxdata 20480
Arraysize 20
Fetchrows 100

Numwidth 10
Charwidth 80
Longwidth 80
Datewidth 9
ServerOutput OFF
```

SHOW Usage Notes

- You do not need to place ";" at the end of the command.
**SPOOL**

**Purpose**
Use this command to specify a filename that captures PGAU output. All output is directed to the terminal unless TERMOUT is off.

**Syntax**
The SPOOL command has the syntax:

SPOOL [filename|OFF]

**Parameters**
If a simple filename is specified, with no periods, then .log is appended to the filename.

filename where the output of your executed commands is placed

**Examples**
SPOOL log.outfile
SPOOL out
SPOOL OFF

**SPOOL Usage Notes**
- You do not need to place ";" at the end of the command.
UNDEFINE CALL

Purpose
Use this command to remove an occurrence of the CALL definition from PG DD.

Syntax
UNDEFINE CALL  cname   [VERSion(callvers|ALL)];

Parameters
The following table describes the UNDEFINE CALL parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL  cname</td>
<td>A mandatory parameter. It specifies the name associated with the item to be dropped; if no version is specified only the latest (highest numbered) version is removed.</td>
</tr>
<tr>
<td>VERSION({datavers</td>
<td>callvers</td>
</tr>
</tbody>
</table>

Examples
Refer to "Sample PGAU UNDEFINE Statements" in Appendix E for examples of UNDEFINE CALL commands.

UNDEFINE CALL Usage Notes:

- Removing definitions only prevents PL/SQL packages from being subsequently generated. TIPs can still be recreated if the .pkh and .pkb specification files exist and those previous TIPs can be invoked if they remain in the database of the Oracle integrating server. Whether such TIPs execute successfully depends on whether the corresponding remote transaction programs are still active.
- Remove a CALL definition only after all TRANSACTIONs which reference it are removed. No integrity checking is done.
- You must place ";" at the end of the command.
**UNDEFINE DATA**

**Purpose**
Use this command to remove an occurrence of the DATA definition in the PG Data Dictionary.

**Syntax**

```
UNDEFINE DATA dname [VERSion(datavers|ALL)];
```

**Parameters**

The following table describes the UNDEFINE DATA parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA dname</td>
<td>A mandatory parameter. It specifies the name associated with the item to be dropped. If no version is specified, only the latest (highest numbered) version is removed.</td>
</tr>
<tr>
<td>VERSion(datavers</td>
<td>callvers</td>
</tr>
</tbody>
</table>

**Examples**

Refer to *Sample PGAU UNDEFINE Statements* in Appendix E for examples of UNDEFINE DATA commands.

**UNDEFINE DATA Usage Notes**

- Removing definitions only prevents PL/SQL packages (TIPs) from being subsequently generated. Previously generated TIPs can still be recreated if the .pkh and .pkb specification files remain in existence. Previously created TIPs can still be invoked if they remain in the database of the Oracle integrating server. Whether such TIPs execute successfully depends on whether the corresponding remote transaction programs are still active.
- Remove a DATA definition only after all CALLs and all TRANSACTIONs which reference it are removed. No integrity checking is done.
You must place ";" at the end of the command.

**UNDEFINE TRANSACTION**

**Purpose**
This command removes an occurrence of the TRANSACTION definition in the PG Data Dictionary.

**Syntax**
UNDEFINE TRANSACTION 
\[\text{tname} \ [\text{VERSion} (=\text{tranvers}|\text{ALL})] ;\]

**Parameters**
The following table describes the UNDEFINE TRANSACTION parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSACTION (tname)</td>
<td>Mandatory parameter. It specifies the name associated with the item to be dropped. If no version is specified, only the latest (highest numbered) version is removed.</td>
</tr>
<tr>
<td>VERSion ({\text{datavers}</td>
<td>\text{callvers}</td>
</tr>
</tbody>
</table>

**Examples**
Refer to "Sample PGAU UNDEFINE Statements" in Appendix E for examples of UNDEFINE TRANSACTION commands.

**UNDEFINE TRANSACTION Usage Notes**
- Removing definitions only prevents PL/SQL packages from being subsequently generated. TIPs can still be recreated if the .pkh and .pkb specification files remain in existence. Previously created TIPs can be invoked if they remain in the database of the Oracle integrating server. Whether such TIPs execute successfully depends on whether the corresponding remote transaction programs are still active.
■ A TRANSACTION definition can be removed at any time.
■ You must place ";" at the end of the command.

VARIABLE

Purpose
Use this command to declare a bind variable for use in the current session with the EXECUTE or PRINT command, or for use with a PL/SQL block.

Syntax
The VARIABLE command has the syntax:

VARIABLE name type

Parameters
The following table describes the VARIABLE parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Is a variable name.</td>
</tr>
<tr>
<td>type</td>
<td>Is the variable datatype</td>
</tr>
</tbody>
</table>

Examples
VARIABLE balance NUMBER
VARIABLE emp_name VARCHAR2

VARIABLE Usage Notes
■ You do not need to place ";" at the end of the command.
This chapter shows in detail how you can define, generate and compile a Transaction Interface Package (TIP). It assumes that a remote host transaction program (RTP) already exists. This transaction program has operational characteristics that dictate how the TIP is defined and how the TIP is used by the client application.

This chapter contains the following sections:

- Granting Privileges for TIP Creators on page 3-2
- Evaluating the RHT on page 3-2
- Writing the PGAU Statements on page 3-5
- Defining and Generating the TIP on page 3-7
- Compiling the TIP on page 3-8
- TIP Content Documentation (tipname.doc) on page 3-9
The following steps create a TIP for use with a remote host transaction (RHT):

- evaluating the RHT
- preparing the PGAU statements
- defining and generating the TIP
- compiling the TIP

This chapter also discusses the generated TIP content file.

Granting Privileges for TIP Creators

Every TIP developer requires access to the following PL/SQL packages, which are shipped with the Oracle server:

- `DBMS_PIPE` in `$ORACLE_HOME/rdbms/admin`
- `UTL_RAW` in `$ORACLE_HOME/rdbms/admin`
- `UTL_PG` in `$ORACLE_HOME/rdbms/admin`

If anyone other than the PG ADMIN will be developing TIPs, they will need explicit grants to perform these operations. Refer to the Optional Configuration Steps section in Chapter 4 of the *Oracle Procedural Gateway for APPC Installation and Configuration Guide* for your platform for more information about private and public grants.

Evaluating the RHT

Follow the steps below to identify and become familiar with your remote host transaction data exchanges.

Identify the Remote Host Transaction

1. You must first identify the RHT data exchange steps. These are the APPC send and receive calls embedded within the RHT program. They are identified under the following languages:
   - You may use IBM VS/COBOL II for:
     * CICS
     * IMS
You may use IBM 370 Assembler for:
* CICS
* IDMS
* IMS

You may use IBM REXX for:
* CICS
* IDMS
* IMS
* MVS

**PGAU DEFINE CALL Command**

2. Make a list of every data exchange. This list dictates a series of PGAU DEFINE CALL statements. Refer to "DEFINE CALL" in Chapter 2, "Procedural Gateway Administration Utility" for more information about this PGAU command.

The three important parameters that you will use for each call are:
- **cname**: the name of the call definition to be created;
- **dname**: the name of the data structure to be exchanged; and
- whether it is send (**OUT**) or receive (**IN**)

RHT send corresponds to a TIP OUT and RHT receive corresponds to a TIP IN. Refer to "APPC Send/Receive Synchronization" on page B-7 for more information.

PGAU call entries are only defined once, so eliminate any duplicates.

This call list defines the TIP function calls, not the order in which they are used. Note that the order in which each call is made is a behavior of the transaction and dictates the order of calls made by the high-level application to the TIP, which then calls the RHT through the Procedural Gateway server. While this calling sequence is critical to maintaining the synchronization between the application and the RHT, the TIP is only an access method for the application and has no knowledge of higher level sequencing of calls.
Evaluating the RHT

PGAU DEFINE DATA Command

3. For each call in the RHT call list, identify the RHT data structures being sent or received in the call buffers.

   Make a data list of every such structure. This list dictates a series of PGAU DEFINE DATA statements.

   The two important parameters that you will use for DEFINE DATA are:
   ■ dname: the name of the data definition to be created; and
   ■ dname.ext: the file in which the data definition is stored.

   PGAU data entries are only defined once, so eliminate any duplicates.

   **Note:** Move COBOL record layouts (copybooks) to the gateway system.

   PGAU can use copybooks as input when defining the data items. Once you have identified the data items to be exchanged, use a file transfer program to download the copybooks to the gateway system. The copybooks are later used to define the data items. The sample copybook used in the example is documented in Appendix E, "Administration Utility Samples".

PGAU DEFINE TRANSACTION Command

4. Determine the network address information for the RHT program. Your network or OLTP system programmer can provide you with this information.

   The five important parameters that you will use for PGAU DEFINE TRANSACTION are:
   ■ Side Profile name
   ■ TP name
   ■ LU name
   ■ LOGMODE
   ■ SYNCLEVEL

   You must also identify the Oracle NLS character set (charset) for the language in which the OLTP expects the data.
Writing the PGAU Statements

After evaluating the RHT, define the TIP to PGAU for placement in the PG DD.

1. Write a DEFINE DATA statement for each entry in your data list. If, for example, your RHT had three different data structures, your data definitions might be:

```
DEFINE DATA dname1 LANGUAGE(IBMVCSCOBOLII) INFILE(dname1.ext);
DEFINE DATA dname2 LANGUAGE(IBMVCSCOBOLII) INFILE(dname2.ext);
DEFINE DATA dname3 LANGUAGE(IBMVCSCOBOLII) INFILE(dname3.ext);
```

Then you must copy or transfer the source file containing these data definitions to the directory where PGAU can read them as input.

2. Write a DEFINE CALL statement for each entry in your call list. If, for example, your RHT had a receive send receive send sequence, your call definitions would be:

```
DEFINE CALL cname1 PARMS((dname1 IN));
DEFINE CALL cname2 PARMS((dname2 OUT));
DEFINE CALL cname3 PARMS((dname3 IN));
DEFINE CALL cname4 PARMS((dname2 OUT));
```

**NOTE:** Optionally, you can rewrite your call definitions to consolidate the data transmission into fewer exchanges, as long as you do not alter the data transmission sequence. For example:

```
DEFINE CALL cname1 PARMS((dname1 IN),
                          (dname2 OUT));
```

```
DEFINE CALL cname3 PARMS((dname3 IN),
                          (dname2 OUT));
```

This reduces the calls between the application and the TIP from four calls to two calls passing an IN and OUT parameter on each call. Because TIPs always process IN parameters before OUT parameters, the data transmission sequence is unchanged. However, this consolidation is not always possible. Refer to "APPC Send/Receive Synchronization" on page B-7 for more information.
3. Write a DEFINE TRANSACTION statement that contains every call, specifying the network address and NLS information:

```
DEFINE TRANSACTION tname CALLS(cname1
  cname2, ....
  cnameN)
  ENVIRONMENT(IBM370)
  SIDEPROF(profname) |
  TPNAME(tpid) LUNAME(luname) LOGMODE(mode)
  SYNCELEVEL(n)
  NLS_LANGUAGE(charset);
```

4. You can add a GENERATE statement to create the TIP specification:

```
GENERATE tname
```

**Note:** You can also add a REPORT statement to list the PG DD entries for tname:

```
REPORT TRANSACTION tname with CALLS with DATA;
```

Also annotate the script with Comments:

```
# this is a Comment
```

### Writing a PGAU Script File

The previous section describes the three steps you need to follow in order to execute PGAU statements via your PGAU command line processor. As a time saving measure, you can choose to write all of the statements (DEFINE DATA, DEFINE CALL, AND DEFINE TRANSACTION) into a single PGAU script file named `tname.ctl`, in the following order:

- define data
- define call
- define transaction
- generate

**Caution:** Because you will probably run this script more than once, you should include UNDEFINE statements first to remove any previous entries in the PG DD.
This is an example of a tname.ctl PGAU script file:

```
UNDEFINE TRANSACTION tname Version(all);
UNDEFINE CALL cname1 Version(all);
UNDEFINE CALL cname2 Version(all);
UNDEFINE DATA dname1 Version(all);
UNDEFINE DATA dname2 Version(all);
UNDEFINE DATA dname3 Version(all);
DEFINE DATA dname1 LANGUAGE(IBMVSCOBOLII) INFILE(dnamel.ext);
DEFINE DATA dname2 LANGUAGE(IBMVSCOBOLII) INFILE(dname2.ext);
DEFINE DATA dname3 LANGUAGE(IBMVSCOBOLII) INFILE(dname3.ext);
DEFINE CALL cname1 PARMS(dname1 IN),
       (dname2 OUT));
DEFINE CALL cname2 PARMS(dname3 IN),
       (dname2 OUT));
DEFINE TRANSACTION tname CALLS(cname1,
       cname2, ....
       cnameN)
   ENVIRONMENT(IBM370)
   SIDEPROF(profname) |
       TPNAME(tpid) LUNAME(luname) LOGMODE(mode)
   SYNCLEVEL(n)
   NLS_LANGUAGE(charset);
```

Defining and Generating the TIP

After you have created your control file, use PGAU to create the PG DD entries and the TIP specification files.

---

**Note:** The user ID under which you run PGAU must have:

- write access to output the specification files (*pgau.pkh, pgau.pkb, and pgau.doc*), where `pgau` is the default name; and
- read access to the data definition source files (*dname.ext*), where `dname.ext` will be specified in PGAU DEFINE DATA statement(s).
Invoke PGAU against your PG DD stored in the Oracle Procedural Gateway for APPC Administrator’s user ID:

```
$ pgau
PGAU> connect pgaadmin/pw@database_specification_string
```

Issue the following commands:

```
$PGAU> set echo on
PGAU> spool tname.def
PGAU> @tname.ctl
PGAU> spool off
```

The TIP is now ready to be compiled. By default, the GENERATE statement writes your TIP specifications to the following output files in your current directory:

- `pgau.pkh` (TIP Header)
- `pgau.pkb` (TIP Body)
- `pgau.doc` (TIP content documentation)

**Note:** You can optionally add `spool` and `echo` to your script (`tname.ctl`) or make other enhancements, such as using PG DD roles and the PGAU GROUP statement for shared PG DDs. Refer to Chapter 4, "Client Application Development" for more information.

---

**Compiling the TIP**

Exit PGAU. Remain in your current directory and invoke SQL*Plus.

```
$ sqlplus userid/pw@database_specification_string
SQL> set echo on
SQL> @pgau.pkh
SQL> @pgau.pkb
```

The last two commands compile the TIP specification and body, respectively.

You have now compiled a TIP which can be called by your client application. If your client application is already written you can begin testing.

For more information about designing your client application and compiling a TIP, refer to Chapter 1, ”Introduction to Oracle Procedural Gateway for APPC” and Appendix B, ”TIP Internals” and refer to Chapter 4, ”Client Application Development” for information about PGAU statement syntax and usage.
This section discusses the TIP documentation file that is produced when the user issues a PGAU GENERATE command. This TIP content file describes the function calls and PL/SQL variables and datatypes available in the TIP.

PGAU GENERATE always produces a TIP content file named `tipname.doc`. The filename is the name of the transaction that was specified in the PGAU GENERATE command, and the filetype is always `.doc`. This TIP content file contains the following sections:

- **GENERATION Status**
  This section contains the status under which the TIP is generated.

- **TIP Transaction**
  This section identifies the defined transaction attributes. These result from the PGAU DEFINE TRANSACTION definition.

- **TIP Default Calls**
  This section identifies the syntax of the calls made by the user’s application to initialize and terminate the transaction. PGAU generates these calls into every TIP regardless of how the TIP or transaction is defined.

- **TIP User Calls**
  This section identifies the syntax of the calls which the user defines for the application to interact with the transaction.

- **TIP User Declarations**
  This section identifies the TIP package public datatype declarations, implied by the user’s data definition specified in each call parameter.

- **TIP User Variables**
  This section contains TIP variables that can be referred to by applications or referenced by applications.
This chapter discusses how you will call a TIP and control a remote host transaction. It also provides you with the steps for preparing and executing a gateway transaction. This chapter assumes:

- a remote host transaction (RHT) has already been written
- a TIP corresponding to the RHT has already been defined using the steps described in Chapter 3, “Creating a TIP”.

This chapter contains the following sections:

- Overview of Client Application on page 4-2
- Preparing the Client Application on page 4-3
- Client Application Requirements on page 4-9
- Ensuring TIP and Remote Transaction Program Correspondence on page 4-14
- Calling the TIP from the Client Application on page 4-20
- Executing the Application on page 4-28
- APPC Conversation Sharing on page 4-28
- Application Development with Multi-Byte Character Set Support on page 4-35
- Modifying a Terminal-Oriented Transaction to Use APPC on page 4-36
- Privileges Needed to Use TIPs on page 4-37
Overview of Client Application

The Procedural Gateway Administration Utility (PGAU) generates a complete TIP using definitions you provide. The client application can then call the TIP to access the remote host transaction. Chapter 2, "Procedural Gateway Administration Utility", discusses the use of PGAU in detail.

This overview explains what you must do in order to call a TIP and control a remote host transaction.

The gateway receives PL/SQL calls from the Oracle integrating server and issues APPC calls to communicate with a remote transaction program. The following three application programs make this possible:

- an APPC-enabled remote host transaction program
- a Transaction Interface Package, or TIP. A TIP is a PL/SQL package that handles communication between the client and the gateway and performs datatype conversions between COBOL and PL/SQL.

PGAU generates the TIP specification for you. In the shipped samples, the PGAU-generated package is called pgadb2i.pkb. This generated TIP includes at least three function calls that map to the remote transaction program:

- pgadb2i_init initializes the conversation with the remote transaction program
- pgadb2i_main exchanges application data with the remote transaction program
- pgadb2i_term terminates the conversation with the remote transaction program

Refer to Appendix B, "TIP Internals" for more information about TIPs, if you are writing your own TIP or debugging.

- a client application that calls the TIP

The client application calls the three TIP functions with input and output arguments. In the example, the client application passes empno, an employee number to the remote transaction and the remote transaction sends back emprec an employee record.
Preparing the Client Application

Table 4–1 demonstrates the logic flow between the PL/SQL driver, the TIP, and the gateway using the example CICS-DB2 transaction.

Table 4–1 Logic Flow of CICS-DB2 Example

<table>
<thead>
<tr>
<th>Client Application</th>
<th>Oracle TIP</th>
<th>Procedures Established Between the Gateway and the Remote Transaction (mainframe)</th>
</tr>
</thead>
<tbody>
<tr>
<td>calls tip_init</td>
<td>Calls PGAINIT</td>
<td>Gateway sets up control blocks and issues APPC ALLOCATE. Mainframe program initiates.</td>
</tr>
<tr>
<td>calls tip_main</td>
<td>Calls PGAXFER to send <code>empno</code> and receive <code>emprec</code></td>
<td>Gateway issues APPC SEND to the mainframe. Mainframe RECEIVE completes. Mainframe performs application logic and issues APPC SEND back to Gateway. The Gateway issues APPC RECEIVE; receive completes. Mainframe issues APPC TERM.</td>
</tr>
<tr>
<td>calls tip_term</td>
<td>Call PGATERM</td>
<td>Gateway cleans up control blocks.</td>
</tr>
</tbody>
</table>

A client application which utilizes the gateway to exchange data with a remote host transaction performs some tasks for itself and instructs the TIP to perform other tasks on its behalf. The client application designer must consequently know the behavior of the remote transaction and how the TIP facilitates the exchange.

The following sections provide an overview of remote host transaction behavior, how this behavior is controlled by the client application and how TIP function calls and data declarations support the client application to control the remote host transaction. These sections also provide background information about what the TIP does for the client application and how the TIP calls exchange data with the remote host transaction.

Preparing the Client Application

To prepare the client application for execution you must understand the remote host transaction requirements and then perform these steps:

1. Move relevant COBOL records layout (copybooks) to the gateway system for input to PGAU.

2. Describe the remote host transaction data and calls to the PG Data Dictionary (PG DD) with DEFINE DATA, DEFINE CALL, and DEFINE TRANSACTION statements.
Preparing the Client Application

3. Generate the TIP in the Oracle integrating server, using GENERATE.
4. Create the client application that calls the TIP public functions.
5. Grant privileges on the newly created package.

Understanding the Remote Host Transaction Requirements

Browse through the remote host transaction program (RTP) to determine:

- the PL/SQL parameters required on the various client application to TIP calls
- the order in which the calls are made

Identify the remote host transaction program (RTP) facilities to be called and the data to be exchanged on each call. You will then define the following, and store them in the PG DD:

- DEFINE DATA
- DEFINE CALL
- DEFINE TRANSACTION

Refer to Chapter 3, "Creating a TIP" for specific definition steps and for the actual creation and generation of a TIP.

TIP Content and Purpose

The content of a PGAU-generated TIP reflects the calls available to the remote host transaction and the data that has been exchanged. Understanding this content helps when designing and debugging client applications that call the TIP.

A TIP is a PL/SQL package, and accordingly has two sections:

1. A Package Specification containing:
   - Public function prototypes and parameters, and
2. A Package Body containing:
   - Private functions and internal control variables
   - Public functions
   - Package initialization following the last public function.

The purpose of the TIP is to provide a PL/SQL callable public function for every allowed remote transaction program interaction. A remote transaction program interaction is a logically related group of data exchanges through one or more
Preparing the Client Application

PGAXFER RPC calls. This is conceptually similar to a screen or menu interaction in which several fields are filled in, the enter key is pressed, and several fields are returned to the user. Carrying the analogy further:

- the user might be likened to the TIP or client application
- fields to be filled in are IN parameters on the TIP function call
- fields returned are OUT parameters on the TIP function call
- screen or menu is the group of IN and OUT parameters combined
- a pressed enter key is likened to the PGAXFER remote procedural call (RPC)

The actual grouping of parameters that constitute a transaction call is defined by the user. The gateway places no restrictions on how a remote transaction program might correspond to a collection of TIP function calls, each call having many IN and OUT parameters.

PGA users typically have one TIP per remote transaction program. How the TIP function calls are grouped and what data parameters are exchanged on each call depends on the size, complexity and behavior of the remote transaction program.

Refer to Oracle’s PL/SQL User’s Guide and Reference for a discussion of how PL/SQL packages work. The following discussion covers the logic that must be performed within a TIP. Refer to the sample TIP and driver supplied in the SORACLE_HOME/pg4appc/demo/CICS directory in files pgadb2i.pkh, pgadb2i.pkb, and pgadb2id.sql.

Remote Host Transaction Types

From a procedural gateway application perspective, there are three main types of remote host transactions:

- one-shot
- persistent
- multi-conversational

One-Shot Transactions

A simple remote transaction program which receives one employee number and returns the employee record could have a TIP which provides one call, passing the employee number as an IN parameter and returning the employee record as an
OUT parameter. An additional two function calls must be provided by this and every TIP:

- a remote transaction program init function call
- a remote transaction program terminate function call

The most simple TIP has three public functions, such as tip_init, tip_main, and tip_term.

The client application calls tip_init, tip_main, and tip_term in succession. The corresponding activity at the remote site is remote transaction program start, data exchange, and remote transaction program end.

The remote transaction program might even terminate itself before receiving a terminate signal from the gateway. This sequence is usual and is handled normally by gateway logic. This kind of remote transaction program is termed one-shot.

**Persistent Transactions**

A more complex remote transaction program has two modes of behavior: an INQUIRY or reporting mode, and an UPDATE mode. These modes can have two TIP data transfer function calls: one for INQUIRY and one for UPDATE. Such a TIP might have five public functions. For example:

- tip_init
  This initializes communications with the remote transaction program.

- tip_mode
  This accepts a mode selection parameter and puts the transaction program into either inquiry or update mode.

- tip_inqr
  This returns an employee record for a given employee number.

- tip_updt
  This accepts an employee record for a given employee number.

- tip_term
  This terminates communications with the remote transaction program.

The client application calls tip_init and then tip_mode to place the remote transaction program in inquiry mode which then scans employee records, searching for some combination of attributes (known to the client application and end-user). Some parameter on an inquiry call is then set to signal a change to update mode.
and the client application calls tip_updt to update some record. The client application finally calls tip_term to terminate the remote transaction program.

The corresponding activity at the remote site is:

- remote transaction program start
- mode selection exchange
- loop reading records
- switch to update mode
- update one record
- remote transaction program end

Such a remote transaction program is called persistent because it interacts until it is signalled to terminate.

The remote transaction program can be written to permit a return to inquiry mode and repeat the entire process indefinitely.

**Multi-Conversational Transactions**

A client application might need to get information from one transaction, tran_A, and subsequently write or lookup information from another, tran_B. This is possible with a properly written client application and TIPs for tran_A and tran_B. In fact, any number of transactions might be concurrently controlled by a single client application. All transactions could be read-only, with the client application retrieving data from each and consolidating it into a local Oracle database or displaying it in an Oracle Form.

Alternatively, a transaction could be capable of operating in different modes or performing different services depending on what input selections were supplied by the client application. For example, one instance of tran_C can perform one service while a second instance of tran_C performs a second service. Each instance of tran_C would have its own unique conversation with the client application and each instance could have its own behavior (one-shot or persistent) depending on the nature of the service being performed.

**NOTE for AIX users:** Oracle Procedural Gateway for APPC for AIX supports two-phase commit. Other platforms support only single-site, read-only, or commit-confirm.
Customized TIPs for Each Remote Host Transaction

Each remote system might have hundreds of remote transaction programs (RTPs) which a user might want to call. Each remote transaction program is different, passes different data, and performs different functions. The interface between the user and each remote transaction program must consequently be specialized and customized to the user’s requirements for each remote transaction program. The Transaction Interface Package (TIP) provides this customized interface.

Example

Assume that the remote site has a transaction program which manages employee information in an employee database or other file system. The remote transaction program’s name, in the remote system, is EMPT for EMPLOYEE Tracking. EMPT provides both inquiry and update facilities, and different Oracle users are required to access and use these EMPT facilities.

Some users might be restricted to inquiry-only use of EMPT, while others might have update requirements. In support of the Oracle users’ client applications, at least three possible TIPs could exist:

1. EMP_MGMT to provide access to all facilities of the EMPT remote transaction program.
2. EMP_UPDT to access only the update functions of the EMPT remote transaction program.
3. EMP_INQR to access only the lookup functions of the EMPT remote transaction program.

Note for non-AIX users: Only one gateway transaction can perform update functions while others are in read-only mode. This restriction exists because your gateway currently supports only single-site update or commit-confirm, and does not support two-phase commit.

Single-site update permits only one database or gateway to be in update mode. Commit-confirm permits multiple databases or gateways to be in update mode, but only one can be commit-confirm; all others must be two-phase commit capable.

With a commit-confirm gateway, the gateway transaction can participate in an Oracle distributed transaction.
End-user access to these TIPs is controlled by Oracle privileges. Additional security might be imposed on the end-user by the remote system.

Each TIP also has encoded within it the name of the remote transaction program (EMPT) and network information sufficient to establish an APPC conversation with EMPT.

Client Application Requirements

Using the TIP, the client application must correspond with and control the remote host transaction. This involves:

1. client application initialization
2. user input and output
3. remote host transaction initialization using the TIP initialization functions (with and without overrides)
4. remote host transaction control and data exchange using the TIP user functions
5. remote host transaction termination using the TIP termination function
6. exception handling
7. client application termination

Steps 3 through 5 vary, based on the requirements of the remote host transaction.

One-shot remote host transaction client applications must:

- Declare RHT/TIP datatypes to be exchanged. All client applications must declare variables to be exchanged with the RHTs using TIPs. PL/SQL datatypes for such variables have already been defined in the TIP corresponding to each RHT and the client application need only reference the TIP datatype in its declaration. Refer also to "Declaring TIP Variables" on page 4-20 for more information. Also refer to the TIP content documentation file for the specific TIP/RHT for more information about the exact usage of these variables.

- Initialize the RHT using the TIP initialization function. The TIP directs the gateway server to initialize a conversation with the desired RHT, specifying either default RHT identifying parameters (supplied when the RHT was defined in the PG DD and encoded within the TIP when it was generated) or override RHT identifying parameters supplied by the user or client application when the TIP initialization function is called. Refer to "Initializing the
Client Application Requirements

- Exchange data with the RHT using the TIP user function (one call). As previously discussed, a one-shot remote host transaction only accommodates a single data exchange and upon completion of that exchange, the RHT terminates on its own. The client application consequently needs only to execute a single call to the user-defined TIP function to cause the data exchange. Refer to the TIP content documentation file in \$ORACLE_HOME/pg4appc/demo/CICS/ for the specific TIP/RHT for the exact syntax of this call.

The client application should initialize values into IN or IN OUT parameter values before calling the TIP function call. These are the same variables that were declared above, when you declared the RHT/TIP datatypes to be exchanged.

All TIP function calls return a 0 return code value and all returned user gateway data values are exchanged in the function parameters. Any exception conditions are raised as required and can be intercepted in an exception handler.

Upon return from the TIP function call, the client application can analyze and operate on the IN OUT or OUT parameter values. These are the same variables that were declared above, when you declared the RHT/TIP datatypes to be exchanged.

Refer to Appendix G, "Datatype Conversions" for details about how TIPs convert the various types and formats of remote host data.

- Terminate the RHT using the TIP termination function. Regardless of the type of RHT being accessed, the TIP terminate function should be called to clean up and terminate the conversation with the RHT. Conversations with one-shot RHTs can be terminated from the gateway server before the RHT terminates. The TIP must perform its cleanup as well. Cleanup is only performed at the termination request of the client application.

The client application can request a normal or an aborted termination.

Refer to "Terminating the Conversation" on page 4-27 for more information.

Persistent remote host transaction client applications must:

- Declare RHT/TIP datatypes to be exchanged. All client applications must declare variables to be exchanged with the RHTs using TIPs. PL/SQL datatypes for such variables have already been defined in the TIP corresponding to each
A persistent RHT can be controlled with one or more TIP function calls. The RHT might be designed, for example, to loop and return output for every input until the conversation is explicitly terminated. Or it could have been designed to accept as input a count or list of operations to perform and return the results in multiple exchanges for which the TIP function has only OUT parameters.

A persistent RHT can also be interactive, each output being specified by a previous input selection and ending only when the conversation has been explicitly terminated by the client application.

The TIP function calls available to the client applications and their specific syntax is documented in the TIP Content documentation file for the specific TIP/RHT.

The manner in which the RHT interprets the TIP IN parameters and returns TIP OUT parameters must be determined from the RHT or explained by the RHT programmer. The TIP provides the function calls and the exchanged parameter datatypes to facilitate the client application’s control of the RHT and imposes no limitations or preconditions on the sequence of operations the RHT is directed to perform. The TIP provides the client application with the calls and data parameters the RHT was defined to accept in the PG DD.
Terminate the RHT using the TIP termination function. Regardless of the type of RHT being accessed, the TIP terminate function should be called to clean up and terminate the conversation with the RHT. Conversations with persistent RHTs can be terminated from the gateway server before the RHT terminates, or the RHT might have already terminated. The TIP must perform its cleanup as well and this cleanup is only performed at the termination request of the client application.

The client application can request a normal or an aborted termination.

Refer to "Terminating the Conversation" on page 4-27 for more information.

**Multi-conversational remote host transaction client applications must:**

- Declare RHT/TIP datatypes to be exchanged. All client applications must declare variables to be exchanged with the RHTs using TIPs. PL/SQL datatypes for such variables have already been defined in the TIP corresponding to each RHT, and the client application need only reference the TIP datatype in its declaration. Refer to "Declaring TIP Variables" on page 4-20 for more information. Also refer to the TIP content documentation file for the specific TIP/RHT for more information about the exact usage of these variables.

- Initialize each RHT involved, using the TIP initializing function. A specific customized TIP exists for each RHT as defined in the PG DD. Client applications that control multiple RHTs are multi-conversational and must start each RHT and its associated conversation. This is done by calling each TIP initialization function as before; but multiple TIPs are initialized.

If a single RHT is designed to perform multiple services for one or more callers and if the client application is designed to use this RHT, the TIP corresponding to that RHT can be initialized multiple times by the client application.

The client application subsequently distinguishes from active RHTs under its control using:

- TIP schema `tipname.callname` when multiple TIP/RHTs are being controlled. By encoding the same TIP schema name on TIP user calls, the client application specifies to which RHT the call is being made.

- `tranuse` IN OUT parameter value when multiple instances of the same TIP/RHT are being controlled. This is the value returned on the TIP initialization function call and subsequently passed as an IN parameter on the user-defined TIP function calls. The returned `tranuse` value corresponds to that conversation connected to a given instance of an RHT. By supplying the same `tranuse` value on TIP user calls, the client
application specifies to which RHT instance the given RHT call is being made.

Client application logic must keep track of which RHTs have been started and which TIPs and tranuse values correspond to started RHTs.

- Exchange data with each RHT, using the TIP user function(s), either once or repetitively if the RHT is one-shot or persistent. Client application logic must sequence the RHTs though their allowed steps in accordance with proper RHT operation, as does a user operating the RHTs interactively.

Client application logic must also perform any cross-RHT result analysis or data transfer that might be required. All TIPs execute in isolation from each other.

Output from one RHT intended as input to another RHT must be received in the client application as an IN or IN OUT parameter from the first RHT and sent as an IN or IN OUT parameter from the client application to the second RHT. All TIP-to-RHT function calls must be performed by the client application and data parameters exchanged must have been declared as variables by the client application. The TIPs provide both the required datatype definitions and the RHT function calls for the client application.

Refer to the TIP content documentation file for each specific TIP/RHT for the exact syntax of the TIP function calls and definitions of the parameter datatypes exchanged.

- Terminate each initialized RHT, using the TIP termination function. To terminate an RHT, its corresponding TIP termination function must be called to terminate the RHT and its conversation and to initiate TIP cleanup. The RHT to be terminated is specified by its TIP schema name (the same schema as for its data exchange function calls) and the tranuse value when multiple instances of the same RHT are being terminated.

RHTs and their corresponding TIPs can be terminated in any sequence desired by the client application and do not have to be terminated in the same order in which they are initialized.
Ensuring TIP and Remote Transaction Program Correspondence

A remote host transaction program and its related TIP with client application must correspond on two key requirements:

- Parameter datatype conversion, which results from the way in which transaction DATA is defined. Refer to Appendix G, "Datatype Conversions" for a discussion of how PGAU-generated TIPs convert data based on the data definitions.
- APPC send/receive synchronization, which results from the way in which transaction CALLs are defined

These DATA and CALL definitions are then included by reference in a TRANSACTION definition.

DATA Correspondence

Using data definitions programmed in the language of the remote host transaction, the PGAU DEFINE DATA command stores in the PG DD the information needed for PGAU GENERATE to create the TIP function logic to perform:

- all data conversion from PL/SQL IN parameters supplied by the receiving remote host transaction
- all buffering into the format expected by the receiving remote host transaction
- all data unbuffering from the format supplied by the sending remote host transaction

Note: The specific syntax of the various TIP data exchange variables function calls is the same as was previously defined in the PG DD for the particular RHT and can be researched by examining the TIP content documentation file (tipname.doc) or the TIP specification file produced when the TIP was generated. If a TIP has not yet been generated for the RHT being accessed, refer to Chapter 3, "Creating a TIP", and "DATA Correspondence" on page 4-14, "CALL Correspondence" on page 4-16, and "TRANSACTION Correspondence" on page 4-18 for more information. It is preferable to define and generate the TIP first, however, so that the client application reference documentation is available to you when needed.
Ensuring TIP and Remote Transaction Program Correspondence

- all data conversion to PL/SQL OUT parameters supplied by the sending remote host transaction

PGAU determines the information needed to generate the conversion and buffering logic from the data definitions included in the remote host transaction program. PGAU DEFINE DATA reads this information from files, such as COBOL copy books, or in-stream from scripts and saves it in the PG DD for repeated use. The Gateway Administrator needs to transfer these definition files from the remote host to the Oracle host where PAGU runs.

From the data definitions stored in the PG DD, PAGU GENERATE determines the remote host datatype and matches it to an appropriate PL/SQL datatype. It also determines data lengths and offsets within records and buffers and generates the needed PL/SQL logic into the TIP. Refer to the PAGU "DEFINE DATA" statement on page 2-10 in Chapter 2, "Procedural Gateway Administration Utility" and "Sample PAGU DEFINE DATA Statements" in Appendix E, "Administration Utility Samples" for more information.

All data that are referenced as parameters by subsequent calls must first be defined using PAGU DEFINE DATA. Simple data items, such as single numbers or character strings, and complex multi-field data aggregates, such as records or structures, can be defined. PAGU automatically generates equivalent PL/SQL variables and records of fields or tables for the client application to reference in its calls to the generated TIP.

As discussed, a parameter might be a simple data item, such as an employee number, or a complex item, such as an employee record. PAGU DEFINE DATA automatically extracts the datatype information it needs from the input program data definition files.

In this example, empno and emprec are the arguments to be exchanged.

```
pgadb2i_main(trannum,empno,emprec)
```

A PAGU DEFINE DATA statement must therefore be issued for each of these parameters:

```
DEFINE DATA EMPNO
  PLSDNAME (EMPNO)
  USAGE (PASS)
  LANGUAGE (IBMVS COBOL II)
{
  01 EMP-NO PIC X(6).
}
```

Client Application Development 4-15
DEFINE DATA EMPREC
   PLSDNAME (DCLEMP)
   USAGE (PASS)
   LANGUAGE (IBM/SCOBOLII)
   INFILE("emp.cob");

Note that a definition is not required for the tranum argument. This is the APPC conversation identifier and does not require a definition in PGAU.

**CALL Correspondence**

The requirement to synchronize APPC SENDs and RECEIVEs means that when the remote transaction program expects data parameters to be input, it issues APPC RECEIVEs to read the data parameters. Accordingly, the TIP must cause the gateway to issue APPC SENDs to write the data parameters to the remote transaction program. The TIP must also cause the gateway to issue APPC RECEIVEs when the remote transaction program issues APPC SENDs.

The PGAU DEFINE CALL statement specifies how the generated TIP is to be called by the client application and which data parameters are to be exchanged with the remote host transaction for that call. Each PGAU DEFINE CALL statement might specify the name of the TIP function, one or more data parameters, and the IN/OUT mode of each data parameter. Data parameters must have been previously defined with PGAU DEFINE DATA statements. Refer to "DEFINE CALL" on page 2-9 in Chapter 2, "Procedural Gateway Administration Utility" and "Sample PGAU DEFINE CALL Statements" in Appendix E for more information.

PGAU DEFINE CALL processing stores the specified information in the PG DD for later use by PGAU GENERATE. PGAU GENERATE then creates the following in the TIP package specification:

- declarations of public PL/SQL functions for each CALL defined with PL/SQL parameters for each DATA definition specified on the CALL
- declarations of the public PL/SQL data parameters

The client application calls the TIP public function as a PL/SQL function call, using the function name and parameter list specified in the PGAU DEFINE CALL statement. The client application might also declare, by reference, private variables of the same datatype as the TIP public data parameters to facilitate data passing and handling within the client application, thus sharing the declarations created by PGAU GENERATE.
In this example, the following PGAU DEFINE CALL statement must be issued to define the TIP public function:

```sql
DEFINE CALL DB2IMAIN
    PKGCALL (pgadb2i_main)
    PARMS ((empno IN),(emprec OUT));
```

**Flexible Call Sequence**

The number of data parameters exchanged between the TIP and the gateway on each call can vary at the user’s discretion, as long as the remote transaction program’s SEND/RECEIVE requests are satisfied. For example, the remote transaction program data exchange sequence might be:

- **APPC SEND** 5 fields (field1-field5)
- **APPC RECEIVE** 1 field (field6)
- **APPC SEND** 1 field (field7)
- **APPC RECEIVE** 3 fields (field8 - field10)

The resulting TIP/application call sequence could be:

```sql
tip_call1(parm1 OUT, <-- APPC SEND field1 from remote TP
    parm2 OUT, <-- APPC SEND field2 from remote TP
    parm3 OUT); <-- APPC SEND field3 from remote TP

tip_call2(parm4 OUT, <-- APPC SEND field4 from remote TP
    parm5 OUT); <-- APPC SEND field5 from remote TP

tip_call3(parm6 IN OUT); --> APPC RECEIVE field6 in remote TP
    <-- APPC SEND field7 from remote TP

tip_call4(parm8 IN,   --> APPC RECEIVE field8 into remote TP
    parm9 IN,   --> APPC RECEIVE field9 into remote TP
    parm10 IN); --> APPC RECEIVE field10 into remote TP
```

To define these four public functions to the TIP, four PGAU DEFINE CALL statements must be issued, each specifying its unique public function name (tip_callx) and the data parameter list to be exchanged. Once a data item is defined using DEFINE DATA, it can be referenced in multiple calls in any mode (IN, OUT, or IN OUT). For example, parm5 could be used a second time in place of parm6. This implies the same data is being exchanged in both instances, received into the TIP and application on `tip_call2` and returned, possibly updated, to the remote host in `tip_call4`.

Notice also that the remote transaction program’s first five written fields are read by two separate TIP function calls, `tip_call1` and `tip_call2`. This could also have been equivalently accomplished with five TIP function calls of one OUT.
parameter each or a single TIP function call with five OUT parameters. Then the remote transaction program’s first read field (field6) and subsequent written field (field7) correspond to a single TIP function call (tip_call3) with a single IN OUT parameter (parm6).

This use of a single IN OUT parameter implies that the remote transaction program’s datatype for field6 and field7 are both the same and correspond to the conversion performed for the datatype of parm6. If field6 and field7 were of different datatypes, then they have to correspond to different PL/SQL parameters (for example, parm6 IN and parm7 OUT). They could still be exchanged as two parameters on a single TIP call or one parameter each on two TIP calls, however.

Lastly, the remote transaction program’s remaining three RECEIVE fields are supplied by tip_call4 parameters 8-10. They also could have been done with three TIP calls passing one parameter each or two TIP calls passing one parameter on one call and two parameters on the other, in either order. This flexibility permits the user to define the correspondence between the remote transaction program’s operation and the TIP function calls in whatever manner best suits the user.

Call Correspondence Order Restrictions

Each TIP public function first sends all IN parameters, before it receives any OUT parameters. Thus, a remote transaction program expecting to send one field and then receive one field must correspond to separate TIP calls.

For example:

```plaintext
tip_call0( parm0 OUT);  <-- APPC SEND outfield from remote TP
```

PGAXFER RPC checks first for parameters to send, but finds none and proceeds to receive parameters:

```plaintext
tip_call1( parm1 IN);  --> APPC RECEIVE infield to remote TP
```

PGAXFER RPC processes parameters to send and then checks for parameters to receive, but finds none and completes; therefore, a single TIP public function with an OUT parameter followed by an IN parameter does not work, because the IN parameter is processed first—regardless of its position in the parameter list.

TRANSACTION Correspondence

The remote host transaction is defined with the PGAU DEFINE TRANSACTION statement with additional references to prior definitions of CALLs that the transaction supports.
You specify the remote host transaction attributes, such as:

- transaction ID or name
- network address or location
- system type (such as IBM370)
- Oracle National Language of the remote host

---

**Note:** The PL/SQL package name is specified when the transaction is defined; this is the name by which the TIP is referenced and which the public function calls to be included within the TIP. Each public function must have been previously defined with a PGAU DEFINE CALL statement, which has been stored in the PG DD. If you do not specify a package name (TIP name) in the GENERATE statement, the transaction name you specified will become the package name by default. In that case, the transaction name (t.name) must be unique and must be in valid PL/SQL syntax within the database containing the PL/SQL packages.

For more information, refer to "DEFINE TRANSACTION" in Chapter 2, "Procedural Gateway Administration Utility" and "Sample PGAU DEFINE TRANSACTION Statement" in Appendix E, "Administration Utility Samples".

---

In this example, the following DEFINE TRANSACTION statements are used to define a remote CICS transaction called DB2I:

```
DEFINE TRANSACTION DB2I
CALL (   DB2IMAIN,
         DB2IDIAG   )
SIDEPROFILE(CICSPROD)
TPNAME(DB2I)
LOGMODE(ORAPLU62)
SYNCELEVEL(0)
NLS_LANGUAGE("AMERICAN_AMERICA.WE8EBCDIC37C");
```
Calling the TIP from the Client Application

Once a TIP is created, a client application must be written to interface with the TIP. A client application that calls the TIP functions must include five logical sections:

- declaring TIP variables
- initializing the conversation
- exchanging data
- terminating the conversation
- error handling

Declaring TIP Variables

The user declarations section of the `tipname.doc` file documents the required declarations.

When passing PL/SQL parameters on calls to TIP functions, the client application must use the exact same PL/SQL datatypes for TIP function arguments as are defined by the TIP in its specification section. Assume, for example, the following is in the TIP specification, or `tipname.doc`:

```plaintext
FUNCTION tip_call1 tranuse, IN BINARY_INTEGER,
    tip_var1 io_mode pls_type1,
    tip_record io_mode tran_rectype)
RETURN INTEGER;

TYPE tran_rectype is RECORD
    (rec_field1 pls_type1,
    ... rec_fieldN pls_typeN);
```

The following table provides a description of the:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tip_call1</code></td>
<td>The TIP function name as defined in the package specification.</td>
</tr>
<tr>
<td><code>tranuse</code></td>
<td>The remote transaction instance parameter returned from the TIP init function identifying the conversation on which this TIP call is to exchange data.</td>
</tr>
</tbody>
</table>
In the client application PL/SQL atomic datatypes should be defined as the exact same datatype of their corresponding arguments in the TIP function definition. The following should be coded in the client application before the BEGIN command:

```
appl_var pls_type1; /* declare appl variable for .... */
```

TIP datatypes need not be redefined. They must be declared locally within the client application, appearing in the client application before the BEGIN:

```
appl_record tipname.tran_rectype; /* declare appl record */
```

The following table describes the command line arguments:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>appl_record</code></td>
<td>Is a PL/SQL record exchanged with the TIP and used within the client application.</td>
</tr>
<tr>
<td><code>tipname</code></td>
<td>Is the PL/SQL package (TIP) name as stored in Oracle database. This is the same value as in the statement CREATE or REPLACE PACKAGE <code>tipname</code> in the TIP specification.</td>
</tr>
<tr>
<td><code>tran_rectype</code></td>
<td>Is the PL/SQL record datatype declared in the <code>tipname</code> TIP specification. This is the same value as in the TYPE <code>tran_rectype</code> is RECORD statement.</td>
</tr>
</tbody>
</table>

Refer to the `tipname.doc` content file for a complete description of the user declarations you can reference.

The client application calls the TIP public function as if it were any local PL/SQL function:
Calling the TIP from the Client Application

```c
rc = tip_call1( tranuse,
                appl_var,
                appl_record);
```

In the CICS-DB2 inquiry example, the PL/SQL driver `pgadb2id.sql`, which is located in `$ORACLE_HOME/pg4appc/demo/CICS` directory, is the client application and includes the following declaration:

```sql
CREATE or REPLACE PROCEDURE db2idriv(empno IN CHAR) IS
    tranuse INTEGER :=0               /* transaction usage number       */
    DCLEMP PGADB2I.DCLEMP_typ;        /* DB2 EMP row definition          */
    DB2 PGADB2I.DB2_typ;              /* DB2 diagnostic information      */
    rc INTEGER :=0                    /* PGA RPC return codes             */
    line VARCHAR2(132);               /* work buffer for output           */
    term INTEGER :=0;                 /* 1 if pgadb2i_term called          */
    ...
    ...
```

### Initializing the Conversation

The call to initialize the conversation serves several purposes:

- To cause the PL/SQL package, the TIP, to be loaded and to perform the initialization logic programmed in the TIP initialization section.

- To cause the TIP init function to call the PGAINIT remote procedural call (RPC), which in turn establishes communication with the remote transaction program (RTP), and returns a transaction instance number to the application.

Optionally, calls to initialize the conversation can be used to:

- Override default RHT/OLTP identification, network address attributes, and conversation security user ID and password.

- Specify what diagnostic traces the TIP is to produce. Refer to Chapter 6, "Problem Determination" for more information about diagnostic traces.

PGAU-generated TIPs provide four different initialization functions that client applications can call. These are overloaded functions which all have the same name, but vary in the types of parameters passed.

Three initialization parameters are passed:

- The transaction instance number for RHT conversation identification. The `tranuse` parameter is required on all TIP initializations.
■ TIP diagnostic flags for TIP runtime diagnostic controls. The `tipdiag` parameter is optional. Refer to Chapter 6, 'Problem Determination' for a discussion of TIP diagnostics.

■ TIP default overrides for overriding OLTP and network attributes. The `override` parameter is optional.

The following four functions are shown as they might appear in the TIP Content documentation file. Examples of client application use are provided later.

```plaintext
TYPE override_Typ IS RECORD (  
  traname  VARCHAR2(255),  /* Transaction Program */  
  transync BINARY_INTEGER, /* RESERVED */  
  trannls VARCHAR2(50),    /* RESERVED */  
  oltpname VARCHAR2(255),  /* Logical Unit */  
  oltpmode VARCHAR2(255),  /* LOG Mode Entry */  
  netaddr VARCHAR2(255),   /* Side Profile */  
  oltpuser VARCHAR2(8),    /* userid for OLTP access */  
  oltppass VARCHAR2(8));   /* password for OLTP access*/

FUNCTION pgadb2i_init(                /* init standard */  
  tranuse IN OUT BINARY_INTEGER)  
  RETURN INTEGER;

FUNCTION pgadb2i_init(                /* init override */  
  tranuse IN OUT BINARY_INTEGER,  
  override IN override_Typ)  
  RETURN INTEGER;

FUNCTION pgadb2i_init(                /* init diagnostic */  
  tranuse IN OUT BINARY_INTEGER,  
  tipdiag IN CHAR)  
  RETURN INTEGER;

FUNCTION pgadb2i_init(                /* init over-diag */  
  tranuse IN OUT BINARY_INTEGER,  
  override IN override_Typ,  
  tipdiag IN CHAR)  
  RETURN INTEGER;
```

**Transaction Instance Parameter**

This transaction instance number (shown in examples as `tranuse`) must be passed to subsequent TIP exchange and terminate functions. It identifies to the
Calling the TIP from the Client Application

gateway on which APPC conversation--and therefore which iteration of a remote transaction program--the data is to be transmitted or communication terminated.

A single client application might control multiple instances of the same remote transaction program or multiple different remote transaction programs, all concurrently. The transaction instance number is the TIP’s mechanism for routing the client application call through the gateway to the intended remote transaction program.

It is the responsibility of the client application to save the transaction instance number of each active transaction and pass the correct one to each TIP function called for that transaction.

The client application calls the TIP initialization function as if it were any local PL/SQL function. For example:

```
... 

tranuse INTEGER := 0;/* transaction usage number*/  
... 

BEGIN  
rc := pgadb2i.pgadb2i_init(tranuse);  
... 
```

Overriding TIP Initializations

Note that in the preceding example the client application did not specify any remote transaction program name, network connection, or security information. The TIP has such information internally coded as defaults and the client application simply calls the appropriate TIP for the chosen remote transaction program. The client application can, however, optionally override some TIP defaults and supply security information.

You do not need to change any client applications that do not require overrides.

When the remote host transaction was defined in the PG DD, the DEFINE TRANSACTION statement specified certain default OLTP and network identification attributes which can be overridden:

- TPname
- LUname
- LOGMODE
• Side Profile

Refer to "DEFINE TRANSACTION" in Chapter 2, "Procedural Gateway Administration Utility" for more information about the DEFINE TRANSACTION statement.

These PG DD-defined transaction attributes are generated into TIPs as defaults and can be overridden at TIP initialization time. This facilitates the use of one TIP, which can be used with a test transaction or system, and can later be used with a production transaction or system, without having to regenerate the TIP.

The override_Typ record datatype describes the various transaction attributes that can be overridden by the client application. The following overrides are currently supported:

• tranname can be set to override the value that was specified by the TPNAME parameter of the DEFINE TRANSACTION statement

• oltpname can be set to override the value that was specified by the LUNAME parameter of the DEFINE TRANSACTION statement

• oltpmode can be set to override the value that was specified by the LOGMODE parameter of the DEFINE TRANSACTION statement

• netaddr can be set to override the value that was specified by the SIDEPROFILE parameter of the DEFINE TRANSACTION statement

In addition to the transaction attributes defined in the PG DD, there are two security-related parameters, conversation security user ID and conversation security password, that can be overridden at TIP initialization time. The values for these parameters normally come from either the database link used to access the gateway or the Oracle database session. There are cases when the Oracle database user ID is not sufficient for accessing the OLTP system. The user ID and password overrides provide a way to specify those parameters to the OLTP system.

The following overrides are currently supported:

• oltpuser can be set to override the user ID used to initialize the conversation with the OLTP

• oltpass can be set to override the password used to initialize the conversation with the OLTP

The security overrides have an effect only if PGA_SECURITY_TYPE=PROGRAM is specified in the gateway initialization file, and the OLTP system is configured to accept a user ID and password on incoming conversation requests.
The `transync` (APPC synclevel) and `trannls` (NLS character set) are defined in the override record datatype, but are reserved for future use. The RHT synclevel and NLS name cannot be overridden.

The client application might override the default attributes at TIP initialization for the following reasons:

- to start a different version of the RHT (such as production instead of test)
- to change the location of the OLTP containing the RHT (if the OLTP was moved due to migration or a switch to backup configuration)

Client applications requiring overrides can use any combination of override and initialization parameters and might alter the combination at any time without regenerating the TIP or affecting applications that do not override parameters.

To override the TIP defaults, an additional client application record variable must be declared as `override_Typ` datatype, values must be assigned to the override subfields, and the override record variable must be passed on the TIP initialization call from the client application. For example:

```plaintext
...
my_overrides pgadb2i.override_Typ;   -- declaration
...
my_overrides.oltpname := 'CICSPROD'; -- swap to production CICS
my_overrides.tranname := 'TNEW';     -- new transaction name
BEGIN
  rc := pgadb2i.pgadb2i_init(tranuse,my_overrides); -- init
...
...
```

Within the TIP, override attributes are checked for syntax problems and passed to the gateway server.

**Security Considerations**

The security requirements of the default and overridden OLTPs must be the same because the same gateway server is used in either conversation, as dictated by the database link names in the PGA RPC calls. The gateway server startup security mode is set at gateway server initialization time and passed unchanged to the OLTP at TIP or conversation initialization time.
Exchanging Data

The client application should pass the transaction instance number, returned from a previous tip_init call, to identify which remote transaction program is affected and to identify any client application data parameters to be exchanged with the remote transaction program.

In this CICS-DB2 inquiry example, we pass an employee number and receive an employee record back:

```c
rc = pgadb2i.pgadb2i_main(tranuse, /* transfer data */
    empno,  /* employee number */
    DCLEMP); /* return employee record*/
```

Terminating the Conversation

The client application calls the TIP termination function as if it were any local PL/SQL function. For example:

```c
...  
...  
term := 1;  /* indicate term called */
rc := pgadb2i.pgadb2i_term(tranuse,0); /* terminate normally */
...  
...  
```

After a transaction instance number has been passed on a TIP terminate call to terminate the transaction, or after the remote transaction program has abended, that particular transaction instance number might be forgotten.

Error Handling

The client application should include an exception handler that can clean up any active APPC conversations before the client application terminates. The sample client application provided in `pgadb2id.sql` contains an example of exception handling.

Gateway exceptions are reported in the range 20900 to 20999. When an exception occurs, the TIP termination function should be called for any active conversations that have been started by prior calls to the TIP initialization function. For example:

```c
EXCEPTION
    WHEN OTHERS THEN
        IF term = 0 THEN  /* terminate function not called yet */
            rc := pgadb2i.pgadb2i_term(tranuse,1); /* terminate abnormally*/
```
The remote transaction should also include provisions for error handling and debugging, such as writing debugging information to the CICS temporary storage queue area. Refer to the PL/SQL User’s Guide and Reference for a discussion of how to intercept and handle Oracle exceptions.

Granting Execute Authority

The TIP is a standard PL/SQL package and execute authority must be granted to users who call the TIP from their client application. In this example, we grant execute on the PGADB2I package to user SCOTT:

```
GRANT EXECUTE ON PGADB2I TO SCOTT
```

Refer to the Oracle9i Server Administrator’s Reference for further information.

Executing the Application

Before executing the client application, ensure that a connection to the host is established and that the receiving partner is available. In this example we use PL/SQL driver DB2IDRIV to execute the CICS-DB2 inquiry. To execute this client application, enter from SQL*Plus:

```
set serveroutput on
execute DB2IDRIV('nnnnnn');
```

APPC Conversation Sharing

Multiple TIPs can share the same APPC conversation with one or more Remote Host Transactions (RHTs) which are also sharing that same conversation. Two benefits derive from this feature:

- Existing RHTs which rely upon passing control of a conversation are supported by Oracle Procedural Gateway for APPC.
- TIPs otherwise too large for PL/SQL compilation can be separated into multiple smaller TIPs, each with fewer user-defined functions, providing the client application with the same set of function calls and data definitions without any change to the RHT.
APPC Conversation Sharing Concepts

Mainframe OLTPs, such as IMS, allow transactions to share a single APPC conversation by passing it when the transaction calls another transaction. RHTs are defined to PGAU as single transactions with calls, inputs and outputs for which PGAU generates a single TIP with initialization, transfer and termination functions corresponding to that specific RHT.

Logic generated into every TIP allows that TIP either:

■ to initiate a new conversation when its init function is called, or
■ to transfer data on an existing conversation when its user-defined functions are called, or
■ to terminate an existing conversation when its "term" function is called.

An APPC conversation is treated as a resource shared and managed by multiple TIPs. There is no requirement for any TIP to be the sole user of an APPC conversation.

Any TIP generated at 3.4.0 or later can perform any of the following combinations of service:

■ initiate
■ initiate and transfer
■ initiate, transfer, and terminate (standard operation)
■ transfer
■ transfer and terminate
■ terminate
■ initiate and terminate (assumes other TIPs perform transfer)

A single APPC conversation can be shared in the following ways:

■ from one TIP to multiple RHTs
■ from multiple TIPs to one RHT
■ from multiple TIPs to multiple RHTs

Without APPC conversation sharing, a single TIP must be defined which contains all functions and data for all RHTs which a client application might need to call. Creating TIPs with a superset of RHTs often causes such TIPs to be too large for PL/SQL to compile.
Conversely, with APPC conversation sharing, each RHT (or even each RHT data exchange for those RHTs which perform multiple, different data exchange operations) can be defined in a single TIP which is smaller and less likely to exceed PL/SQL compilation limits.

**APPC Conversation Sharing Usage**

APPC conversation sharing is automatically available in every TIP generated at 3.4.0 or later. No TIPs generated before 3.4.0 can participate in APPC conversation sharing. TIPs generated before 3.4.0 must be regenerated using PGAU 3.4.0 or later to participate in APPC conversation sharing. PGAU is upward compatible and regeneration should be transparent, provided only the regenerated TIP body *(tipname.pkb)* is recompiled. If the TIP specification is also recompiled, the client application needs recompilation as well. Refer to Appendix B, "TIP Internals" for more detailed information.

Definition and generation of TIPs is accomplished as previously discussed in Chapters 1, 2, and 3. No additional options or parameters need be specified.

Run-time use of APPC conversation sharing is under the control of the client application. It is accomplished simply by calling the init function of one of the TIPs that share a conversation and passing the *tranuse* value returned to the other TIP functions as each is called in its desired order. Any TIP init function can be used, provided that all TIPs were defined with the same DEFINE TRANSACTION TPNAME or SIDEPROFILE value. The TPNAME or SIDEPROFILE value specifies which RHT to initialize.

When the init function of an APPC conversation sharing-capable TIP is called to initialize a conversation, the *tranuse* value returned indicates conversation sharing is enabled. By passing that same *tranuse* value when calling functions in other TIPs, those other TIPs perform their transfers on the same conversation already initialized, provided that all TIPs involved were generated at Version 3.4.0 or later.

**APPC Conversation Sharing TIP Compatibility**

TIPs generated at 3.4.0 or later of the procedural gateway use and expect different values for *tranuse* than do pre-3.4.0 TIPs. If a pre-3.4.0 TIP is used to initialize a conversation and its *tranuse* value is passed to a 3.4.0 or later generated TIP, the following exception is raised:

ORA-20704 PGA_TIP: tranuse value cannot be shared
Pre-3.4.0 generated TIPs do not detect the different tranuse value for shared conversations, however, and this can result in unpredictable errors.

---

**Caution:** All TIPs called in a shared conversation must have been generated at 3.4.0 or later.

No TIPs generated before 3.4.0 can participate in APPC conversation sharing.

---

The tranuse values are incompatible between pre-3.4.0 and 3.4.0 or later releases. This should not pose a problem for you for the following reason: before 3.4.0, all RHT functions defined in a TIP had to be called through that TIPs functions, and the init function of that same TIP had to be called first to initialize the conversation. The tranuse value was only valid for the TIP which initialized it. Thus, unless you make programming changes, it is not possible for an existing application to accidentally mix tranuse values.

Pre-3.4.0 TIPs and client applications can continue to be used without change and old client applications can call new 3.4.0 or later TIPs without change. This is made possible when an old TIP body is regenerated and compiled; the TIP now becomes capable of APPC conversation sharing, even though the old client application has not changed.

None of the functions of a pre-3.4.0 TIP can share an APPC conversation. However, once a TIP is regenerated at 3.4.0 or later, any of its functions can share APPC conversations.

**APPC Conversation Sharing for TIPs That Are Too Large**

You can use conversation sharing to circumvent a TIP that is too large to compile. This is identified by 'PLS-00123 - package too large to compile', or some other problem symptom such as PL/SQL compilation hanging. In this case you must choose which function calls to remove from the former TIP and define into new TIPs.

Specifically, you must decide which PGAU DEFINE CALL statements and their related DEFINE DATA statements should be moved from the old PGAU control file (.ctl) into one or more new PGAU control files. In addition, you must decide which PGAU DEFINE TRANSACTION statements should be included in each new PGAU control file defining each new TIP.
You must consider the following PGAU statements:

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFINE DATA statements</td>
<td>Must be unique. They can be shared by all affected PGAU control files, provided they are defined to the Procedural Gateway Data Dictionary (PG DD) before being referenced by DEFINE CALL statements. No changes are needed to these statements.</td>
</tr>
<tr>
<td>DEFINE CALL statements</td>
<td>Must be unique. They need only be referenced by the new DEFINE TRANSACTION statement of the TIP in which they are included, provided they are defined to the PG DD before being referenced by a DEFINE TRANSACTION statement. The DEFINE CALL statements can optionally be moved to the new PGAU control file of the TIP in which they are included.</td>
</tr>
<tr>
<td>DEFINE TRANSACTION statements</td>
<td>Specified for each new TIP desired and will reference those call definitions moved from the former large TIP to the new small TIPs. No transaction attributes will change. This allows any new TIP to perform the same initialization or termination with the same RHT as the former large TIP. The old DEFINE TRANSACTION statement (of the former large TIP) should now exclude any call definitions which are being moved to new small TIPs.</td>
</tr>
</tbody>
</table>

**APPC Conversation Sharing Example**

Assume the existence of RHTs A, B and C, and that RHT A performs a menu selection and calls RHT B for a query function or RHT C for an update followed by a select function.

You could define the following DATA and CALLs:

- DEFINE DATA choice ...
- DEFINE DATA input ...
- DEFINE DATA answer ...
- DEFINE DATA record ...
- DEFINE CALL menu_A callname(pick) parms(choice in);
- DEFINE CALL query_B callname(query) parms((input in),
  (answer out));
- DEFINE CALL update_C callname(update) parms(record in);
- DEFINE CALL select_C callname(select) parms(record out);

The following example TIPs could be defined:

**Example 1**
This example does not use APPC conversation sharing, but is a valid TIP definition created before release 3.4.0, combining the functions of RHTs A, B and C.

```
DEFINE TRANSACTION rhtABC calls(menu_A,
       query_B,
       update_C,
       select_C)
  tpname(RHTA);
```

This TIP includes all data definitions and calls, and might be too large to compile. This TIP does not use APPC conversation sharing as there is only the one TIP, rhtABC. The RHTs do, however, perform their normal sharing of the conversation at the remote host. If the TIP was small enough to compile, the client application calls TIP functions as follows:

```
rc := rhtABC.rhtABC_init(tranuse);
rc := rhtABC.pick(tranuse, choice);
rc := rhtABC.query(tranuse, input, answer);
rc := rhtABC.update(tranuse, record);
rc := rhtABC.select(tranuse, record);
rc := rhtABC.rhtABC_term(tranuse);
```

**Example 2**
This example demonstrates defining a set of TIPs with APPC conversation sharing, separating the functions of RHTs A, B and C into three TIPs:

```
DEFINE TRANSACTION rhtA calls(menu_A) tpname(RHTA);
DEFINE TRANSACTION rhtB calls(query_B) tpname(RHTA);
DEFINE TRANSACTION rhtC calls(update_C,
       select_C) tpname(RHTA);
```
Each TIP includes only the call and data it requires, and each TIP automatically performs APPC conversation sharing. The client application calls these functions as follows:

```c
rc := rhtA.rhtA_init(tranuse);
r := rhtA.pick(tranuse, choice);
r := rhtB.query(tranuse, input, answer);
r := rhtC.update(tranuse, record);
r := rhtC.select(tranuse, record);
r := rhtB.rhtB_term(tranuse);
```

The only client application difference between the two examples is in the schema qualifier on each of the TIP calls. This is because the function being called is in a different TIP which has a different package name in the database.

Only new DEFINE TRANSACTION statements were needed to make use of APPC conversation sharing. The CALL and DATA definitions were used as-is. This means the old TIP `rhtABC` is still defined as it was and might still be too large to compile.

**Example 3**

If you performed Sample 2 but you still believe that the TIP may be too large to compile, try this:

```c
DEFINE TRANSACTION rhtABC calls(menu_A) tpname(RHTA);
DEFINE TRANSACTION rhtB   calls(query_B)  tpname(RHTA);
DEFINE TRANSACTION rhtCU  calls(update_C) tpname(RHTA);
DEFINE TRANSACTION rhtCS  calls(select_C) tpname(RHTA);
```

TIP `rhtABC` has had three functions removed so it is now smaller and more likely to compile. TIP `rhtB` has one function and TIP `rhtC` has been separated into two TIPs even though the corresponding host functions remain in a single RHT.

The client application calls these functions as follows:

```c
rc := rhtB.rhtB_init(tranuse);
r := rhtABC.pick(tranuse, choice);
r := rhtB.query(tranuse, input);
r := rhtCU.update(tranuse, record);
r := rhtCS.select(tranuse, record);
r := rhtABC.rhtABC_term(tranuse);
```

A different TIP is used for initialization, illustrating that all TIPs contain the init and term functions, and because the DEFINE TRANSACTION statements all specified
the same tpname (RHTA), the same remote host transaction is always called for initialization.

**APPC Conversation Sharing Overrides and Diagnostics**

TIP default override parameters are processed in the TIP init function which was called to perform initialization. Once the APPC conversation is established, no further sharing of overriding parameters is necessary. You need do nothing more than pass the overrides to the TIP init function.

TIP diagnostic parameters are shared among all TIPs sharing a given conversation. In effect, requesting diagnostics of the TIP performing initialization causes the same diagnostics to be requested of all TIPs sharing the conversation. Requesting diagnostics from only one TIP of several sharing a conversation is not possible. The application designer or user need only pass the TIP runtime trace controls to the TIP init function.

**Application Development with Multi-Byte Character Set Support**

IBM VS COBOLII presently only supports double byte character sets (DBCS) for PIC G datatypes.

PGAU processes IBM VS COBOLII PIC G datatypes as PL/SQL VARCHAR2 variables and generates TIPs which automatically convert the data according to the Oracle NLS_LANGUAGEs specified for the remote host data and the local Oracle data.

These Oracle NLS_LANGUAGEs can be specified as defaults for all PIC G data exchanged by the TIP with the remote transaction (see DEFINE TRANSACTION ... REMOTE_MBCS or LOCAL_MBCS). The Oracle NLS_LANGUAGEs for any individual PIC G data item can be further overridden (see REDEFINE DATA ... REMOTE or LOCAL_LANGUAGE).

DBCS data can be encoded in any combination of supported DBCS character sets. For example, a remote host application which allows different codepages for each field of data in a record is supported by the Oracle Procedural Gateway MBCS support.

Use of REDEFINE DATA ... REMOTE_LANGUAGE or LOCAL_LANGUAGE on PIC X items is also supported. Thus a TIP can perform DBCS or MBCS conversions for specified PIC X data fields, in addition to SBCS conversions by default for the remaining PIC X data fields. Default SBCS conversion is according to the DEFINE
TRANSACTION...  NLS_LANGUAGE and local Oracle default LANGUAGE environment values.

When PGAU is generating a TIP, the PIC G datatypes are converted to PL/SQL VARCHAR2s. After conversion by the TIP, received ‘PIC G’ VARCHAR2s can have a length less then the maximum due to deletion of shift-out and shift-in meta characters, and sent ‘PIC G’ RAWs will have the shift-out and shift-in characters inserted as required by the remote host character set specified.

This is different from the conversions performed for PIC X data which is always a known fixed-length and hence CHAR datatypes are used in TIPs for PIC X data fields. However, even when the PIC X field contains DBCS or MBCS data, a CHAR variable is still used and padded with blanks if needed.

Some remote host applications bracket a PIC G field with PIC X bytes used for shift-out, shift-in meta-character insertion. Such a COBOL definition might look like:

```cobol
01 MY_RECORD.
   05 SO PIC X.
   05 MY_MBCS_DATA PIC G(50).
   05 SI PIC X.
```

This is not processed correctly by PGAU, because all three fields are defined, and consequently treated, as separate data items when conversion is performed.

To be properly processed, the definition input to PGAU should be:

```cobol
01 MY_RECORD.
   05 MY_MBCS_DATA PIC G(51).
```

The PGAU REDEFINE DATA statement can redefine the 3-field definition to the 1-field definition by specifying USAGE(SKIP) on fields SO and SI, and ‘05 MY_MBCS_DATA PIC G(51).’ to redefine MY_MBCS_DATA. The three REDEFINE statements can be placed in the PGAU input control file, and thus the remote host definition need not be altered.

**Modifying a Terminal-Oriented Transaction to Use APPC**

The remote transaction program must include mapped APPC verbs to initiate, communicate, and terminate the APPC conversation. However, when the remote transaction program is terminal-oriented, the following options are available:

- You can separate the terminal logic from the application and I/O logic. Once this separation is achieved, a small front end remote transaction program can be
Privileges Needed to Use TIPs

Execute privileges must be explicitly granted to callers of TIPs or procedures. This privilege cannot be granted through a role.

Any TIP user wanting to trace a TIP must be granted execute privileges on the rtrace and ptrace procedures. Refer to the Oracle Procedural Gateway for APPC Installation and Configuration Guide for your platform for more information.

For example:

```
$ sqlplus pgaadmin/pw@database_specification_string
SQL> grant execute on pgaadmin.purge_trace to tip_user_userid;
SQL> grant execute on pgaadmin.read_trace to tip_user_userid;
```

After a TIP has been developed, the TIP user must be granted execute privileges on the TIP by the TIP owner. The TIP owner is usually PGAADMIN, but can be another user who has been granted either the PGDDDEF or PGDDGEN roles. For example:

```
$ sqlplus tip_owner/pw@database_specification_string
```
Privileges Needed to Use TIPs

SQL> grant execute on tipname to tip_user_userid;
where database_specification_string is the Oracle Net identifier for the Oracle integrating server where the gateway UTL_RAW and UTL_PG components were installed. This is the same Oracle integrating server where the TIPs are executed and where grants on the TIPs are performed from the TIP owner user ID.

A SQL script for performing these grants is provided in the $ORACLE_HOME/pg4apppc/admin directory. The pgddausr.sql script performs the grants for private access to the packages by a single TIP user. If private grants are to be used, the pgddausr.sql script must be run once for each TIP user’s user ID.

To run these scripts, use SQL*Plus to connect to the Oracle integrating server as user PGAADMIN. From SQL*Plus, run the pgddausr.sql script from the $ORACLE_HOME/pg4apppc/admin directory. The script performs the necessary grants as previously described. You are prompted for the required user IDs, passwords, and database specification strings. If you are using private grants, repeat this step for each user ID requiring access to the packages.

No script has been provided to perform public grants. To do this, issue the following commands:

$ sqlplus tip_owner/pw@database_specification_string
SQL> grant execute on tipname to PUBLIC;
Commit-confirm allows the updating of local Oracle resources to occur in the same Oracle transaction as the updating of non-Oracle resources accessed through the Oracle Procedural Gateway for APPC. This chapter assumes that you are familiar with the basic concepts of two-phase commit as discussed in Oracle9i Server Concepts. This chapter includes the following sections:

- Overview of Commit-Confirm on page 5-2
- Supported OLTPs on page 5-2
- Required Components on page 5-3
- Configuring Commit-Confirm on page 5-5
- Application Design Requirements on page 5-8
- Commit-Confirm Architecture on page 5-9
- Sample Applications on page 5-14
Overview of Commit-Confirm

Commit-confirm is a special implementation of two-phase commit that allows a database or gateway that does not support full two-phase commit to participate in distributed update transactions with other databases or gateways that do support full two-phase commit. In this implementation, the commit-confirm site is always the first to be committed, after all other sites have been prepared. This allows all sites to be kept in sync, because if the commit-confirm site fails to commit successfully, all other sites can be rolled back.

Within an Oracle distributed transaction, all work associated with that transaction is assigned a common identifier, known as the Oracle Global Transaction ID. This identifier is guaranteed to be unique, so that it can be used to exclusively identify a particular distributed transaction. The key requirement for commit-confirm support is the ability for the commit-confirm site (in this case, the Oracle Procedural Gateway for APPC) to be able to log the Oracle Global Transaction ID as part of its unit of work, so that if a failure occurs, the gateway’s recovery processing can determine the status of a particular Oracle Global Transaction ID by the presence or absence of a log entry for that transaction. A new Oracle Global Transaction ID is generated after every commit or rollback operation.

The Oracle Procedural Gateway for APPC implements commit-confirm using LU6.2 synclvel 1. This is similar to the implementation of single-site update, with the added advantage that resources on both the Oracle site and the OLTP being accessed by the gateway can be updated and kept in sync. The main difference is that the commit-confirm implementation requires some additional programming in the OLTP transaction to perform the transaction logging necessary for recovery support.

Supported OLTPs

Since commit-confirm uses LU6.2 synclvel 1, it can be supported by any OLTP that supports APPC, including CICS/ESA and IMS/TM. The Oracle Procedural Gateway for APPC provides sample commit-confirm applications for both CICS/ESA and IMS/TM.

With CICS/ESA, the standard command-level EXEC CICS interface can be used for all APPC communications. In addition, the CPI-C interface can be used if it is preferred. A sample DB2 update transaction written in COBOL using the EXEC CICS interface is provided with the gateway. Any language supported by CICS can be used for writing commit-confirm transactions.
With IMS/TM, the CPI-C interface must be used, making the IMS transaction an "explicit APPC transaction," as referred to in the IBM IMS/ESA manuals. This is necessary because it is the only way that the LU6.2 synclevel 1 control flows are accessible to the IMS transaction. When using "implied APPC" where "GU" from the IOPCB and "ISRT" to the IOPCB are used for receiving and sending data, there is no way for the IMS transaction to access the LU6.2 synclevel 1 control flow, making it impossible to use this method for commit-confirm. A sample DLI database update transaction written in COBOL using the CPI-C APPC interface is provided with the gateway. Any language supported by IMS and CPI-C can be used for writing commit-confirm transactions.

**Required Components**

The following components are required to support commit-confirm:

- **Procedural Gateway for APPC Server**

  The gateway server supports commit-confirm when PGA_CAPABILITY=COMMIT_CONFIRM is specified in the gateway initialization file. When the gateway server is running with commit-confirm enabled, it will connect to a local Oracle integrating server where it maintains a commit-confirm transaction log, similar to the Oracle two-phase commit log stored in the DBA_2PC_PENDING table. The gateway’s transaction log is stored in the PGA_CC_PENDING table. A row is stored in this table for each in-flight transaction and remains there until the transaction has completed. The life span of rows in PGA_CC_PENDING is normally quite short, lasting only from the time the commit is received by the gateway until the time the integrating server completes all commit processing and tells the gateway to forget the transaction.

  The commit-confirm gateway SID should be reserved for use only to invoke update transactions that implement commit-confirm. There is some extra overhead involved in the setup for logging when PGA_CAPABILITY is set to COMMIT_CONFIRM. Read-only transactions should be invoked through a separate gateway SID with PGA_CAPABILITY set to READ_ONLY so that they will not incur the extra overhead.

- **Oracle Logging Server**

  An Oracle server must be available for use by the gateway server for storing the PGA_CC_PENDING table. For maximum performance and reliability, Oracle Corporation recommends that this Oracle server reside on the same system as the gateway server.
Required Components

- OLTP Commit-Confirm Transaction Log
  A commit-confirm transaction log database must be defined to the OLTP system being accessed. This database must be recoverable and must be accessible by the OLTP as part of the same unit of work as the OLTP application’s databases, so that updates to the transaction log database will be kept in sync with updates to the application’s databases in a single unit of work.

  The commit-confirm transaction log database need contain only the Oracle Global Transaction ID and a date/time stamp. The Oracle Global Transaction ID is 169 bytes long and must be the key field. The date/time stamp is used for purging old entries that can be left in the log after certain failure scenarios.

  For simplicity, all commit-confirm applications under a particular OLTP should share the same commit-confirm transaction log.

- OLTP Transaction Logging Code
  Code must be added to each OLTP transaction invoked by a commit-confirm gateway to perform the transaction logging required by the gateway’s commit-confirm implementation. This code must receive the Oracle Global Transaction ID from the gateway and write that information into the OLTP commit-confirm transaction log database. For maximum flexibility and ease of use, this code can be written as a subroutine callable from any commit-confirm transaction on your OLTP system.

  This code must be executed at the beginning of each commit-confirm transaction prior to the first APPC receive and then immediately after each COMMIT or ROLLBACK in the transaction. This ensures that the logging is done at the beginning of each unit of work.

- OLTP Forget/Recovery Transaction
  A separate APPC transaction must be created on the OLTP system that can be started by the gateway to forget a transaction once it has been successfully committed and to query a transaction’s state during recovery processing. This transaction deletes the entry for a particular Oracle Global Transaction ID from the OLTP commit-confirm transaction log database during forget processing and queries the entry for a particular Oracle Global Transaction ID from the OLTP commit-confirm transaction log database during recovery processing.
Configuring Commit-Confirm

The steps for configuring commit-confirm include:

- configuring the Oracle integrating server where the gateway server will store its transaction log information
- configuring the gateway initialization parameters, and
- configuring the OLTP.

All of these steps must be performed before attempting to use any applications that use commit-confirm.

Configuring the Oracle Integrating Server

The Oracle integrating server where the gateway server will store its transaction log information should ideally be on the same system where the gateway runs. The configuration of the server consists of creating the gateway DBA user, creating the commit-confirm log tables and creating the PL/SQL stored procedure used by the gateway server for logging transactions.

The `pgaccau.sql` script from the `$ORACLE_HOME/pg4appc/admin` directory creates the gateway DBA user ID. The default user ID is PGADBA with the initial password set to PGADBA. If you want to change the user ID or initial password, you must modify the script.

1. Use SQL*Plus to connect to the Oracle integrating server as user SYSTEM.
2. From SQL*Plus, run the `pgaccau.sql` script from the `$ORACLE_HOME/pg4appc/admin` directory. This script creates the gateway DBA user ID. If you want to change the password at any time after running this script, you can use the ALTER USER command to change the password. For further information, refer to the *Oracle9i Server SQL Reference*.
3. Use SQL*Plus to connect to the Oracle integrating server as user PGADBA.
4. From SQL*Plus, run the `pgaccpnd.sql` script from the `$ORACLE_HOME/pg4appc/admin` directory. This script creates the PGA_CC_PENDING table used by the gateway server for its commit-confirm transaction log.
5. From SQL*Plus, run the `pgacclog.sql` script from the `$ORACLE_HOME/pg4appc/admin` directory. This script creates the PGA_CC_LOG PL/SQL stored procedure used by the gateway server for updating the PGA_CC_PENDING table.
6. Disconnect from the Oracle integrating server.

Configuring the Gateway Initialization Parameters

The gateway initialization parameters are discussed in Appendix A, "Gateway Initialization Parameters," of the Oracle Procedural Gateway for APPC Installation and Configuration Guide for your platform. The parameters necessary for commit-confirm support in the gateway are:

- **PGA_CAPABILITY**
- **PGA_LOG_DB**
- **PGA_LOG_USER**
- **PGA_LOG_PASS**
- **PGA_RECOVERY_USER**
- **PGA_RECOVERY_PASS**
- **PGA_RECOVERY_TPNAME**

These parameters should be added to your `init<sid>.ora` file, where `sid` is the gateway SID for your commit-confirm gateway.

---

**Caution:** Because the logging and recovery user IDs and passwords are specified in the `init<sid>.ora` file, Oracle Corporation recommends that the file permissions be set to remove read permissions for non-DBA users.

---

PGA_CAPABILITY must be set to COMMIT_CONFIRM to enable commit-confirm.

PGA_LOG_DB specifies a Oracle Net service name that is used by the gateway server to connect to the Oracle integrating server where the PGA_CC_PENDING table and the PGA_CC_LOG PL/SQL procedure are stored. The service name must be defined in a `tnsnames.ora` file that is accessible to the gateway server. The gateway server accesses the `tnsnames.ora` file through the TNS_ADMIN environment variable setting. TNS_ADMIN must specify the full path name of the directory in which the `tnsnames.ora` file is stored.

PGA_LOG_USER specifies the Oracle user ID that will be used by the gateway when connecting to the Oracle integrating server identified by the PGA_LOG_DB parameter. This user ID is the PGADBA user ID created by the `pgaccau.sql` script.
The user ID specified must be the same user ID under which the `pgaccpnd.sql` and `pgacclog.sql` scripts were run.

PGA_LOG_PASS specifies the Oracle password that is used by the gateway when connecting to the Oracle integrating server identified by the PGA_LOG_DB parameter. This is the password for the Oracle user ID specified by PGA_LOG_USER.

PGA_RECOVERY_USER specifies the user ID that is used by the gateway when allocating an LU6.2 conversation with the transaction specified by the PGA_RECOVERY_TPNAME parameter. This parameter is necessary only if the PGA_SECURITY_TYPE parameter has been set to "PROGRAM".

PGA_RECOVERY_PASS specifies the password that is used by the gateway when allocating an LU6.2 conversation with the transaction specified by the PGA_RECOVERY_TPNAME parameter. This parameter is necessary only if the PGA_SECURITY_TYPE parameter has been set to "PROGRAM".

PGA_RECOVERY_TPNAME specifies the TP name of the transaction installed in the OLTP for performing operations against the commit-confirm transaction log database on the OLTP side. The default value is RECO, which is the name used by the installation procedure for the sample transaction shipped with the gateway. Note that when the OLTP is CICS/ESA, the TP name is limited to four characters, and when the OLTP is IMS/TM, the TP name is limited to eight characters. Other OLTPs may have other limits on the length of the TP name.

**Configuring the OLTP**

Configuration of the OLTP includes the following:

- defining and installing the commit-confirm transaction log database
- defining and installing the commit-confirm forget/recovery transaction
- defining and installing the sample commit-confirm applications provided with the gateway

**Note:** A restart of the OLTP may be necessary to implement the changes required for commit-confirm support. You should plan for this with your OLTP system administrator.
Detailed instructions for configuring CICS/ESA and IMS/TM are provided in the $ORACLE_HOME/pg4appc/demo/CICS/README.doc and $ORACLE_HOME/pg4appc/demo/IMS/README.doc files, respectively.

**Application Design Requirements**

When designing commit-confirm applications for use with the Oracle Procedural Gateway for APPC, there are some requirements you must meet to provide the ability for the gateway to determine the state of a transaction in the event of a failure. If these requirements are not met, attempting to use an application with a commit-confirm gateway will produce unpredictable results.

The first thing that must be done by an OLTP transaction invoked by a commit-confirm gateway is to receive the Oracle Global Transaction ID from the gateway and log it into the OLTP commit-confirm transaction log database. This must be done before the normal data flow between the OLTP transaction and the Oracle application begins. The gateway always sends the Oracle Global Transaction ID as the very first data item.

If the OLTP transaction is a one-shot transaction, this is the only change needed. If the transaction is a persistent transaction that performs more than one unit of work (issues more than one commit or rollback), then a new Oracle Global Transaction ID must be received and logged after every commit or rollback.

The Oracle Global Transaction ID is sent by the gateway in a variable-length record with a maximum length of 202 bytes. The first 32 bytes contain a special binary string used to verify that the data came from the gateway and not from some other application. The next 1 byte is a reserved field. The Oracle Global Transaction ID is next, with a maximum length of 169 bytes. You must log the reserved field and the Oracle Global Transaction ID, as well as a date/time stamp and any other information you wish to log. Note that the Oracle Global Transaction ID must be the key field for the log database so that the forget/recovery transaction can use the Oracle Global Transaction ID to directly access a log entry.

**Note:** If your OLTP is IMS/TM, you must add a PCB for the commit-confirm transaction log database to the PSB for each transaction that you will use with a commit-confirm gateway. This PCB must be the first PCB in the PSB.
Commit-Confirm Architecture

The architecture of the commit-confirm implementation in the Oracle Procedural Gateway for APPC consists of three main components:

- Oracle integrating server
- Procedural Gateway for APPC server (gateway server)
- Oracle logging server

This section describes the role each component plays in the operation of commit-confirm and how these components interact.

Components

The Oracle integrating server is the controlling component in the commit-confirm architecture. It tells the gateway server when to commit a transaction and when to rollback a transaction. It does the same with all other servers participating in a distributed transaction. When a failure has occurred, it is the integrating server which drives the recovery process in each participating server, including the gateway server.

The gateway server performs the task of converting instructions from the Oracle integrating server into LU6.2 operations and then logs the transaction into the Oracle logging server. The gateway server stores the log information in a table called PGA_CC_PENDING on the logging server. If a failure occurs during transaction processing, the gateway server determines which error should be returned to the integrating server.

The Oracle logging server is an Oracle server available to the gateway server for storing and accessing its commit-confirm log information. The logging server need not be the same Oracle server as the integrating server, but can be. Because the logging server is an integral component of gateway commit-confirm operations, the best place for it to reside is on the same system as the gateway server. This allows the communication between the gateway server and the logging server to use interprocess communications, providing a high-speed, low overhead, local connection between the components.
Interactions

There is a specific set of interactions that occur between the components. They are:

- Oracle Integrating Server <---> Gateway Server

  The Oracle integrating server drives all actions by the gateway server. At the request of the Oracle application, the integrating server can instruct the gateway server to begin a new Oracle transaction, start a commit sequence, start a rollback sequence, or start a forget sequence. It can also call gateway remote procedural call (RPC) functions (PGAINIT, PGAXFER, PGATERM) on behalf of the Oracle application.

- Gateway Server <---> Oracle Logging Server

  The gateway server calls the Oracle logging server to insert and delete rows from its PGA_CC_PENDING table. This is actually done by calling a PL/SQL stored procedure, PGA_CC_LOG, in the logging server to reduce the number of open cursors required by the gateway server for performing its logging. Only a single cursor is needed by the gateway server for logging.

Commit-Confirm Flow

The flow of control for a successful commit between an Oracle application and an OLTP transaction is shown in Figure 5–1, “Commit-Confirm Flow with Synclevel 1”. This figure assumes that both Oracle and OLTP resources have been updated. The steps below outline the commit-confirm logic flow.

Commit-Confirm Logic Flow, Step by Step

1. The application issues a COMMIT to the Oracle integrating server.
2. The Oracle integrating server sends PREPARE to each participant in the distributed transaction other than the gateway.
3. Each participant prepares its database updates and responds PREPARE OK to the Oracle integrating server.
4. The Oracle integrating server sends COMMIT to the gateway. The gateway receives the COMMIT from the Oracle integrating server and inserts a new pending transaction row into the PGA_CC_PENDING table.
5. The gateway sends an APPC CONFIRM to the OLTP application. The OLTP application receives the CONFIRM request in the form of a status from the last APPC RECEIVE.
6. The OLTP application issues a COMMIT using an appropriate OLTP function. The OLTP commits all database updates made by the application since the last COMMIT, including the commit-confirm transaction log update.

7. Once the database updates have been committed, the OLTP returns control to the application with a return code indicating the status of the COMMIT.

8. The OLTP application sends an APPC CONFIRMED to the gateway.

9. The gateway receives the CONFIRMED and returns COMMIT OK to the Oracle integrating server.

10. The Oracle integrating server sends COMMIT to each participant in the distributed transaction other than the gateway.

11. Each participant commits its database updates and responds COMMIT OK to the Oracle integrating server.

12. The Oracle integrating server sends a FORGET to the gateway.

13. The gateway receives the FORGET and starts a new APPC conversation with the FORGET/RECOVERY transaction at the OLTP, sends it a FORGET request and an APPC CONFIRM. The FORGET/RECOVERY transaction receives the FORGET request and deletes the entry from the commit-confirm transaction log for the current Oracle transaction, and commits the delete.

14. The FORGET/RECOVERY transaction sends an APPC CONFIRMED to the gateway to indicate that the FORGET was processed, and then terminates. The gateway receives the CONFIRMED and deletes the pending transaction row from the PGA_CC_PENDING table.

15. The gateway returns FORGET OK to the Oracle integrating server.

16. The Oracle integrating server returns control to the Oracle application.
Figure 5–1  Commit-Confirm Flow with Syncllevel 1

Application

Oracle Integrating Server

Gateway

OLTP

OLTP App

1. COMMIT

2. PREPARE

3. COMMIT

4. CONFIRM

5. COMMIT

6. CONFIRM

7. CONFIRM

8. CONFIRM

9. COMMIT

10. COMMIT

11. COMMIT

12. COMMIT

13. COMMIT

14. COMMIT

15. COMMIT

16. COMMIT

Program Continues

Program Continues

Program Continues

Forget/Recover Transaction

Forget/Recover Transaction

Forget/Recover Transaction

Forget/Recover Transaction

Forget/Recover Transaction

Forget/Recover Transaction

Forget/Recover Transaction

Forget/Recover Transaction
Gateway Server Commit-Confirm Transaction Log

The commit-confirm transaction log consists of a single table, PGA_CC_PENDING. This table contains a row for each in-flight Oracle transaction that includes the commit-confirm gateway. The table is maintained by the gateway server and is similar in function to the Oracle server’s DBA_2PC_PENDING table. Note that a row is not inserted into this table until a COMMIT is received by the gateway and the row is deleted when a FORGET is received by the gateway. There is no involvement by the gateway during the PREPARE phase.

The PGA_CC_PENDING table contains the following columns:

- **GLOBAL_TRAN_ID**
  
  This is the Oracle Global Transaction ID for the transaction. It is identical to the corresponding column in the DBA_2PC_PENDING table.

- **SIDE_NAME**
  
  This is the Side Information Profile name that was used by the gateway to allocate the APPC conversation with the target LU. It corresponds to the SIDENAME parameter passed to the PGAINIT gateway function.

- **LU_NAME**
  
  This is the fully-qualified partner LU name of the target LU. This value is either the LU name from the Side Information Profile or the LUNAME parameter passed to the PGAINIT gateway function. This name fully identifies the OLTP system on which the transaction was executed.

- **MODE_NAME**
  
  This is the Mode name that was used by the gateway to allocate the APPC conversation with the target LU. The value is either the Mode name from the Side Information Profile or the MODENAME parameter passed to the PGAINIT gateway function.

- **TP_NAME**
  
  This is the transaction program name executed at the target LU. The value is either the TP name from the Side Information Profile or the TPNAME parameter passed to the PGAINIT gateway function. This name fully identifies the OLTP transaction program that was executed.
Sample Applications

Samples are provided with the gateway for CICS/ESA and IMS/TM for implementing commit-confirm support. They are in the following directories, respectively: 

- $ORACLE_HOME/pg4appc/demo/CICS
- $ORACLE_HOME/pg4appc/demo/IMS

A README.doc file in each directory provides detailed information about installing and using the samples. JCL files for compiling and linking the sample programs are provided as well. The samples included with the gateway assist you with the following:

- Creating and initializing the commit-confirm transaction log databases and defining those databases to the OLTP. For CICS/ESA, the sample uses a VSAM file for the log database. For IMS/TM, a SHISAM/VSAM database is used.

- Using subroutines for receiving the Oracle Global Transaction ID from the gateway and logging it into the commit-confirm transaction log database. These subroutines are provided in the pgacclg.asm files. They can be used in your applications to reduce the complexity of the code changes to your programs. For CICS/ESA, the subroutine provided is called using the EXEC CICS LINK interface. For IMS/TM, the subroutine provided is called using the standard CALL statement or its equivalent in your application's programming language. Both of these subroutines are written in 370 assembler to eliminate any inter-language interface complexities and compiler dependencies.

- Forget/recovery transactions. These are provided in the pgareco.asm files. Forget/recovery transactions must be installed into your OLTP and accessible through APPC so that the gateway can invoke them to forget a transaction once it has been successfully committed, and to query a transaction's state during recovery processing. These transactions delete the entry for a particular Oracle Global Transaction ID from the OLTP commit-confirm transaction log database during forget processing, and query the entry for a particular Oracle Global Transaction ID from the OLTP commit-confirm transaction log database during recovery processing. For both CICS/ESA and IMS/TM, these transactions are written in 370 assembler.

- Using the sample commit-confirm transaction log databases and subroutines. For CICS/ESA, a sample DB2 update transaction, DB2C, is provided in the pgadb2c.cob file. This is a COBOL example that updates the DB2 sample EMP table. For IMS/TM, a sample DLI update transaction, PGAIMSU, is provided in the pgaimsu.cob file. This is a COBOL example that updates the DLI sample PARTS database.
This chapter discusses diagnostic techniques and aids for determining and resolving problems with data conversion, truncation, and conversation startup. It also describes how to collect the data when the debugging (trace) option is on.

You will want to trace the PL/SQL stored procedures only when you suspect problems. Do not run with tracing enabled during normal operations, because it will affect performance.

This chapter contains the following sections:

- **TIP Definition Errors** on page 6-2
- **Problem Analysis with PG DD Diagnostic References** on page 6-3
- **Problem Analysis with PG DD Select Scripts** on page 6-4
- **Data Conversion Errors** on page 6-5
- **Problem Analysis of Data Conversion and Truncation Errors** on page 6-6
- **Problem Analysis with TIP Runtime Traces** on page 6-8
- **TIP Runtime Trace Controls** on page 6-10
- **Suppressing TIP Warnings and Tracing** on page 6-12
- **Gateway Server Tracing** on page 6-13
TIP Definition Errors

TIP definition errors occur when a TRANSACTION, CALL, or DATA entry in the PG DD is not properly defined.

Use the REPORT with DEBUG statement to list the PG DD contents and GENERATE DIAGNOSE(PKGEX(DR)) option to include corresponding ID numbers in the TIP.

The following table shows the mnemonic used to represent ID numbers and their correspondence with:
- PGAU REPORT with debug listings, GENERATE traces and TIPS
- PG DD tables and columns from which ID numbers are selected
- Oracle sequence objects from which ID numbers originate

<table>
<thead>
<tr>
<th>PGAU REPORT/TIP</th>
<th>PDGG table(col)</th>
<th>Sequence Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>v# transaction version</td>
<td>pga_trans(version)</td>
<td>pga.transvers</td>
</tr>
<tr>
<td>v# call version</td>
<td>pga_call(version)</td>
<td>pga.callvers</td>
</tr>
<tr>
<td>v# data version</td>
<td>pga_data(version)</td>
<td>pga.datavers</td>
</tr>
<tr>
<td>t# transaction id#</td>
<td>pga_trans(trans#)</td>
<td>pga.transeq</td>
</tr>
<tr>
<td>c# call id#</td>
<td>pga_call(call#)</td>
<td>pga.callseq</td>
</tr>
<tr>
<td></td>
<td>pga_call_parm(call#)</td>
<td></td>
</tr>
<tr>
<td>d# data id#</td>
<td>pga_call_parm(data#)</td>
<td>pga.dataseq</td>
</tr>
<tr>
<td></td>
<td>pga_data(data#)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pga_fields(data#)</td>
<td></td>
</tr>
<tr>
<td>f# field id#</td>
<td>pga_fields(fld#)</td>
<td>pga.fieldseq</td>
</tr>
<tr>
<td>q# qualifier id#</td>
<td>pga_data_values(qual#)</td>
<td>pga.fieldseq</td>
</tr>
<tr>
<td>a# trans attribute id#</td>
<td>pga_trans_values(attr#)</td>
<td>pga.tattrseq</td>
</tr>
<tr>
<td></td>
<td>pga_trans_attr(attr#)</td>
<td></td>
</tr>
<tr>
<td>a# field attribute id#</td>
<td>pga_data_values(attr#)</td>
<td>pga.dtattseq</td>
</tr>
<tr>
<td></td>
<td>pga_data_attr(attr#)</td>
<td></td>
</tr>
<tr>
<td>e# environment</td>
<td>pga_environments(env#)</td>
<td>pga.envrseq</td>
</tr>
<tr>
<td>l# compiler/language</td>
<td>pga_compilers(comp#)</td>
<td>pga.compseq</td>
</tr>
</tbody>
</table>
These ID numbers can be used to associate the conversions performed in the TIP with the definitions stored in the PG DD.

The PG DD diagnostic references appear in TIPs generated with the PKGEX(DR) option as single line Comments:

```
-- PG DD type idno=nnn ...
```

The PG DD diagnostic references appear in REPORT with DEBUG listings before or to the right of their related definition entry as end-delimited Comments:

```
/* idno=nnn */
```

Refer to Appendix A, "Procedural Gateway for APPC Data Dictionary" for more information about PG DD, including a complete list of dictionary tables.

**Problem Analysis with PG DD Diagnostic References**

TIPs should be generated by the PGAU GENERATE command with the PKGEX(DR) diagnostic option, to include PG DD reference Comments in the TIP. These diagnostic references are Comments only and do not affect the runtime overhead of the TIP. Refer to "GENERATE" in Chapter 2, "Procedural Gateway Administration Utility" for a description of the PKGEX (DR) parameter.

1. Before defining the PL/SQL package, identify the transaction name, ID number (t#), and version (v#) from the TIP specification within the TIP.

2. Invoke PGAU REPORT WITH DEBUG specifying the same transaction name and version.
   
   REPORT selects definitions from the PG DD and produces a listing showing the DATA, CALL, and TRANSACTION definitions and the ID number of each user-supplied definition.

3. Compare the reported definitions with those used in the remote transaction program and identify all corresponding APPC exchanges and the data formats transmitted.

4. Look for and investigate any mismatches.
   
   - different numbers of APPC send/receive calls
   - different sequence of APPC send/receive calls
   - different parameter lists on APPC send/receive calls
   - different data fields within each exchanged parameter
Problem Analysis with PG DD Select Scripts

- different lengths for each exchanged parameter
- unsupported datatypes for each exchanged parameter
- improperly initialized control fields for:
  - repeating group counts
    IBMVSCOBOLII affected clauses include
    OCCURS n TIMES DEPENDING ON field
  - remapped group criteria
    IBMVSCOBOLII affected clauses include
    REDEFINES field1 WHEN field2 = criteria

Problem Analysis with PG DD Select Scripts

PGAU GENERATE error messages and TRACE(OC) entries reference SQL SELECT statements by the name designations in the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>SED</td>
<td>Select Environment Data</td>
</tr>
<tr>
<td>STL</td>
<td>Select Transaction (latest version)</td>
</tr>
<tr>
<td>STV</td>
<td>Select Transaction (specific version)</td>
</tr>
<tr>
<td>STC</td>
<td>Select Transaction Calls</td>
</tr>
<tr>
<td>SPD</td>
<td>Select Parameter Data</td>
</tr>
<tr>
<td>SF</td>
<td>Select Fields</td>
</tr>
<tr>
<td>SFA</td>
<td>Select Field Attributes</td>
</tr>
<tr>
<td>SXF</td>
<td>Select conversion Formats</td>
</tr>
<tr>
<td>SXA</td>
<td>Select Attribute conversions</td>
</tr>
</tbody>
</table>
The SQL*Plus test scripts in the following table are provided to perform the identical SELECTS as GENERATE performs to determine which PG DD rows are being used when the TIP is generated. These files are loaded into the $ORACLE_HOME/pg4appc/admin directory during installation.

<table>
<thead>
<tr>
<th>Script</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>pgddsed.sql</td>
<td>Select Environment Data</td>
</tr>
<tr>
<td>pgddstl.sql</td>
<td>Select Transaction (latest version)</td>
</tr>
<tr>
<td>pgddstv.sql</td>
<td>Select Transaction (specific version)</td>
</tr>
<tr>
<td>pgddstc.sql</td>
<td>Select Transaction Calls</td>
</tr>
<tr>
<td>pgddspd.sql</td>
<td>Select Parameter Data</td>
</tr>
<tr>
<td>pgddsf.sql</td>
<td>Select Fields</td>
</tr>
<tr>
<td>pgddsfa.sql</td>
<td>Select Field Attributes</td>
</tr>
<tr>
<td>pgddsxf.sql</td>
<td>Select Conversion Formats</td>
</tr>
<tr>
<td>pgddsxa.sql</td>
<td>Select Attribute conversions</td>
</tr>
</tbody>
</table>

The scripts are shown in the same order used by GENERATE and each script prompts the SQL*Plus user for the required input. The information retrieved from a previous select is often used as input to a subsequent select. If a you suspect that a PG DD field entry has produced inaccurate data, browse the .sql files listed above to determine the source of the problem. These files are loaded into the $ORACLE_HOME/pg4appc/admin directory during installation.

Data Conversion Errors

Data conversion errors are usually the result of:

- incorrect determination of datatype

or

- incorrect specification of data position.

PGAU determination of the datatype is based on the values found in the PG DD, pga_fields(mask), and pga_fields(maskopts) columns. PGAU generates PL/SQL code to perform conversions based on the mask value:

- PIC X converted to CHAR
Problem Analysis of Data Conversion and Truncation Errors

using UTL_RAW CONVERT and UTL_RAW CAST_TO_VARCHAR2

- PIC G converted to CHAR
  using UTL_RAW CONVERT and UTL_RAW CAST_TO_VARCHAR2
- PIC 9 converted to NUMBER
  using UTL_PG NUMBER/RAW

Character datatype is presumed for all PIC X and PIC G mask values and conversion errors are more likely the result of position, length, and justification errors.

Determination of numeric datatype depends on several factors, including the combination of mask and maskopts values and how they apply to the actual remote host data in its internal format. Values for mask, maskopts, and data might conflict in unexpected ways. For example, an option such as USAGE IS COMP might be overridden if the data is in display format. While compilers occasionally perform such overrides correctly, they can cause unexpected results when exchanging data with systems coded in other languages.

To notify the user of such overrides, a warning function has been included in the following UTL_PG functions:

- MAKE_NUMBER_TO_RAW_FORMAT
- MAKE_RAW_TO_NUMBER_FORMAT
- NUMBER_TO_RAW
- RAW_TO_NUMBER

Problem Analysis of Data Conversion and Truncation Errors

Procedural Gateway for APPC data lengths are limited only by APPC and PL/SQL to 32,763 bytes per APPC exchange and PL/SQL variable.

The following steps can be used to diagnose data conversion or truncation errors.

Refer to Chapter 3, "Creating a TIP" to review the proper values and definitions referenced in items 1 through 4 below:

1. Ensure that the COBOL definitions used in the RHT match the input to PGAU;
2. Ensure the RHT transmission buffers are of sufficient length;
3. Ensure the RHT APPC call addresses the correct transmission buffer and uses the correct data length;
4. Ensure the client application has declared the correct TIP datatypes used as arguments in the TIP calls.

5. Ensure that the client application is calling the TIP functions in the proper sequence (init, user-defined..., term), and that any input data to the RHT is correct. Also ensure that if multiple user-defined functions exist, they are being called in the proper sequence and passed the correct input values, if any.

DBMS_OUTPUT calls can be inserted in the client application to trace its behavior.

For more information about calling TIP functions in proper sequence, refer to the chapter on configuring the Oracle integrating server for first time installations, in the Oracle Procedural Gateway for APPC Installation and Configuration Guide appropriate for your platform.

6. Optionally, regenerate the TIP with diagnostic traces included and enable them. The following traces are particularly useful:
   - data conversion trace
   - function entry/exit trace
   - gateway exchange trace

Refer to "Problem Analysis with TIP Runtime Traces" on page 6-8 for more information about traces; refer also to "GENERATE" in Chapter 2, "Procedural Gateway Administration Utility".

The gateway server tracing must also be enabled in $ORACLE_HOME/pg4appc/admin/initsid.ora. Set the parameters

SET TRACE_LEVEL=255 and
SET LOG_DESTINATION=/oracle/pga/9.2.0/pg4appc/log.

Refer to "Gateway Server Tracing" on page 6-13 and to Appendix A "Gateway Initialization Parameters" in your Oracle Procedural Gateway for APPC Installation and Configuration Guide for more information about these parameters.

Rerun the client application and examine the trace (see the next step for details).

To disable the trace, reset TRACE_LEVEL=0.

7. Examine the trace output.

The TIP trace output can be saved in a spool file, such as:

spool tipname.trc
TIP trace output is written to a named DBMS_PIPE and can be retrieved under SQL*Plus by issuing the following command:

```sql
exec rtrace('tipname');
```
or it can be purged by issuing the following command:

```sql
exec ptrace('tipname');
```

**Note:** `tipname` is case-sensitive and must be specified exactly as it is in the TIP.

Gateway server trace output is written to a log file in a default directory path specified by the SET LOG_DESTINATION gateway parameter in `$ORACLE_HOME/pg4appc/admin/initsid.ora`. For example:

```sql
SET LOG_DESTINATION=/oracle/pga/9.2.0/pg4appc/log/
```

Refer to "Gateway Server Tracing" on page 6-13 for more information.

The gateway server log file can be viewed by editing the file or by issuing other system commands that display file contents. The log file can also be copied and saved to document problem symptoms.

---

**Problem Analysis with TIP Runtime Traces**

TIPs should be generated by the PGAU GENERATE command with the PKGEX(DC) diagnostic option to include TIP data conversion trace logic in the TIP. TIP function call trace logic is always included in every TIP. This is runtime trace instrumentation and has some overhead when tracing is enabled, but negligible overhead when tracing is disabled. Refer to "GENERATE" in Chapter 2, "Procedural Gateway Administration Utility" for more information.

1. Regenerate TIPs with the PKGEX(DC,DR) options and recompile the TIP body file, `tipname.pkb`. Avoid recompiling the TIP specification.
2. Revise the application that calls the TIP initialization function (`tipname_init`) to pass the trace flags parameter with data conversion and function call tracing enabled. Refer to "Controlling TIP Runtime Data Conversion Tracing" on page 6-11.

If the problem causes an exception to be raised in the TIP and the application contains an exception handler, the application exception handler should be Commented out to prevent it from handling the exception and preventing the
exception point of origin from being reported. When the TIP exception is next raised, its source line number in the TIP is reported. Record this information.

Note: Before you execute Step 3, review Step 4 below and be prepared to execute it before starting the application, to avoid the problem referred to in Step 3.

3. Execute the application with diagnostic TIP initialization.

If the TIP trace pipe inlet overflows due to the application calls causing the TIP to write trace messages in the TIP trace pipe inlet, you have one minute from the start of the overflow condition to begin Step 4 and empty the TIP trace pipe. Otherwise, exception "ORA-20703 PGA-TIP: pipe send error" is issued, ending the diagnostic session, possibly before any relevant trace information is generated.

4. Retrieve and record the TIP trace message stream.

Use SQL*Plus to connect to the same Oracle user ID executing the application or the user ID under which the TIP is executed. This establishes a second session from which the trace pipe outlet can be read, preventing the TIP trace pipe from overflowing at the TIP trace pipe inlet.

a. Issue the command:

   set serveroutput on size nnnnn

b. Issue the command to record the trace output:

   spool tipname.trc

c. Issue the command to retrieve the trace stream:

   exec rtrace('tipname');

   If the application is long-running, repeat this command as often as needed until all trace messages have been retrieved.

5. If any exceptions are raised, note their prefix, number, and full message text.

6. Analyze the TIP trace message stream. A normal trace is shown for the pgadb2i TIP in Appendix E, "Administration Utility Samples".
TIP Runtime Trace Controls

Runtime trace control is the second parameter specified on a TIP initialization call. It is a CHAR(8) datatype of the following form:

```c
rc := yourtip_init(trannum,'wxyz0000');
```

The following table describes position one to four:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position 1 (w)</td>
<td>controls UTL_RAW warning. A value of 0 suppresses warnings; a value of 1 issues warnings.</td>
</tr>
<tr>
<td>position 2 (x)</td>
<td>controls the function entry/exit tracing. A value of 0 suppresses the function entry/exit tracing; a value of 1 enables the function entry/exit tracing.</td>
</tr>
<tr>
<td>position 3 (y)</td>
<td>controls data conversion tracing. A value of 0 suppresses data conversion tracing; a value of 1 enables data conversion tracing.</td>
</tr>
<tr>
<td>position 4 (z)</td>
<td>controls gateway exchange tracing. A value of 0 suppresses gateway exchange tracing; a value of 1 enables gateway exchange tracing.</td>
</tr>
</tbody>
</table>

Positions 5 through 8 are reserved and ignored.

Generating Runtime Data Conversion Trace and Warning Support

Use PGAU to regenerate the TIP and specify the GENERATE parameter DIAGNOSE(PKGEX(DC)). This includes runtime PL/SQL code in the TIP which tests for and displays warnings of correct, but possibly unexpected NUMBER_TO_RAW and RAW_TO_NUMBER conversions.

Refer to "GENERATE" in Chapter 2, "Procedural Gateway Administration Utility" for more information about this parameter.

Recompile the TIP body under SQL*Plus. Avoid recompiling the TIP specification.
Controlling TIP Runtime Conversion Warnings

After the TIP has been regenerated, the issuance of runtime warnings is under control of the application. By default, warnings are suppressed and are only issued when they are enabled.

Errors and exceptions are always issued if they occur.

To enable the issuance of warnings, an additional parameter must be supplied when calling the TIP initialization function. This parameter is a CHAR(8) datatype and each character position controls a particular TIP runtime diagnostic function.

To enable warnings in yourtip, the client application should call the TIP initialization function with the statement:

```
rc := yourtip_init(trannum,'10000000');
```

The following is input to the TIP trace pipe inlet at initialization time:

"UTL_PG warnings enabled"

Controlling TIP Runtime Function Entry/Exit Tracing

To enable function entry/exit tracing in yourtip, the client application should call the TIP initialization function with the statement:

```
rc := yourtip_init(trannum,'01000000');
```

The following is input to the TIP trace pipe inlet at initialization time:

"function entry/exit trace enabled"
"tipname_init entered"
"time date/time stamp"

Controlling TIP Runtime Data Conversion Tracing

To enable data conversion tracing in yourtip, the client application should call the TIP initialization function with the following statement:

```
rc := yourtip_init(trannum,'00100000');
```

The following is input to the TIP trace pipe inlet at initialization time:

"data conversion trace enabled"
Controlling TIP Runtime Gateway Exchange Tracing

To enable runtime gateway exchange tracing in yourtip, the client application should call the TIP initialization function with the following statement:

\[
rc := \text{yourtip\_init}(\text{trannum}, '00010000');
\]

The following is input to the TIP trace pipe inlet at initialization time:

‘gateway exchange trace enabled’

Suppressing TIP Warnings and Tracing

After debugging is finished, there are two ways to suppress the following:

- data conversion tracing
- conversion warnings
- function entry/exit tracing
- gateway exchange tracing

You can:

a. Call the TIP initialization function without passing any diagnostic control parameters:

\[
rc := \text{yourtip\_init}(\text{trannum});
\]

b. Call the TIP initialization function passing a revised diagnostic control parameter which disables all tracing and warnings:

\[
rc := \text{yourtip\_init}(\text{trannum}, '00000000');
\]

A third method, described in Method C, removes the logic for:

- data conversion tracing
- conversion warnings

  c. Generate the TIP again without:

  \[
  \text{PKGEX(DC)}
  \]

Or you can recompile the previous version of the TIP body if it was saved.

Methods A and B allow you to use the same TIP without alteration, but without tracing or warnings. These methods are reversible without alteration or replacement of the TIP. Tracing and warnings can be redisplayed should a problem recur.
Method C also suppresses data conversion tracing and warnings and incurs reduced overhead by avoiding tests, but is not reversible without regenerating the TIP or recompiling an alternate version with data conversion tracing and warning diagnostics imbedded.

The logic for function entry/exit and gateway exchange tracing is included in every TIP and cannot be removed. It can be disabled by method A or B.

**Gateway Server Tracing**

The gateway contains extensive tracing logic in the gateway remote procedural calls (RPCs), and the APPC-specific code. Tracing is enabled through gateway initialization parameters or dynamic RPC calls to the gateway. The trace provides information about the execution of the gateway RPC functions and about the execution of the APPC interface. The trace file contains a text stream written in chronological sequence of events. The trace is designed to assist application programmers with the debugging of their OLTP transaction programs and Oracle applications that communicate with those transaction programs through the gateway.

A single trace file is created for an entire gateway session from the time the database link is opened until it is closed. The trace can be directed to a specific path/filename or to a path (directory) only. In the first case, the file is overwritten each time a new session begins for the gateway being traced. When the trace target is a directory, a separate file with a generated name (containing the operating system process ID) is written for each gateway session. The latter approach must be used whenever the gateway to be traced might be the target of new sessions after the desired trace is written but before it can be copied and saved. Conversely, in some situations you might choose to create a distinct gateway system identifier used solely for tracing, and direct its trace to a single specific filename. This avoids the problem of an ever-increasing set of trace files when, for example, repeated attempts are necessary to reproduce or debug a problem. A fixed filename should never be used if there is any chance that an unexpected gateway session could overlay a useful trace.

**Defining the Gateway Trace Destination**

This section describes how to define the destination of trace files to the gateway, and how to cause the gateway to create the trace files during initialization. Note that this does not enable any gateway tracing, it merely defines the destination of any trace output produced when the gateway tracing is enabled.
1. Choose a gateway system identifier to trace. Decide whether you will be tracing an existing gateway system identifier or a new one created specifically for tracing. If a new system identifier will be used, configure the new system identifier exactly the same as the old one by creating a new `initsid.ora` (a copy of the old), entries in `listener.ora` as necessary, and a new Oracle database link. Test the new system identifier to ensure it works before proceeding.

2. In `/oracle/pga/9.2.0/bin`, edit the `initsid.ora` file so it contains the following:

   ```
   SET TRACE_LEVEL=255
   SET LOG_DESTINATION=logdest
   ```
   
   where `logdest` is the directory path or full filename for the trace output. The logfile is usually in `/oracle/pga/9.2.0/pg4appc/admin`. Refer to the earlier discussion about “Problem Analysis of Data Conversion and Truncation Errors” on page 6-6 for more information.

---

**Note:** Misspelled parameter names in `initsid.ora` are not detected. The parameter is ignored.

---

Once these two steps are completed, the gateway opens the specified trace file during initialization. Each session on this system identifier writes a trace file as specified by the SET LOG_DESTINATION parameter described in Step 2 above. If a directory path was specified, each trace file has a name of the form:

```
    sid_pid.log
```

where `sid` is the gateway sid and `pid` is the operating system process ID of the gateway server expressed in decimal.

---

**Enabling the Gateway Trace**

There are two ways to enable the gateway server tracing. The first is to set the tracing options in the gateway initialization file, `initsid.ora`. The second is to use the additional PGA remote procedural call (RPC) function, PGATCTL, to dynamically control the tracing from within the Oracle application. The first method causes tracing to be performed for all users of the gateway system identifier and is recommended only when the use of the gateway system identifier can be limited to users actually needing the trace. The second method is more flexible and allows the
application programmer to selectively trace events on a single gateway session without affecting the operation of other users’ gateway sessions.

Before the gateway server trace is enabled, perform the tasks listed in "Defining the Gateway Trace Destination" on page 6-13.

**Enabling the APPC Trace Using Initialization Parameters**

1. Edit the `initsid.ora` file, and add the following line at the end of the file (or, if a `SET TRACE_LEVEL` parameter is already specified, modify it):

   ```
   SET TRACE_LEVEL=trace
   ```

   where `trace` is a numeric value from 1 to 255 indicating which traces are to be enabled. For further information on the use of this parameter, refer to "PGA Parameters" in Appendix A of the *Oracle Procedural Gateway for APPC Installation and Configuration Guide*.

   Once this step is completed, tracing is enabled for the desired gateway system identifier.

**Enabling the APPC Trace Dynamically from PL/SQL**

The following is only needed for user-written TIPs. PGAU-generated TIPs automatically include the following facilities. Refer to "Controlling TIP Runtime Gateway Exchange Tracing" on page 6-12 for more information.

Make the following changes to the PL/SQL application that calls the Transaction Interface Package(s) to execute remote transaction(s).

1. Add a call to `PGATCTL` before any calls to TIP initialization functions are made:

   ```
   PGATCTL@dblink(HEXTORAW('000000000000000000000000'),
   'S', traces);
   ```

   where `dblink` is the name of the database link to the gateway and `traces` specifies which traces are to be enabled, as described previously in the discussion of the SET TRACE_LEVEL initialization parameter.

   This call sets the trace flags for all new conversations started after the call to the value specified by `traces`.

2. Recompile the PL/SQL application to pick up the new trace call.
Sample Gateway Server Initialization and Trace Output Files

The following is an example of the trace output produced by the gateway server where:

- the TRACE_LEVEL parameter is set to 255
- the gateway sid is PGASNIX92
- the Oracle home directory is /avail/stalmoud/T920
- the PGA_CAPABILITY parameter is set to COMMIT_CONFIRM
- the gateway server was invoked by a Oracle Net database link from another system

HGAINIT: entered.
  : LOG_DESTINATION => /avail/stalmoud/T920/pg4appc/log/PGASNIX92_20209.log
  : TRACE_LEVEL => 255
  ORACLE_HOME: /avail/stalmoud/T920, LEN <20>, MAX_LEN <256>.
  ORACLE_SID: PGASNIX92, LEN <9>, MAX_LEN <256>.
  FDS_INSTANCE: PGA, LEN <3>, MAX_LEN <256>.
  FDS_CLASS_VERSION: 2, LEN <1>, MAX_LEN <30>.
  : PGA_CAPABILITY = COMMIT_CONFIRM
HGAINIT: exiting, rc = 0
HGALGON: entered
  : reco    = 0
  : namel   = 8
  : name    = PGAADMIN
  : passwdl = 8
  : passwd  = pgaadmin
  : gpau is at 13e53c
  : PGA_CAPABILITY = COMMIT_CONFIRM
  : up->gpauicap = 3
  : up->gpauflg = 00
  : PGA_LOG_USER = regpga
  : PGA_LOG_PASS = regpga
  : PGA_LOG_DB = R901
  : PGA_RECOVERY_TPNAME = RECO
  : ORACLE_SID = PGASNIX92
SGTAINI: entered
SGTAINI: exiting, rc = 0
GPALLON: entered
gpal is at 13dd48
  logging onto Oracle9
OCIInitialize, rc = 0
OCIEnvInit, rc = 0
OCIHandleAlloc -- STMT  rc = 0
OCIHandleAlloc -- ERROR,  rc = 0
logon complete, rc = 0
GPALLON: exiting, rc= 0
GPALLOG: entered
Preparing the statement
OCIStmtPrepare, rc = 0
binding variable :func[1]
OCIBindByName complete, rc = 0
binding variable :gtxid[169]
OCIBindByName complete, rc = 0
binding variable :profile[8]
OCIBindByName complete, rc = 0
binding variable :luname[17]
OCIBindByName complete, rc = 0
binding variable :modename[8]
OCIBindByName complete, rc = 0
binding variable :tpname[64]
OCIBindByName complete, rc = 0
GPALLOG: exiting, rc= 0
HGALGON: exiting, rc = 0
HGAUPLC: entered
HGAUPLC: exiting, rc = 0
HGAUPLD: entered
HGAUPLD: exiting, rc = 0
HGABEGN: entered, hgagt = efffe72c
GPALBGN: entered
lp->gpalgtit = 29
GPALBGN: exiting
HGABEGN: exiting, rc = <0
HGAPDSC: entered
HGAPFND: entered
HGAPFND: exit - procedure 'PGAXFER' was found
HGAPDSC: exiting, rc = <0
HGAPFND: entered
HGAPFND: exit - procedure 'PGAINIT' was found
HGAPDSC: exiting, rc = <0
HGAPFND: entered
HGAPFND: exit - procedure 'PGATCTL' was found
HGAPDSC: exiting, rc = <0
HGAPFND: entered
HGAPFND: entered
Sample Gateway Server Initialization and Trace Output Files

HGAPFND: exit - procedure 'PGATCTL' was found
HGAPDSC: exiting, rc = <0>
HGADAFR: entered
HGADAFR: exit
HGAPDSC: entered
HGAPFND: entered
HGAPFND: exit - procedure 'PGTERM' was found
HGAPDSC: exiting, rc = <0>
HGACOMM: entered, keepinfo = 0
HGACOMM: exiting, rc = 0
HGABEGN: entered, hgapf = effe72c
GPALBGN: entered
1p->gpalgtil = 28
GPALBGN: exiting
HGABEGN: exiting, rc = <0>
HGAPDSC: entered
HGAPFND: entered
HGAPFND: exit - procedure 'PGAINIT' was found
HGAPDSC: exiting, rc = <0>
HGADAFR: entered
HGADAFR: exit <0>
HGAPEXEC: entered
HGAPFND: entered
HGAPFND: exit - procedure 'PGAINIT' was found
GPACINI: entered
GPACLOC: executed
GPACLOC: exit, rc = 0
PGAINIT: TP name = 'DB2C'
PGAINIT: LU name = ''
PGAINIT: Mode name = 'ORAPLU62'
PGAINIT: Profile name = 'CICSPGA'
PGAINIT: Sync level = 1
PGAINIT: Userid = PGAADMIN
PGAINIT: passwd = pgaadmin
GPACAECl: entered
GPACAECl: exiting, rc = 0
GPAC allocated at 198d20
SGPAALL: entered
SGPA allocated at 198ea0
CMINIT issued: profile = CICSPGA
CMINIT complete: rc = 0, errno = 11
CMENLN issued
CMENLN complete: rc = 0, errno = 11, luname = CICSPGA
CMSTPN issued: tname = DB2C
CMSTPN complete: rc = 0, errno = 11
Sample Gateway Server Initialization and Trace Output Files

CMSMN issued: modename = ORAPLU62
CMSMN complete: rc = 0, errno = 11
XCSCST issued: security type = 0
XCSCST complete: rc = 0, errno = 11
CMSSL issued: synclevel = 1
CMSSL complete: rc = 0, errno = 11
CMSPTR issued: type = 1
CMSPTR complete: rc = 0, errno = 11
CMALLC issued
CMALLC complete: rc = 0, errno = 11
SGPAALL: exiting, rc = 0
  gpaucps set to 198d20
GPACAEC: entered
GPACAEC: exiting, rc = 0
The Dump in ASCII
PGAINIT: Conversation id - @ 198e10
  0000:  00000001 01000000 00000000           *............*
And the same dump in EBCDIC
PGAINIT: Conversation id - @ 198e10
  0000:  00000001 01000000 00000000           *............*
GPACINI: exiting rc = <0>
HGAPXE: exiting, rc = <0>
HGAPDSC: entered
HGAPFND: entered
HGAPFND: exit - procedure 'PGAXFER' was found
HGAPDSC: exiting, rc = <0>
HGADAFR: entered
HGADAFR: exit <0>
HGAPXE: entered
HGAPFND: entered
HGAPFND: exit - procedure 'PGAXFER' was found
GPACXFR: entered
GPACLOC: executed
GPACLOC: exit, rc = 0
  GPAU found at 13e53c
  GPAC found at 198d20
  sbufsz = 6
  rbufsz = 32763
  slensz = 8
  rlensz = 8
The Dump in ASCII
PGAXFER: Conversation id - @ efffe454
  0000:  00000001 01000000 00000000           *............*
And the same dump in EBCDIC
PGAXFER: Conversation id - @ efffe454
0000: 00000001 01000000 00000000 ..........
PGAXFER: Send buffer length = 6
PGAXFER: Receive buffer length = 91
The Dump in ASCII
PGAXFER: Send lengths array - @ efff6350
0000: 00000001 00000006 ........
And the same dump in EBCDIC
PGAXFER: Send lengths array - @ efff6350
0000: 00000001 00000006 ........
The Dump in ASCII
PGAXFER: Send lengths array - @ efff6248
0000: 00000001 00000005B ........[
And the same dump in EBCDIC
PGAXFER: Send lengths array - @ efff6248
0000: 00000001 00000005B ........$
The Dump in ASCII
PGAXFER: Send buffer - @ efff6458
0000: F0F0F0F3 F4F0 ........
And the same dump in EBCDIC
PGAXFER: Send buffer - @ efff6458
0000: F0F0F0F3 F4F0 000340*
GPACAEC: entered
GPACAEC: exiting, rc = 0
  sends = 1
  receives = 1
SGPASND: entered, type = 0
SGPARTS: entered, deal = 0
SGPARTS: exiting, rc = 0, deal = 0
  CMSST issued: send type = 0
  CMSST complete: rc = 0, errno = 11
The Dump in ASCII
Sending data: - @ 13dd48
0000: 50473441 50504320 00DEADBE EF00DEAD *PG4APPC ........*
0010: BEEF00DE ADBEEF00 50473441 50504320 ..........PG4APP* 
0020: 00523930 312E5444 4E415941 2E356131 *.R901.TENAYA.5a1*
0030: 393066662 652E372E 312E3433 38 *90bfe.7.1.438*
And the same dump in EBCDIC
Sending data: - @ 13dd48
0000: 50473441 50504320 00DEADBE EF00DEAD *50473441 50504320 00DEADBE EF00DEAD *.R901.TENAYA.5a1* 
0010: BEEF00DE ADBEEF00 50473441 50504320 ..........PG4APP* 
0020: 00523930 312E5444 4E415941 2E356131 *.R901.TENAYA.5a1*
0030: 393066662 652E372E 312E3433 38 *90bfe.7.1.438*
CMSEND issued: length = 61
CMSEND complete: rc = 0, errno = 11, rts = 0
SGPASND: exiting, rc = 0
Sample Gateway Server Initialization and Trace Output Files

slens[1] = 6
SGPASND: entered, type = 2
SGPARTS: entered, deal = 0
SGPARTS: exiting, rc = 0, deal = 0
CMSST issued: send type = 3
CMSST complete: rc = 0, errno = 11

The Dump in ASCII
Sending data: @ efff6458
  0000: F0 F0 F0 F3 F4 F0
And the same dump in EBCDIC
Sending data: @ efff6458
  0000: F0 F0 F0 F3 F4 F0
CMSSEND issued: length = 6
CMSSEND complete: rc = 0, errno = 11, rts = 0
SGPASND: exiting, rc = 0
rlens[1] = 91
SGPARCV: entered
CMECS issued
CMECS complete: rc = 0, errno = 11, state = 4
CMRCV issued: length = -268476352
CMRCV complete: rc = 0, errno = 11, rts = 0, dr = 2, sr = 1, length = 91

The Dump in ASCII
Received data: @ efff6458
  0000: F0 F0 F0 F3 F4 F0 0000 0000 0005 D1 C1 E2 D6 D5 00 00 00 00 00
  0010: 00 00 00 5B
And the same dump in EBCDIC
Received data: @ efff6458
  0000: F0 F0 F0 F3 F4 F0 0000 0000 0005 D1 C1 E2 D6 D5 00 00 00 00 00
  0010: 00 00 00 5B
PGAXFER: Receive buffer length = 91

The Dump in ASCII
PGAXFER: Receive lengths array - @ efff6458
  0000: 00 00 00 00 00 00 00 00 00 05 00
And the same dump in EBCDIC
PGAXFER: Receive lengths array - @ efff6458
The Dump in ASCII
PGAXFER: Receive buffer - @ 175858
0000: 00000001 00000005 01000000 00000000...*
0010: 800C4277 D90006C7 D8E4D5D6 E3F006D8...*
0020: 800C4112 000C5F2 F1F5F6F9 F8F1F9F4...*
0030: F760F0F5 60F0F5C6 9C5D3C4 D9C5D700...*
0040: 10D4F1F9 F2F660F0 F560F1F7 00238400...*
0050: 0000050 000C0001 90700C...*
And the same dump in EBCDIC
PGAXFER: Receive buffer - @ 175858
0000: F0F0F0F3 F4F00005 01000000 00000000...*
0010: 800C4277 D90006C7 D8E4D5D6 E3F006D8...*
0020: 800C4112 000C5F2 F1F5F6F9 F8F1F9F4...*
0030: F760F0F5 60F0F5C6 9C5D3C4 D9C5D700...*
0040: 10D4F1F9 F2F660F0 F560F1F7 00238400...*
0050: 0000050 000C0001 90700C...*

And the same dump in EBCDIC
PGAXFER: Conversation id - @ efffe454
0000: 0000001 01000000 00000000...*
And the same dump in EBCDIC
PGAXFER: Conversation id - @ efffe454
0000: 0000001 01000000 00000000...*
PGAXFER: Send buffer length = 91
PGAXFER: Receive buffer length = 4
The Dump in ASCII
PGAXFER: Send lengths array - @ efffe6350
0000: 0000001 0000005B...*
And the same dump in EBCDIC
PGAXFER: Send lengths array - @ efff6350
  0000: 00000001 0000005B *........* 

The Dump in ASCII
PGAXFER: Receive lengths array - @ efff6248
  0000: 00000001 00000004 *........* 

And the same dump in EBCDIC
PGAXFER: Receive lengths array - @ efff6248
  0000: 00000001 00000004 *........* 

The Dump in ASCII
PGAXFER: Send buffer - @ efff6458
  0000: F0F0F0F3 F4F00005 D1C1E2D6 D5000000 *................* 
  0010: 800C4277 D90006C7 D6E4D5D6 E3F0ABD8 *..Bw........* 
  0020: 800C4112 0000E3E2 E3F5F6F9 F8F1F9F4 *..A........* 
  0030: F760F0F5 60F0F5C6 C9C5D3C4 D9C5D700 *.
   *........* 
  0040: 10D4F1F9 F2F660F0 F560F1F7 00238400 *......'..#..* 
  0050: 0C000050 000C0001 90700C *...P.....p.* 

And the same dump in EBCDIC
PGAXFER: Send buffer - @ efff6458
  0000: F0F0F0F3 F4F00005 D1C1E2D6 D5000000 *000340..JASON...* 
  0010: 800C4277 D90006C7 D6E4D5D6 E3F0ABD8 *..R...GOUNOTO.Q* 
  0020: 800C4112 0000E3E2 E3F5F6F9 F8F1F9F4 *..TST5698194* 
  0030: F760F0F5 60F0F5C6 C9C5D3C4 D9C5D700 *7-05-05FIELDREP.* 
  0040: 10D4F1F9 F2F660F0 F560F1F7 00238400 *.M1926-05-17..d.* 
  0050: 0C000050 000C0001 90700C *...&.......* 

GPACAEC: entered
GPACAEC: exiting, rc = 0
sends = 1
receives = 1
slens[1] = 91
SGPASND: entered, type = 2
SGPARTS: entered, deal = 0
SGPARTS: exiting, rc = 0, deal = 0
CMSST issued: send type = 3
CMSST complete: rc = 0, errno = 11
The Dump in ASCII
Sending data: - @ efff6458
  0000: F0F0F0F3 F4F00005 D1C1E2D6 D5000000 *................* 
  0010: 800C4277 D90006C7 D6E4D5D6 E3F0ABD8 *..Bw........* 
  0020: 800C4112 0000E3E2 E3F5F6F9 F8F1F9F4 *..A........* 
  0030: F760F0F5 60F0F5C6 C9C5D3C4 D9C5D700 *.
   *........* 
  0040: 10D4F1F9 F2F660F0 F560F1F7 00238400 *......'..#..* 
  0050: 0C000050 000C0001 90700C *...P.....p.* 

And the same dump in EBCDIC
Sending data: - @ efff6458
  0000: F0F0F0F3 F4F00005 D1C1E2D6 D5000000 *000340..JASON...*
Sample Gateway Server Initialization and Trace Output Files

0010:  800C4277 D90006C7 D6E4D5D6 E3F0ABD8 *....R..GOUNOT0.Q*
0020:  800C4112 0000E3E2 E3F5F6F9 F8F1F9F4 *......TST5698194*
0030:  F760F0F5 60F0F5C6 C9C5D3C4 D9C5D700 *7-05-05FIELDREP.*
0040:  10D4F1F9 F2F660F0 F560F1F7 002348400 *.M1926-05-17..d.*
0050:  0000050 0000001 90700C *....&.......
CMSEND issued: length = 91
CMSEND complete: rc = 0, errno = 11, rts = 0
SGPASND: exiting, rc = 0
   rlen[1] = 4
SGPARCV: entered
   CMECS issued
   CMECS complete: rc = 0, errno = 11, state = 4
   CMRCV issued: length = -268476352
   CMRCV complete: rc = 0, errno = 11, rts = 0, dr = 2, sr = 1, length = 4
The Dump in ASCII
Received data: - @ 166ca8
   0000:  00000000 *....*
And the same dump in EBCDIC
Received data: - @ 166ca8
   0000:  00000000 *....*
   conversation state is now SEND
SGPARCV: exiting, rc = 0
PGAXFER: Receive buffer length = 4
The Dump in ASCII
PGAXFER: Receive lengths array - @ efff6248
   0000:  00000001 00000004 *...*.*
And the same dump in EBCDIC
PGAXFER: Receive lengths array - @ efff6248
   0000:  00000001 00000004 *...*.*
The Dump in ASCII
PGAXFER: Receive buffer - @ 166ca8
   0000:  00000000 *....*
And the same dump in EBCDIC
PGAXFER: Receive buffer - @ 166ca8
   0000:  00000000 *....*
GPACAEC: entered
GPACAEC: exiting, rc = 0
GPACXFR: exiting rc = <0>
HGAPEXE: exiting, rc = <0>
HGACOMM: entered, keepinfo = 1
GPALIPT: entered
   gpalfun = I
   gpalglti = R901.TENAYA.5a190fbe.7.1.438
   len = 28
   gpalprf = CICSPGA
len = 8
gpalln = CICSPGA
len = 7
gpalmdn = ORAPLU62
len = 8
gpaltpn = DB2C
len = 4
OCISstmtExecute complete, rc = 0
GPALIPT: exiting, rc= 0
SGPACTL: entered, type = 0
SGPARTS: entered, deal = 0
SGPARTS: exiting, rc = 0, deal = 0
CMCFM issued
CMCFM complete: rc = 0, errno = 11, rts = 0
SGPACTL: exiting, rc = 0
HGACOMM: exiting, rc = 0
HGAFORG: entered
SGPHAALL: entered
SGPA allocated at 1bbef8
CMINIT issued: profile = CICSPGA
CMINIT complete: rc = 0, errno = 11
CMSPLN issued: luname = CICSPGA
CMSPLN complete: rc = 0, errno = 11
CMSTPN issued: tpname = RECO
CMSTPN complete: rc = 0, errno = 11
CMSMN issued: modename = ORAPLU62
CMSMN complete: rc = 0, errno = 11
XCSCST issued: security type = 0
XCSCST complete: rc = 0, errno = 11
CMSSL issued: synclevel = 1
CMSSL complete: rc = 0, errno = 11
CMSPTR issued: type = 1
CMSPTR complete: rc = 0, errno = 11
CMALLC issued
CMALLC complete: rc = 0, errno = 11
SGPAALL: exiting, rc = 0
SGPASND: entered, type = 3
SGPARTS: entered, deal = 0
SGPARTS: exiting, rc = 0, deal = 0
CMSSST issued: send type = 2
CMSSST complete: rc = 0, errno = 11
The Dump in ASCII
Sending data: - @ 13dd68
0000:  01523930 312E5444 4E415941 2E356131  *.R901.TENAYA.5a1*
0010: 39306662 652E372E 312E3433 38         *90fbe.7.1.438*
And the same dump in EBCDIC
Sending data: - @ 13dd68
 0000: 01523930 312E5445 4E415941 2E356131  *........../.*
 0010: 39306662 652E372E 312E3433 38         *.............*
CMSEND issued: length = 29
  CMSEND complete: rc = 0, errno = 11, rts = 0
  SGPASND: exiting, rc = 0
  SGPADEA: entered, type = 0
  SGPARTS: entered, deal = 1
  SGPARTS: exiting, rc = 0, deal = 0
    CMSDT issued: deallocate type = 1
    CMSDT complete: rc = 0, errno = 11
    CMDEAL issued
    CMDEAL complete: rc = 0, errno = 11
  SGPADEA: exiting, rc = 0
  HGAFORG: exiting, rc = 0
  GPALDPT: entered
gpalfun = D
    OCIStmtExecute complete, rc = 0
  GPALDPT: exiting, rc= 0
  HGAFRT: exiting, rc = 0
  HGABEGN: entered, hgagt = efffe72c
  GPALBGN: entered
  lp->gpalgtil = 30
  GPALBGN: exiting
  HGABEGN: exiting, rc = <0>
  HGPADSC: entered
  HGPFND: entered
  HGPFND: exit - procedure 'PGATERM' was found
  HGPADSC: exiting, rc = <0>
  HGDAFR: entered
  HGDAFR: exit <0>
  HGAPEXEC: entered
  HGPFND: entered
  HGPFND: exit - procedure 'PGATERM' was found
  GPACTRM: entered
  GPACLOC: executed
  GPACLOC: exit, rc = 0
    GPAU found at 13e53c
    GPAC found at 198d20
The Dump in ASCII
PGATERM: Conversation id - @ efffe544
  0000: 00000001 01000000 00000000    *............*
And the same dump in EBCDIC
Sample Gateway Server Initialization and Trace Output Files

PGATERM: Conversation id - @ efffe544
0000: 00000001 01000000 00000000  *............*
PGATERM: Termination type = 0
GPACAE0: entered
GPACAE0: exiting, rc = 0
SGPADE0: entered, type = 0
SGP0RTS: entered, deal = 1
SGP0RTS: exiting, rc = 0, deal = 0
CMS0DT issued: deallocate type = 1
CMS0DT complete: rc = 0, errno = 11
CM0DEAL issued
CM0DEAL complete: rc = 0, errno = 11
SGPADE0: exiting, rc = 0
cp = 198d20
cpp = 0
cpn = 0
up->gpaucpt = 0

gpaucps cleared
freed gpac at 198d20
GPACTRM: exiting
HGAPEXE: exiting, rc = <0>
HGAPDSC: entered
HGAPPND: entered
HGAPPND: exit - procedure 'PGAINIT' was found
HGAPDSC: exiting, rc = <0>
HGADAFL: entered
HGADAFL: exit <0>
HGAPEXE: entered
HGAPPND: entered
HGAPPND: exit - procedure 'PGAINIT' was found
GPACIN0: entered
GPACL0C: executed
GPACL0C: exit, rc = 0
PGAINIT: TP name = 'DB2C'
PGAINIT: LU name = ''
PGAINIT: Mode name = 'ORAPLU62'
PGAINIT: Proflle name = 'CICSPGA'
PGAINIT: Sync level = 1
PGAINIT: Userid = PGAADMIN
PGAINIT: passwd = pgaadmin
GPACAE0: entered
GPACAE0: exiting, rc = 0
GPAC allocated at lc3d3c
SGP0ALL: entered
SGPA allocated at lc4758
Sample Gateway Server Initialization and Trace Output Files

CMINIT issued: profile = CICSPGA
CMINIT complete: rc = 0, errno = 0
CMEFLN issued
CMEFLN complete: rc = 0, errno = 0, luname = CICSPGA
CMSTPN issued: tpname = DB2C
CMSTPN complete: rc = 0, errno = 0
CMSMN issued: modename = ORAPLU62
CMSMN complete: rc = 0, errno = 0
XCSCST issued: security type = 0
XCSCST complete: rc = 0, errno = 0
CMSSL issued: synclevel = 1
CMSSL complete: rc = 0, errno = 0
CMSPTR issued: type = 1
CMSPTR complete: rc = 0, errno = 0
CMALLC issued
CMALLC complete: rc = 0, errno = 11
SGPAALL: exiting, rc = 0
gpaucps set to 1c3d3c
GPACAEC: entered
GPACAEC: exiting, rc = 0
The Dump in ASCII
PGAINIT: Conversation id - @ 19682c
0000: 00000002 01000002 00000000 *............*
And the same dump in EBCDIC
PGAINIT: Conversation id - @ 19682c
0000: 00000002 01000002 00000000 *............*
GPACINI: exiting rc = <0>
HGAPEXE: exiting, rc = <0>
HGAPDSC: entered
HGAPFND: entered
HGAPFND: exit - procedure 'PGAXFER' was found
HGAPDSC: exiting, rc = <0>
HGADAFO: entered
HGADAFO: exit <0>
HGAPEXE: entered
HGAPFND: entered
HGAPFND: exit - procedure 'PGAXFER' was found
GPACXFR: entered
GPACLOC: executed
GPACLOC: exit, rc = 0
GPAU found at 13e53c
GPAC found at 1c3d3c
sbufsz = 6
rbufsz = 32763
slenSz = 8
rlensz = 8

The Dump in ASCII
PGAXFER: Conversation id - @ efffe454
  0000: 00000002 01000002 00000000 *............*
And the same dump in EBCDIC
PGAXFER: Conversation id - @ efffe454
  0000: 00000002 01000002 00000000 *............*
PGAXFER: Send buffer length = 6
PGAXFER: Receive buffer length = 91

The Dump in ASCII
PGAXFER: Send lengths array - @ efff6350
  0000: 00000001 00000006 *........*
And the same dump in EBCDIC
PGAXFER: Send lengths array - @ efff6350
  0000: 00000001 00000006 *........*
PGAXFER: Receive lengths array - @ efff6248
  0000: 00000001 0000005B *.....*
And the same dump in EBCDIC
PGAXFER: Receive lengths array - @ efff6248
  0000: 00000001 0000005B *.....*$
PGAXFER: Send buffer - @ efff6458
  0000: F0F0F0F3 F4F0 *.....*
And the same dump in EBCDIC
PGAXFER: Send buffer - @ efff6458
  0000: F0F0F0F3 F4F0 *000340*

GPACAEC: entered

GPACAEC: exiting, rc = 0
  sends = 1
  receives = 1

SGPASND: entered, type = 0

SGPARTS: entered, deal = 0

SGPARTS: exiting, rc = 0, deal = 0

CMSST issued: send type = 0

CMSST complete: rc = 0, errno = 11

The Dump in ASCII

Sending data: - @ 13dd48
  0000: 50473441 50504320 00DEADBEEF00000000 *PG4APPC .........*
  0010: BEEF00DE ABEEF00 50473441 50504320 *..........PG4APPC *
  0020: 00523930 312E5445 4E415941 2E556131 *R901.TENAYA.5a1*
  0030: 39306662 652E3130 2E33302E 343530 *90f6e.10.30.450*
And the same dump in EBCDIC

Sending data: - @ 13dd48
  0000: 50473441 50504320 00DEADBEEF00000000 *%...%............*
Sample Gateway Server Initialization and Trace Output Files

0010:  BEEF00DE ADBEEF00 50473441 50504320 *........&...&&..*
0020:  00523930 312E544E 415941 2E356131 *........+...../.*
0030:  39306662 652E3130 2E33302E 343530 *................*
CMSEND issued: length = 63
CMSEND complete: rc = 0, errno = 11, rts = 0
SGPASND: exiting, rc = 0
   slens[1] = 6
SGPASND: entered, type = 2
SGPARTS: entered, deal = 0
SGPARTS: exiting, rc = 0, deal = 0
   CMSST issued: send type = 3
   CMSST complete: rc = 0, errno = 11
The Dump in ASCII
Sending data: - @ efff6458
   0000:  F0F0F0F3 F4F0                        *......*
And the same dump in EBCDIC
Sending data: - @ efff6458
   0000:  F0F0F0F3 F4F0                        *000340*
CMSEND issued: length = 6
CMSEND complete: rc = 0, errno = 11, rts = 0
SGPASND: exiting, rc = 0
   rlens[1] = 91
SGPARCV: entered
CMECS issued
   CMECS complete: rc = 0, errno = 11, state = 4
CMRCV issued: length = -268476352
CMRCV complete: rc = 0, errno = 11, rts = 0, dr = 2, sr = 1, length = 91
The Dump in ASCII
Received data: - @ 175858
   0000:  F0F0F0F3 F4F00005 D1C1E2D6 D5000000 *.................*
   0010:  800C44F2 D6E4D5D6 E3000000 *.D.............*
   0020:  9BF005DB 0000E3E2 F8F1F9F4 F8F1F9F4 *................*
   0030:  F760F0F5 60F0F5C6 C9C5D3C4 D9C5D700 *................*
   0040:  10D4F1F9 F2F660F0 F560F1F7 00238400 *.............#
   0050:  0C000050 000C0001 90700C *.....P...p.*
And the same dump in EBCDIC
Received data: - @ 175858
   0000:  F0F0F0F3 F4F00005 D1C1E2D6 D5000000 *000340..JASON...*
   0010:  800C44F2 D6E4D5D6 E3000000 *.2R..GOUNOT...*
   0020:  9BF005DB 0000E3E2 F8F1F9F4 F8F1F9F4 *)0.....TST569194*
   0030:  F760F0F5 60F0F5C6 C9C5D3C4 D9C5D700 *7-05-05FIELDREP.*
   0040:  10D4F1F9 F2F660F0 F560F1F7 00238400 *.M1926-05-17..d.*
   0050:  0C000050 000C0001 90700C *.....&...*
   conversation state is now SEND
SGPARCV: exiting, rc = 0

6-30  Oracle Procedural Gateway for APPC User's Guide
PGAXFER: Receive buffer length = 91
The Dump in ASCII
PGAXFER: Receive lengths array - @ eeff6248
   0000: 00000001 0000005B *........[*
And the same dump in EBCDIC
PGAXFER: Receive lengths array - @ eeff6248
   0000: 00000001 0000005B *........$*
The Dump in ASCII
PGAXFER: Receive buffer - @ 175858
   0000: F0F0F0F3 F4F00005 D1C1E2D6 D5000000 *000340...JASON...*
   0010: 80C44F2 D90006C7 D6E4D5D6 E3000000 *...2R...GOUNOT...*
   0020: 9BF000DB 000032E2 E3F5F6F9 F8F1F9F4 *010...IST5698194*
   0030: F760F0F5 60F0F5C6 C9C5D3C4 D9C5D700 *7-05-05FIELDREP.*
   0040: 10D4F1F9 F2F660F0 F560F1F7 00238400 *........7...#*
   0050: 0C000050 0000001 90700C *P..p.*
And the same dump in EBCDIC
PGAXFER: Receive buffer - @ 175858
   0000: F0F0F0F3 F4F00005 D1C1E2D6 D5000000 *000340...JASON...*
   0010: 80C44F2 D90006C7 D6E4D5D6 E3000000 *...2R...GOUNOT...*
   0020: 9BF000DB 000032E2 E3F5F6F9 F8F1F9F4 *010...IST5698194*
   0030: F760F0F5 60F0F5C6 C9C5D3C4 D9C5D700 *7-05-05FIELDREP.*
   0040: 10D4F1F9 F2F660F0 F560F1F7 00238400 *M1926-05-17..d.*
   0050: 0C000050 0000001 90700C *.......*
GPACAEC: entered
GPACAEC: exiting, rc = 0
GPACXFR: exiting rc = <0>
HGAPXCE: exiting, rc = <0>
HGAPXCE: entered
HGAPFND: entered
HGAPFND: exit - procedure 'PGAXFER' was found
GPACXFR: entered
GPACLOC: executed
GPACLOC: exit, rc = 0
GPAC found at 13e53c
GPAC found at 1c3d3c
sbufsz = 91
rbufsz = 91
slensz = 8
rlensz = 8
The Dump in ASCII
PGAXFER: Conversation id - @ efffe454
   0000: 00000002 01000002 00000000 *...........
And the same dump in EBCDIC
PGAXFER: Conversation id - @ efffe454
   0000: 00000002 01000002 00000000 *...........
Sample Gateway Server Initialization and Trace Output Files

PGAXFER: Send buffer length = 91
PGAXFER: Receive buffer length = 4

The Dump in ASCII
PGAXFER: Send lengths array - @ efff6350
0000: 00000001 0000005B *. ....... [*
And the same dump in EBCDIC
PGAXFER: Send lengths array - @ efff6350
0000: 00000001 0000005B *. ....... $*

The Dump in ASCII
PGAXFER: Receive lengths array - @ efff6248
0000: 00000001 00000004 *. ....... *
And the same dump in EBCDIC
PGAXFER: Receive lengths array - @ efff6248
0000: 00000001 00000004 *. ....... *

The Dump in ASCII
PGAXFER: Send buffer - @ efff6458
0000: F0F0F0F3 F4F00005 D1C1E2D6 D5000000 *. ................ *
0010: 800C44F2 D900006C7 D6E4D5D6 E3000000 *. ..D ............ *
0020: 9BF005DB 000C5F2 F1F5F6F9 F8F1F9F4 *. ................ *
0030: F760F0F5 D60F05C6 C9C5D3C4 D9C5D700 *. '. ` ............ *
0040: 1004F1F9 F2F660F0 F560F1F7 00238400 *. ....` ...# ... *
0050: 0C000050 000C0001 90700C *. ....F ......p *

And the same dump in EBCDIC
PGAXFER: Send buffer - @ efff6458
0000: F0F0F0F3 F4F00005 D1C1E2D6 D5000000 *00340..JASON...*
0010: 800C44F2 D900006C7 D6E4D5D6 E3000000 *. ...2R..GOUNOT...*
0020: 9BF005DB 000C5F2 F1F5F6F9 F8F1F9F4 *. ............. *
0030: F760F0F5 D60F05C6 C9C5D3C4 D9C5D700 *. 7-05-05FIELDREP...*
0040: 1004F1F9 F2F660F0 F560F1F7 00238400 *. M1926-05-17. ..d. *
0050: 0C000050 000C0001 90700C *. ....& .....

GPACAE: entered
GPACAE: exiting, rc = 0
    sends = 1
    receives = 1
    slens[1] = 91
SGPASND: entered, type = 2
SGPARTS: entered, deal = 0
SGPARTS: exiting, rc = 0, deal = 0
CMSST issued: send type = 3
CMSST complete: rc = 0, errno = 11

The Dump in ASCII
Sending data: - @ efff6458
0000: F0F0F0F3 F4F00005 D1C1E2D6 D5000000 *. ............. *
0010: 800C44F2 D900006C7 D6E4D5D6 E3000000 *. ...D ............ *
0020: 9BF005DB 000C5F2 F1F5F6F9 F8F1F9F4 *. ............. *
Sample Gateway Server Initialization and Trace Output Files

And the same dump in EBCDIC

Sending data: - @ efff6458
0000: F0F0F0F3 F4F00005 D1C1E2D6 D5000000 *00340..JASON...*
0010: 80C44F2 D90006C7 6E4D56D6 E3000000 *...2R..GOUNOT...*
0020: 9BF0005B 000C5F2 F1F5F6F9 F8F1F9F4 *)...E215698194*
0030: 760F0F0F 560F0F5C6 C9C5D3C4 D9C5D700 *7-05-05FIELDREP.*
0040: 10D4F1F9 F2F660F0 F560F1F7 00238400 *.M1926-05-17..d.*
0050: 0C000050 000C0001 90700C *....*******
CMSEND issued: length = 91
CMSEND complete: rc = 0, errno = 11, rts = 0
SGPASND: exiting, rc = 0
r lens[1] = 4
SGPARCV: entered
CMMECS issued
CMMECS complete: rc = 0, errno = 11, state = 4
CMRCVC issued: length = -268476352
CMRCVC complete: rc = 0, errno = 11, dr = 2, sr = 1, length = 4
The Dump in ASCII
Received data: - @ 166ce8
0000: 00000000 *....*
And the same dump in EBCDIC
Received data: - @ 166ce8
0000: 00000000 *....*
conversation state is now SEND
SGPARCV: exiting, rc = 0
PGAXFER: Receive buffer length = 4
The Dump in ASCII
PGAXFER: Receive lengths array - @ efff6248
0000: 00000001 00000004 *.........*
And the same dump in EBCDIC
PGAXFER: Receive lengths array - @ efff6248
0000: 00000001 00000004 *.........*
The Dump in ASCII
PGAXFER: Receive buffer - @ 166ce8
0000: 00000000 *....*
And the same dump in EBCDIC
PGAXFER: Receive buffer - @ 166ce8
0000: 00000000 *....*
GPACAEC: entered
GPACAEC: exiting, rc = 0
GPACXFR: exiting rc = <0>
HGAPEXE: exiting, rc = <0>
HGAROLL: entered
SGPACTL: entered, type = 1
   CMSERR issued
      CMSERR complete: rc = 0, errno = 11, rts = 0
SGPACTL: exiting, rc = 0
HGAROLL: exiting, rc = 0
HGABEGN: entered, hagt = efffe72c
GPALBGN: entered
lp->gpalgt1l = 29
GPALBGN: exiting
HGABEGN: exiting, rc = <0>
HGAPDSC: entered
HGAPFND: entered
HGAPFND: exit - procedure 'PGATERM' was found
HGAPDSC: exiting, rc = <0>
HGADAFR: entered
HGADAFR: exit <0>
HGAPEXE: entered
HGAPFND: entered
HGAPFND: exit - procedure 'PGATERM' was found
GPACTRM: entered
GPACLOC: executed
GPACLOC: exit, rc = 0
   GPAU found at 13e53c
   GPAC found at 1c3d3c
The Dump in ASCII
PGATERM: Conversation id - @ efffe544
   0000:  00000002 01000002 00000000           *............*
   *............* And the same dump in EBCDIC
PGATERM: Conversation id - @ efffe544
   0000:  00000002 01000002 00000000           *............*
   *............* PGATERM: Termination type = 0
GPACAEC: entered
GPACAEC: exiting, rc = 0
SGPDEA: entered, type = 0
SGPARTS: entered, deal = 1
SGPARTS: exiting, rc = 0, deal = 0
   CMSDT issued: deallocate type = 1
   CMSDT complete: rc = 0, errno = 11
   CMDEAL issued
   CMDEAL complete: rc = 0, errno = 11
SGPDEA: exiting, rc = 0
   cp = 1c3d3c
cpp = 0
cpn = 0
up->gpaucpt = 0
gpaucps cleared
freed gpac at 1c3d3c
GPACTRM: exiting
HGAPEXE: exiting, rc = <0>
HGAPDSC: entered
HGAPFND: entered
HGAPFND: exit - procedure 'PGAINIT' was found
HGAPDSC: exiting, rc = <0>
HGADAFR: entered
HGADAFR: exit <0>
HGAPEXE: entered
HGAPFND: entered
HGAPFND: exit - procedure 'PGAINIT' was found
GPACINI: entered
GPACLOC: executed
GPACLOC: exit, rc = 0
PGAINIT: TP name      = 'DB2C'
PGAINIT: LU name      = ''
PGAINIT: Mode name    = 'ORAPLU62'
PGAINIT: Profile name = 'CICSPFGA'
PGAINIT: Sync level   = 1
PGAINIT: Userid       = PGAADMIN
PGAINIT: passwd       = pgaadmin
GPACAEC: entered
GPACAEC: exiting, rc = 0
GPAC allocated at la0db8
SGPAALL: entered
SGPA allocated at lcc7b0
CMINIT issued: profile = CICSPFGA
CMINIT complete: rc = 0, errno = 0
CMEPLN issued
CMEPLN complete: rc = 0, errno = 0, luname = CICSPFGA
CMSTPN issued: tpname = DB2C
CMSTPN complete: rc = 0, errno = 0
CMSMN issued: modename = ORAPLU62
CMSMN complete: rc = 0, errno = 0
XCSCST issued: security type = 0
XCSCST complete: rc = 0, errno = 0
CMSSL issued: synclevel = 1
CMSSL complete: rc = 0, errno = 0
CMSPTR issued: type = 1
CMSPTR complete: rc = 0, errno = 0
CMALLC issued
CMALLC complete: rc = 0, errno = 11
SGPAALL: exiting, rc = 0
gpaucps set to 1a0db8
GPACAEC: entered
GPACAEC: exiting, rc = 0
The Dump in ASCII
PGAINIT: Conversation id - @ 1a0ea8
  0000:  00000003 01000003 00000000           *............*
And the same dump in EBCDIC
GAINIT: Conversation id - @ 1a0ea8
  0000:  00000003 01000003 00000000           *............*
GPACINI: exiting rc = <0>
HGAPEXE: exiting, rc = <0>
HGAPDSC: entered
HGAPFND: entered
HGAPFND: exit - procedure 'PGAXFER' was found
HGAPDSC: exiting, rc = <0>
HGADAFR: entered
HGADAFR: exit <0>
HGAPEXE: entered
HGAPFND: entered
HGAPFND: exit - procedure 'PGAXFER' was found
GPACXFR: entered
GPACLLOC: executed
GPACLLOC: exit, rc = 0
GPAC found at 13e53c
GPAC found at 1a0db8
sbufsz = 6
rbufsz = 32763
slenz = 8
rlensz = 8
The Dump in ASCII
PGAXFER: Conversation id - @ efffe454
  0000:  00000003 01000003 00000000           *............*
And the same dump in EBCDIC
PGAXFER: Conversation id - @ efffe454
  0000:  00000003 01000003 00000000           *............*
PGAXFER: Send buffer length = 6
PGAXFER: Receive buffer length = 91
The Dump in ASCII
PGAXFER: Send lengths array - @ efff6350
  0000:  00000001 00000006           *........*
And the same dump in EBCDIC
PGAXFER: Send lengths array - @ efff6350
  0000:  00000001 00000006           *........*
The Dump in ASCII
PGAXFER: Receive lengths array - @ efff6248
   0000:  00000001 000005B                    *.......[*
And the same dump in EBCDIC
PGAXFER: Receive lengths array - @ efff6248
   0000:  00000001 000005B                    *.......$*
The Dump in ASCII
PGAXFER: Send buffer - @ efff6458
   0000:  F0F0F0F3 F4F0                        *......*
And the same dump in EBCDIC
PGAXFER: Send buffer - @ efff6458
   0000:  F0F0F0F3 F4F0 *000340*
GPACAEC: entered
GPACAEC: exiting, rc = 0
   sends = 1
   receives = 1
SGPASND: entered, type = 0
SGPARTS: entered, deal = 0
SGPARTS: exiting, rc = 0, deal = 0
   CMSST issued: send type = 0
   CMSST complete: rc = 0, errno = 11
The Dump in ASCII
Sending data: - @ 13dd48
   0000:  50473441 50504320 00DEADBE EF00DEAD  *PG4APPC ........*
   0010:  BEEF00DE ADBEF00 50473441 50504320  *........PG4APPC *
   0020:  00523930 312E54E15941 2E356131  *.R901.TENAYA.5a1*
   0030:  39306662 652E3132E 31342E34 3239  *90fbe.1.14.429*
And the same dump in EBCDIC
Sending data: - @ 13dd48
   0000:  50473441 50504320 00DEADBE EF00DEAD  *PG4APPC ........*
   0010:  BEEF00DE ADBEF00 50473441 50504320  * symbolism.
   0020:  00523930 312E54E15941 2E356131  *.R901.TENAYA.5a1*
   0030:  39306662 652E3132E 31342E34 3239  *90fbe.1.14.429*
CMSSEND issued: length = 62
   CMSSEND complete: rc = 0, errno = 11, rts = 0
SGPASND: exiting, rc = 0
   slens[1] = 6
SGPASND: entered, type = 2
SGPARTS: entered, deal = 0
SGPARTS: exiting, rc = 0, deal = 0
   CMSST issued: send type = 3
   CMSST complete: rc = 0, errno = 11
The Dump in ASCII
Sending data: - @ efff6658
   0000:  F0F0F0F3 F4F0                        *......*
And the same dump in EBCDIC
Sending data: - @ efff6458
0000: F0F0F0F3 F4F00005 D1C1E2D6 D5000000 *.................*  
0010: 800C44F2 D90006C7 D6E4D5D6 E3000000 *..D.............*  
0020: 9BF005DB 0000E3E2 E3F5F6F9 F8F1F9F4 *..................*  
0030: F760F0F5 60F0F5C6 C9C5D3C4 D9C5D700 *..................*  
0040: 10D4F1F9 F2F666F0 F560F1F7 00238400 *.......'...#..*  
0050: 0C000050 000C0001 90700C *...P.....p.*  

And the same dump in EBCDIC

Receiving data: - @ 175858
0000: F0F0F0F3 F4F00005 D1C1E2D6 D5000000 *000340..JASON....*  
0010: 800C44F2 D90006C7 D6E4D5D6 E3000000 *...2R.GOUNOT....*  
0020: 9BF005DB 0000E3E2 E3F5F6F9 F8F1F9F4 *00..00..TST5698194*  
0030: F760F0F5 60F0F5C6 C9C5D3C4 D9C5D700 *7-05-05FIELDREP.*  
0040: 10D4F1F9 F2F666F0 F560F1F7 00238400 *..M1926-05-17..d.*  
0050: 0C000050 000C0001 90700C *...&.......*  

conversation state is now SEND

The Dump in ASCII

PGAXFER: Receive buffer length = 91

And the same dump in EBCDIC

The Dump in ASCII
Sample Gateway Server Initialization and Trace Output Files

PGAXFER: Receive buffer - @ 175858
0000:  F0F0F0F3 F4F00005 D1C1E2D6 D5000000 *000340..JASON...*
0010:  80C44F2 D90006C7 D6E4D5D6 E3000000 *...2R..GOUNOT...*
0020:  9BF005DB 000E3E2 E3F56F9F F8F1F9F4 *)0....TST5698194*
0030:  F760F0F5 60F0F5C6 C9C5D3C4 D9C5D700 *7-05-05FIELDREP.*
0040:  10D4F1F9 F2F66F0F F560F1F7 00238400 *.M1926-05-17..d.*
0050:  0C000050 000C0001 90700C *...&.......
GPACAECE: entered
GPACAECE: exiting, rc = 0
GPACXFR: exiting rc = <0>
HGAPXEXE: exiting, rc = <0>
HGAPXEXE: entered
HGAPFND: entered
HGAPFND: exit - procedure 'PGAXFER' was found
GPACXFR: entered
GPACLOC: executed
GPACLOC: exit, rc = 0
GPAU found at 13e53c
GPAC found at la8db8
sbufsz = 91
rbufsz = 91
slensz = 8
rlensz = 8
The Dump in ASCII
PGAXFER: Conversation id - @ efffe454
0000:  00000003 01000303 00000000 *............*
And the same dump in EBCDIC
PGAXFER: Conversation id - @ efffe454
0000:  00000003 01000303 00000000 *............*
PGAXFER: Send buffer length = 91
PGAXFER: Receive buffer length = 4
The Dump in ASCII
PGAXFER: Send lengths array - @ efff6350
0000:  00000001 0000005B *........[*
And the same dump in EBCDIC
PGAXFER: Send lengths array - @ efff6350
0000:  00000001 0000005B *........$*
The Dump in ASCII
PGAXFER: Receive lengths array - @ efff6248
0000:  00000001 00000004 *........*
And the same dump in EBCDIC
PGAXFER: Receive lengths array - @ efff6248
0000:  00000001 00000004 *........*
The Dump in ASCII
PGAXFER: Send buffer - @ efff6458
Sample Gateway Server Initialization and Trace Output Files

0000: F0F0F0F3 F4F00005 D1C1E2D6 D5000000 *................*
0010: 800C44F2 D90006C7 D6E4D5D6 E3000000 *..D.............*
0020: 9BF005DB 000C5F2 F1F5F6F9 F8F1F9F4 *................*
0030: F760F0F5 60F0F5C6 C9C5D3C4 D9C5D700 * `
0040: 10D4F1F9 F2F660F0 F560F1F7 00238400 *...........%.*
0050: 0C000050 000C0001 90700C *...P.......p.*

And the same dump in EBCDIC

PGAXFER: Send buffer - @ efff6458

0000: F0F0F0F3 F4F00005 D1C1E2D6 D5000000 *000340..JASON...*
0010: 800C44F2 D90006C7 D6E4D5D6 E3000000 *...2R..GOUNOT...*
0020: 9BF005DB 000C5F2 F1F5F6F9 F8F1F9F4 *)0...E215698194*
0030: F760F0F5 60F0F5C6 C9C5D3C4 D9C5D700 *7-05-05FIELDREP.*
0040: 10D4F1F9 F2F660F0 F560F1F7 00238400 *.M1926-05-17..d.*
0050: 0C000050 000C0001 90700C *...&.........*

GPACAE: entered
GPACAE: exiting, rc = 0

sends = 1
receives = 1
slens[1] = 91

SGPASND: entered, type = 2
SGPARTS: entered, deal = 0
SGPARTS: exiting, rc = 0, deal = 0

CMSST issued: send type = 3
CMSST complete: rc = 0, errno = 11

The Dump in ASCII

Sending data: @ efff6458

0000: F0F0F0F3 F4F00005 D1C1E2D6 D5000000 *................*
0010: 800C44F2 D90006C7 D6E4D5D6 E3000000 *..D.............*
0020: 9BF005DB 000C5F2 F1F5F6F9 F8F1F9F4 *................*
0030: F760F0F5 60F0F5C6 C9C5D3C4 D9C5D700 * `
0040: 10D4F1F9 F2F660F0 F560F1F7 00238400 *...........%.*
0050: 0C000050 000C0001 90700C *...P.......p.*

And the same dump in EBCDIC

Sending data: @ efff6458

0000: F0F0F0F3 F4F00005 D1C1E2D6 D5000000 *000340..JASON...*
0010: 800C44F2 D90006C7 D6E4D5D6 E3000000 *...2R..GOUNOT...*
0020: 9BF005DB 000C5F2 F1F5F6F9 F8F1F9F4 *)0...E215698194*
0030: F760F0F5 60F0F5C6 C9C5D3C4 D9C5D700 *7-05-05FIELDREP.*
0040: 10D4F1F9 F2F660F0 F560F1F7 00238400 *.M1926-05-17..d.*
0050: 0C000050 000C0001 90700C *...&.........*

CMSSEND issued: length = 91
CMSSEND complete: rc = 0, errno = 11, rts = 0

SGPASND: exiting, rc = 0

rlens[1] = 4
SGPARECV: entered
CMECS issued
CMECS complete: rc = 0, errno = 11, state = 4
CMRCV issued: length = -268476352
CMRCV complete: rc = 0, errno = 11, rts = 0, dr = 2, sr = 1, length = 4

The Dump in ASCII
Received data: - @ 166ca8
0000: 00000000 *....*
And the same dump in EBCDIC
Received data: - @ 166ca8
0000: 00000000 *....*

conversation state is now SEND
SGPARCV: exiting, rc = 0
PGAXFER: Receive buffer length = 4

The Dump in ASCII
PGAXFER: Receive lengths array - @ efff6248
0000: 00000001 00000004 *........*
And the same dump in EBCDIC
PGAXFER: Receive lengths array - @ efff6248
0000: 00000001 00000004 *........*

The Dump in ASCII
PGAXFER: Receive buffer - @ 166ca8
0000: 00000000 *....*
And the same dump in EBCDIC
PGAXFER: Receive buffer - @ 166ca8
0000: 00000000 *....*

GPACAEC: entered
GPACAEC: exiting, rc = 0
GPACXFR: exiting rc = <0>
HGAPEXE: exiting, rc = <0>
HGACOMM: entered, keepinfo = 1
GPALIPT: entered
gpalfun = I
gpalgti = R901.TENAYA.5a190fbe.1.14.429
len = 29
gpalprf = CICSPGA
len = 8
gpalun = CICSPGA
len = 7
gpalmdn = ORAPLU62
len = 8
gpaltpn = DB2C
len = 4
OCIStmtExecute complete, rc = 0
GPALIPT: exiting, rc= 0
SGPACTL: entered, type = 0

Problem Determination 6-41
Sample Gateway Server Initialization and Trace Output Files

SGPARTS: entered, deal = 0
SGPARTS: exiting, rc = 0, deal = 0
   CMC FM issued
   CMC FM complete: rc = 0, errno = 11, rts = 0
SGPACTL: exiting, rc = 0
HGACOMM: exiting, rc = 0
HGAFRGT: entered
HGAFORG: entered
SGPAALL: entered
   SGPA allocated at 1d4808
   CMINIT issued: profile = CICSPGA
   CMINIT complete: rc = 0, errno = 11
   CMSPLN issued: luname = CICSPGA
   CMSPLN complete: rc = 0, errno = 11
   CMSTPN issued: tpname = RECO
   CMSTPN complete: rc = 0, errno = 11
   CMSMN issued: modename = ORAPLU62
   CMSMN complete: rc = 0, errno = 11
   XCSCST issued: security type = 0
   XCSCST complete: rc = 0, errno = 11
   CMSSL issued: synclevel = 1
   CMSSL complete: rc = 0, errno = 11
   CMSPTR issued: type = 1
   CMSPTR complete: rc = 0, errno = 11
   CMALLC issued
   CMALLC complete: rc = 0, errno = 11
SGPAALL: exiting, rc = 0
SGPASND: entered, type = 3
SGPARTS: entered, deal = 0
SGPARTS: exiting, rc = 0, deal = 0
   CMSST issued: send type = 2
   CMSST complete: rc = 0, errno = 11
The Dump in ASCII
Sending data: - @ 13dd68
  0000:  01523930 312E5445 4E415941 2E356131  *.R901.TENAYA.5a1*
  0010:  39306662 652E312E 31342E34 3239      *90fbe.1.14.429*
CMSEND issued: length = 30
CMSEND complete: rc = 0, errno = 11, rts = 0
And the same dump in EBCDIC
Sending data: - @ 13dd68
  0000:  01523930 312E5445 4E415941 2E356131  *.R901.TENAYA.5a1*
  0010:  39306662 652E312E 31342E34 3239      *90fbe.1.14.429*
   CMSST issued: send type = 2
   CMSST complete: rc = 0, errno = 11
SGPASND: exiting, rc = 0
SGPADEA: entered, type = 0
SGPARTS: entered, deal = 1
SGPARTS: exiting, rc = 0, deal = 0
CMSDT issued: deallocate type = 1
CMSDT complete: rc = 0, errno = 11
CMDEAL issued
CMDEAL complete: rc = 0, errno = 11
SGPADEA: exiting, rc = 0
HGAFORG: exiting, rc = 0
GPALDPT: entered
gpalfun = D
OCISstmtExecute complete, rc = 0
GPALDPT: exiting, rc = 0
HGAFRG: exiting, rc = 0
HGABEGN: entered, hgagt = efffe72c
GPALBGN: entered
lp->gpalgtil = 29
GPALBGN: exiting
HGABEGN: exiting, rc = <0>
HGAPDSC: entered
HGAPFND: entered
HGAPFND: exit - procedure 'PGATERM' was found
HGAPDSC: exiting, rc = <0>
HGADAFR: entered
HGADAFR: exit <0>
HGAPEXE: entered
HGAPFND: entered
HGAPFND: exit - procedure 'PGATERM' was found
GPACTRM: entered
GPACLOC: executed
GPACLOC: exit, rc = 0
GPAU found at 13e53c
GPAC found at 1a0db8
The Dump in ASCII
PGATERM: Conversation id - 0 efffe544
0000: 00000003 01000003 00000000 *............*
PGATERM: Termination type = 0
GPACAEC: entered
GPACAEC: exiting, rc = 0
SGPADEA: entered, type = 0
SGPARTS: entered, deal = 1
SGPARTS: exiting, rc = 0, deal = 0
CMSDT issued: deallocate type = 1
CMSDT complete: rc = 0, errno = 11
CMDEAL issued
CMDEAL complete: rc = 0, errno = 11
SGPADEA: exiting, rc = 0
cp = 1a0db8
cpp = 0
cpn = 0
up->gpaucpt = 0
gpaucps cleared
freed gpac at 1a0db8
GPACTRM: exiting
HGAPEXE: exiting, rc = <0>
HGACOMM: entered, keepinfo = 0
HGACOMM: exiting, rc = 0
HGADAFR: entered
HGADAFR: exit <0>
HGADAFR: entered
HGADAFR: exit <0>
HGADAFR: entered
HGADAFR: exit <0>
HGADAFR: entered
HGADAFR: exit <0>
HGALGOF: entered
GPALLOFF: entered
logoff, rc = 0
OCIHandleFree -- STMT, rc = 0
OCIHandleFree -- Error, rc = 0
GPALLOF: exiting, rc= 0
HGALGOF: exiting, rc = 0
HGAEXIT: entered
HGAEXIT: exiting, rc = 0
This appendix contains the following sections:

- **PG DD Environment Dictionary** on page A-2
- **PG DD Active Dictionary** on page A-4
The Procedural Gateway Data Dictionary (PG DD) is maintained in a conventional Oracle database. It is installed by a SQL*Plus installation script (`pgddcr8.sql` in the `SORACLE_HOME/pg4appc/admin` directory on the gateway system) and manipulated by PGAU statements and standard SQL statements.

The dictionary is divided into two sections:
- the environment dictionary
- the active dictionary

The environment dictionary is static and should not be changed. The contents of the environment dictionary support proper translation from the remote transaction’s environment to the integrating Oracle integrating server’s environment, and is platform-specific. The active dictionary is updated at the user’s location by the PGAU in response to definitions supplied by the user.

**PG DD Environment Dictionary**

The PGAU uses some dictionary tables strictly as input. These dictionary tables define environmental parameters for PGAU. Both table and values are installed by a SQL*Plus script at gateway installation time and are not to be modified by the installation.

The environment dictionary does not reference the active dictionary, but the active dictionary does reference environment dictionary entries.

**Environment Dictionary Sequence Numbers**

The environment dictionary requires unique identifying numbers in some columns to join environment dictionary entries together. Oracle sequence objects are therefore created by the Oracle Procedural Gateway for APPC to support this requirement. The following table presents the Oracle sequence objects and their descriptions.

<table>
<thead>
<tr>
<th>Oracle sequence objects</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pga.envrseq</td>
<td>Environment id tag</td>
</tr>
<tr>
<td>pga.compseq</td>
<td>Compiler id tag</td>
</tr>
<tr>
<td>pga.eattrseq</td>
<td>Environment Attribute id tag</td>
</tr>
<tr>
<td>pga.dtypeseq</td>
<td>Datatype id tag</td>
</tr>
</tbody>
</table>
Environment Dictionary Tables
The environment dictionary tables contain constants that describe these components of the operating environment:

- pga_maint
- pga_usage
- pga_modes

pga_maint
The pga_maint table stores the PG DD version number and change history, as presented in the following table.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>number(10,4)</td>
<td>PG DD version in format VVRRFF.rrff, where: VV - base version; RR - base release; FF - base fix; rr - port-specific release; ff - port-specific fix.</td>
</tr>
<tr>
<td>mntdate</td>
<td>date</td>
<td>Oracle date and time at which the PG DD was upgraded.</td>
</tr>
<tr>
<td>change</td>
<td>varchar2(256)</td>
<td>Description of the PG DD upgrade.</td>
</tr>
</tbody>
</table>
pga_usage
The pga_usage table performs a referential integrity check of pga_data and pga_field column "usage" as presented in the following table.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>varchar2(6)</td>
<td>Value for the &quot;usage&quot; field of data dictionary tables. For example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘PASS’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘SKIP’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘NULL’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘ASIS’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Primary key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max length =&gt; 4-char string length</td>
</tr>
</tbody>
</table>

pga_modes
The pga_modes table performs a referential integrity check of pga_call_parm column "mode", as presented in the following table.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>varchar2(6)</td>
<td>Name of valid parameter call modes. For example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘IN’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘OUT’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘IN OUT’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max length =&gt; ‘IN OUT’ string length</td>
</tr>
</tbody>
</table>

PG DD Active Dictionary
The PG DD active data dictionary is created by pgddcr8.sql at installation, but maintained using PGAU. The active dictionary can refer to items (by ID number) in the environment dictionary.

Active Dictionary Versioning
The PG DD active dictionary tables contain the descriptions of transactions and data structures. There might be more than one version of a definition. Old versions are retained indefinitely.
In all PGAU dictionary operations, a definition is referred to by its "name". That name can be qualified by specific version number. If omitted, the most recent version is assumed.

Active Dictionary Sequence Numbers

Because the active dictionary is constantly changing, the identifying numbers needed to join active dictionary entries together must also change. To support this requirement, PG DD installation creates the following Oracle sequence objects. The table lists the Oracle sequence objects and their descriptions:

<table>
<thead>
<tr>
<th>Oracle sequence objects</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pga.transeq</td>
<td>Transaction id tag</td>
</tr>
<tr>
<td>pga.tranvers</td>
<td>Transaction Version id tag</td>
</tr>
<tr>
<td>pga.tattrseq</td>
<td>Transaction Attribute id tag</td>
</tr>
<tr>
<td>pga.callseq</td>
<td>APPC-Call id tag</td>
</tr>
<tr>
<td>pga.callvers</td>
<td>Call Version id tag</td>
</tr>
<tr>
<td>pga.parmseq</td>
<td>APPC-Call Parameter id tag</td>
</tr>
<tr>
<td>pga.dataseq</td>
<td>Data id tag</td>
</tr>
<tr>
<td>pga.fieldseq</td>
<td>Data subfield id tag</td>
</tr>
<tr>
<td>pga.datavers</td>
<td>Data Version id tag</td>
</tr>
<tr>
<td>pga.dattrseq</td>
<td>Data Attribute id tag</td>
</tr>
</tbody>
</table>

Active Dictionary Tables

Following is a list of active dictionary tables:

- pga_trans
- pga_trans_attr
- pga_trans_values
- pga_trans_calls
- pga_call
- pga_data
- pga_fields
### pga_trans

One row exists in the pga_trans table for each user transaction. The row is created by a PGAU DEFINE TRANSACTION statement and used by a PGAU GENERATE statement to create the PL/SQL package (TIP).

The following table presents the column, type and content information for pga_trans:

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>tname</td>
<td>varchar2(64)</td>
<td>Transaction name as defined by the customer. Primary key. Max length =&gt; APPC TPname string length.</td>
</tr>
<tr>
<td>version</td>
<td>number(9,0)</td>
<td>Version identification of this entry; it exists in the table because multiple archived or invalid entries might exist and be kept for possible future reactivation. Primary key. Set from an Oracle sequence object for transaction version inserted into the PG DD.</td>
</tr>
<tr>
<td>updtdate</td>
<td>date</td>
<td>Audit-trail date/time record last updated.</td>
</tr>
<tr>
<td>updtuser</td>
<td>varchar2(30)</td>
<td>Audit-trail user ID/program which last updated this record.</td>
</tr>
<tr>
<td>trans#</td>
<td>number(9,0)</td>
<td>PGA Transaction number, used for the define call, define data and define transaction statements. Foreign key. pga_trans_values(trans#), pga_trans_calls(trans#). Set from an Oracle sequence object for transaction inserted into the PG DD.</td>
</tr>
</tbody>
</table>

### pga_trans_attr

The pga_trans_attr table relates a character string defining the transaction attributes supported by PGA to pga_trans_values entries through an attribute id number and type.
The pga_trans_attr table is also used for integrity checks of transaction attributes when new transactions are being defined.

There is an entry in the pga_trans_attr table for each transaction attribute name. All possible transaction attribute names supported by PGA on any defined transaction are specified. There is one row for each attribute, and no duplicates are allowed.

The following table presents the column, type and content information for pga_trans_attr:

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>varchar2(16)</td>
<td>Character string name of attribute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Primary key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Contains:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;ENVIRONMENT&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;LUNAME&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;TPNAME&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;LOGMODE&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;SIDEPROFILE&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;SYNCELEVEL&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;NLS_LANGUAGE&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;REMOTE_MBCS&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;LOCAL_MBCS&quot;</td>
</tr>
<tr>
<td>attr#</td>
<td>number(9,0)</td>
<td>Attribute id assigned.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Foreign key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pga_data_values(attr#).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Set from an Oracle sequence object for each supported transaction attribute inserted into the PG DD.</td>
</tr>
<tr>
<td>coltype</td>
<td>varchar2(4)</td>
<td>Type of Oracle column from which attribute value is retrieved from pga_tran_values. For example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'NUM' =&gt; pga_tran_values(numval)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'CHAR' =&gt; pga_tran_values(charval)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'DATE' =&gt; pga_tran_values(dateval)</td>
</tr>
<tr>
<td>required</td>
<td>char(1)</td>
<td>If not null, required keyword for DEFINE TRANSACTION; if null, optional.</td>
</tr>
</tbody>
</table>
**pga_trans_values**

The `pga_trans_values` table describes the values of transaction attributes. A row exists to specify the value of each attribute of each transaction defined in the data dictionary. The column, type and content information for `pga_trans_values` is presented in the following table:

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>trans#</td>
<td>number(9,0)</td>
<td>Transaction id from <code>pga_trans(trans#)</code>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Primary key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Set from an Oracle sequence object for transaction inserted into the PG DD.</td>
</tr>
<tr>
<td>attr#</td>
<td>number(9,0)</td>
<td>Attribute id from <code>pga_trans_attr(attr#)</code>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Primary key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Set from an Oracle sequence object for each supported transaction attribute inserted into the PG DD.</td>
</tr>
<tr>
<td>numval</td>
<td>number(9,0)</td>
<td>Attribute’s numeric value, for example for a given transaction’s SYNCELEVEL attribute 0.</td>
</tr>
<tr>
<td>charval</td>
<td>varchar2(64)</td>
<td>Attribute’s character value; for example, a given transaction’s TPNAME attribute, the CICS transaction name.</td>
</tr>
<tr>
<td>dateval</td>
<td>date</td>
<td>Attribute’s date value. Probably always null; included for completeness.</td>
</tr>
</tbody>
</table>
**pga_trans_calls**

The pga_trans_calls table relates all calls available with any single transaction to each specific call definition through a call ID number.

An entry exists in the pga_trans_calls table for each PL/SQL call referenced in a transaction definition through the CALL(cname,...) operand. One row per transaction call; no duplicates.

The column, type and content information for pga_trans_calls is presented in the following table:

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>trans#</td>
<td>number(9,0)</td>
<td>Transaction id number from pga_trans(trans#). Primary key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Set from an Oracle sequence object for transaction inserted into the PG DD.</td>
</tr>
<tr>
<td>seq#</td>
<td>number(9,0)</td>
<td>Sequence number of this call. Primary key.</td>
</tr>
<tr>
<td>call#</td>
<td>number(9,0)</td>
<td>Call id number in pga_call(call#). Foreign key. Copied from pga_call.call# for the referenced call when this transaction definition was inserted or updated.</td>
</tr>
</tbody>
</table>
**pga_call**

The pga_call table relates all calls that are available for all defined transactions, to a unique call id number and PL/SQL remote procedural call (RPC) name. One entry exists in this table for each PL/SQL call (defined in a DEFINE CALL statement).

One row per call, duplicates are possible when multiple transactions make identical calls. The plsrpc specification must be unique within the Oracle server which makes the calls, and rows are uniquely distinguished by call#.

The column, type and content information for pga_call are presented in the following table:

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>cname</td>
<td>varchar2(48)</td>
<td>Call name for PGAU reference;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Primary key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max length =&gt; COBOL name string length</td>
</tr>
<tr>
<td>plsrpc</td>
<td>varchar2(30)</td>
<td>RPC call name for reference in PL/SQL (public procedure to be generated).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max length =&gt; PL/SQL RPC name length</td>
</tr>
<tr>
<td>updtdate</td>
<td>date</td>
<td>Audit trail date/time of record’s last update.</td>
</tr>
<tr>
<td>updtuser</td>
<td>varchar2(30)</td>
<td>Audit trail user id/program which last updated this record.</td>
</tr>
<tr>
<td>version</td>
<td>number(9,0)</td>
<td>Version identification of this entry, because multiple archived or invalid entries might exist and be kept for possible future reactivation. Primary key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Set from an Oracle sequence object for call version inserted into PG DD.</td>
</tr>
<tr>
<td>call#</td>
<td>number(9,0)</td>
<td>Call id number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Foreign key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pga_trans_calls(call#), pga_call_parm(call#).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Set from an Oracle sequence object for each call inserted into the PG DD.</td>
</tr>
</tbody>
</table>
**pga_call_parm**

The `pga_call_parm` table relates all parameters of any single transaction call to the data definitions describing each parameter.

One entry exists in the `pga_call_parm` table for each parameter on a call in the PARMS() operand of the PG AU DEFINE CALL statement. One row per parameter, duplicates allowed when multiple calls (in the `pga_call` table) refer to the same parameters.

The following table presents the column, type and content information for `pga_call_parm`:

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>call#</td>
<td>number(9,0)</td>
<td>Call number for the referencing call from <code>pga_calls</code>. Primary key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Set from an Oracle sequence object for each call inserted into the PG DD.</td>
</tr>
<tr>
<td>parm#</td>
<td>number(9,0)</td>
<td>Position in the PARMS() argument of DEFINE CALL operation (1,2,3...).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Primary key.</td>
</tr>
<tr>
<td>cmode</td>
<td>varchar2(6)</td>
<td>Call mode of this parameter; one of the values in <code>pga_data_modes</code>. For example: 'IN', 'OUT', 'IN OUT'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max length =&gt; 'IN OUT' string length</td>
</tr>
<tr>
<td>data#</td>
<td>number(9,0)</td>
<td>Data definition # in <code>pga_data(data#)</code> of this item. Foreign key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>pga_data(data#),pga_data_values(data#)</code>. Copied from <code>pga_data.data#</code> for the data item when this call/parm definition was inserted or updated.</td>
</tr>
</tbody>
</table>
pga_data
The pga_data table defines each data item used as a parameter in a call and relates the remote host data name to its PL/SQL variables and any component subfields or clauses within each data item (if the data item is an aggregate, such as a record). Each data item might have attributes related to it through its corresponding field definition. Even atomic data items have a single row in the pga_field table.

One row exists in the pga_data table for each data item defined by a PGAU DEFINE DATA or REDEFINE DATA statement.

The following table presents the column, type and content information for pga_data:
<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>comp#</td>
<td>number(9,0)</td>
<td>Compiler id number; Foreign key. (pga_compiler(comp#). Set from pga_compiler(comp#) based on the language parameter specified on the DEFINE DATA statement when the data definition is inserted.</td>
</tr>
<tr>
<td>compopts</td>
<td>varchar2(100)</td>
<td>Compiler options from the COMPOPTS keyword on the DEFINE DATA statement.</td>
</tr>
<tr>
<td>dname</td>
<td>varchar2(255)</td>
<td>Name from the DEFINE statement; Primary key. Max length =&gt; COBOL name length</td>
</tr>
<tr>
<td>plsdvar</td>
<td>varchar(30)</td>
<td>PL/SQL variable name of data item for reference in PL/SQL. Max length =&gt; PL/SQL variable length</td>
</tr>
<tr>
<td>version</td>
<td>number(9,0)</td>
<td>Version number of this entry. Set from an Oracle sequence object for data version inserted into the PGADD.</td>
</tr>
<tr>
<td>updtdate</td>
<td>date</td>
<td>Audit-trail date/time this control record last updated.</td>
</tr>
<tr>
<td>updtuser</td>
<td>varchar2(30)</td>
<td>Audit-trail user id/program which last updated this record.</td>
</tr>
<tr>
<td>usage</td>
<td>varchar2(6)</td>
<td>Default usage of this data item: PASS, SKIP, NULL, ASIS. Used primarily by PGAU REPORT. Max length =&gt; 4-char string length</td>
</tr>
<tr>
<td>data#</td>
<td>number(9,0)</td>
<td>Data definition number. Foreign key. (pga_call_parm(data#), (pga_field(data#) Set from an Oracle sequence object.</td>
</tr>
</tbody>
</table>
pga_fields

The pga_fields table defines each field within a data item and relates the remote host data field to its PL/SQL variables or nested records. Each field item might have attributes related to it (by field#) in the pga_data_attr and pga_data_values tables.

One row exists in the pga_fields table for each atomic item, field, clause, or nested record defined by a PGAU DEFINE DATA statement. Several rows would exist (related by a single data# and incrementing fld#) to define an aggregate data item, one row per field or group.

The following table presents the column, type and content information for pga_fields:

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>data#</td>
<td>number(9,0)</td>
<td>Data definition number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Primary key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(pga_data(data#), pga_call_parm(data#). Set from an Oracle sequence object.</td>
</tr>
<tr>
<td>fname</td>
<td>varchar2(255)</td>
<td>Extracted or derived name of a field if dname defines aggregate data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max length =&gt; COBOL name length</td>
</tr>
<tr>
<td>plsfvar</td>
<td>varchar2(30)</td>
<td>PL/SQL variable name of subfield in aggregate data for reference in PL/SQL. Max length =&gt; PL/SQL variable length</td>
</tr>
<tr>
<td>updtdate</td>
<td>date</td>
<td>Audit-trail date/time this control record last updated.</td>
</tr>
<tr>
<td>updtuser</td>
<td>varchar2(30)</td>
<td>Audit-trail user id/program which last updated this record.</td>
</tr>
<tr>
<td>fld#</td>
<td>number(9,0)</td>
<td>Clause or field within data definition id no.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Foreign key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pga_data_values(fld#). Set from an Oracle sequence object.</td>
</tr>
<tr>
<td>pos#</td>
<td>number(9,0)</td>
<td>Relative position number of each field defined within an aggregate data item (for example, 1, 2, 3, and so on) or NULL if data is atomic.</td>
</tr>
<tr>
<td>Column</td>
<td>Type</td>
<td>Content</td>
</tr>
<tr>
<td>---------</td>
<td>---------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>usage</td>
<td>varchar2(6)</td>
<td>Usage of this data field:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'PASS', 'SKIP', 'NULL', 'ASIS'.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max length =&gt; 4-char string length</td>
</tr>
<tr>
<td>mask</td>
<td>varchar2(30)</td>
<td>Datatype or Mask value. For example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'S9(4)'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'X(24)'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'VARCHAR2(24)'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'BINARY_INTEGER(16)'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When NULL, item defined is assumed to be a COBOL group or PL/SQL nested record.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max length =&gt; arbitrarily chosen</td>
</tr>
<tr>
<td>maskopts</td>
<td>varchar2(100)</td>
<td>Datatype or Mask options value. For example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'USAGE COMP-4'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'DISPLAY'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max length =&gt; arbitrarily chosen</td>
</tr>
</tbody>
</table>
PGA_DATA_ATTR

The pga_data_attr table defines all possible data attribute names allowed by PGA and relates each attribute name to a number and type, by which the value of this attribute for a specific data item can be selected from pga_data_values.

The pga_data_attr table is also used for integrity checks of data attributes when new data items are defined.

There is one entry in the pga_data_attr table for every possible attribute name to which any PGA supported data item might relate.

The following table presents the column, type and content information for pga_data_attr:

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>varchar2(16)</td>
<td>Character string name of attribute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Primary key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Contains:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;LEVEL&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;RENAMEMF&quot; (renames member first)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;RENAMEML&quot; (renames member last)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;REMAPSMF&quot; (redefines member first)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;REMAPSML&quot; (redefines member last)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;REMAPSWM&quot; (redefines when member)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;REMAPSWC&quot; (redefines when char value)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;REMAPSWN&quot; (redefines when num value)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;REPGRPFF&quot; (occurs n)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;REPGRPVF&quot; (odo first n)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;REPGRPVML&quot; (odo last n)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;REPGRPVW&quot; (odo depending member)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;REPGRPKA&quot; (either Key Asc name)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;REPGRPKD&quot; (either Key Desc name)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;REPGRPIX&quot; (either index name)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;PLSTYPE&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;JUST&quot; (justified char data)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;SYNC&quot; (aligned aggregate data)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;LOCAL_LANGUAGE&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;REMOTE_LANGUAGE&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;LENGTH&quot; (LENGTH IS variable)</td>
</tr>
</tbody>
</table>

Max length => attr name string lengths
A row exists in the pga_data_values table for each attribute of each data item defined by each data definition.

The following table presents the column, type and content information for pga_data_values:

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr#</td>
<td>number(9,0)</td>
<td>Attribute id assigned.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Foreign key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pga_data_values(attr#). Set from an Oracle sequence object for each supported data attribute inserted into the PG DD.</td>
</tr>
<tr>
<td>coltype</td>
<td>varchar2(4)</td>
<td>Type of Oracle column from which attribute value is retrieved from pga_data_values. For example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'NUM' =&gt; pga_data_values(numval)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'CHAR' =&gt; pga_data_values(charval)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'DATE' =&gt; pga_data_values(dateval)</td>
</tr>
<tr>
<td>required</td>
<td>char(1)</td>
<td>If not null, required keyword.</td>
</tr>
</tbody>
</table>

**pga_data_values**

A row exists in the pga_data_values table for each attribute of each data item defined by each data definition.

The following table presents the column, type and content information for pga_data_values:

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>fld#</td>
<td>number(9,0)</td>
<td>Data Field Definition number from pga_data(fld#). Primary key.</td>
</tr>
<tr>
<td>attr#</td>
<td>number(9,0)</td>
<td>Attribute id from pga_data_attr(attr#). Primary key.</td>
</tr>
<tr>
<td>numval</td>
<td>number(9,0)</td>
<td>Attribute’s numeric value. For example: number for &quot;LEVEL&quot; number for &quot;REMAPSWN&quot; (redefines) number for &quot;REPGRPFF&quot; (occurs n) number for &quot;REPGRPVF&quot; (odo first n) number for &quot;REPGRPVL&quot; (odo last n)</td>
</tr>
</tbody>
</table>

If a non-numeric attribute, this item is NULL.
<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>charval</td>
<td>varchar2(40)</td>
<td>Attribute’s character value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fname for &quot;RENAMEMF&quot; (renames first)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fname for &quot;RENAMEML&quot; (renames last)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fname for &quot;REMAPSMF&quot; (redefines first)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fname for &quot;REMAPSML&quot; (redefines last)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fname for &quot;REMAPSWM&quot; (redefines when)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fname for &quot;REPGPVM&quot; (odo member)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>string for &quot;REMAPSWC&quot; (redefines)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>string for &quot;REPGRPKA&quot; (occurs key)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>string for &quot;REPGRPKD&quot; (occurs key)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>string for &quot;REPGRPIX&quot; (occurs index)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>string for &quot;PLSTYPE&quot; (PL/SQL data type)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>string for &quot;JUST&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>string for &quot;SYNC&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>string for &quot;REMOTE_LANGUAGE&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fname for &quot;LENGTH&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If a non-character attribute, this item is NULL.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Max length =&gt; NLS_charset string length</td>
</tr>
<tr>
<td>dateval</td>
<td>date</td>
<td>Attribute’s date value. Always null, included for completeness.</td>
</tr>
<tr>
<td>qual</td>
<td>number (9,0)</td>
<td>Qualified name number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Foreign key.</td>
</tr>
</tbody>
</table>
The PGAU generates complete and operational TIPs for most circumstances. TIP internals information is provided to assist you in diagnosing problems with PGAU-generated TIPs, and in writing custom TIPs, if you choose to do so.

This appendix refers to a sample called pgadb2i. The source for this TIP is in file pgadb2i.sql in the $ORACLE_HOME/pg4appc/demo/CICS directory.

This appendix contains the following sections:

- Background Reading on page B-2
- PL/SQL Package and TIP File Separation on page B-2
- TIP Call to APPC Call Relationship on page B-5
- Oracle Procedural Gateway for APPC Restrictions on page B-9
- TIP Specification on page B-10
- TIP Body on page B-11
- TIP Initialization on page B-18
- TIP Init and Term Functions on page B-19
Background Reading

Several topics are important to understanding TIP operation and development; following is a list of concepts that are key to TIP operation and suggested sources to which you can refer for more information.

PL/SQL Packages: Refer to the *PL/SQL User’s Guide and Reference* for information.

PGA Application Concepts: Refer to Chapter 4, "Client Application Development" for information.

PGA RPC Interface: Refer to Appendix C, "Gateway RPC Interface" for information.

PGA UTL_PG/UTL_RAW Interface: Refer to Appendix D, "The UTL_PG and UTL_RAW Interfaces" for information.

remote host APPC transactions: Refer to your remote host vendor publications for information.

**PL/SQL Package and TIP File Separation**

PGAU GENERATE writes each output TIP into a standard PL/SQL package specification file and body file. This separation is beneficial and important. Refer to the *Oracle9i Application Developer’s Guide* and the *PL/SQL User’s Guide and Reference* for more information. Also refer to “GENERATE” in Chapter 2, "Procedural Gateway Administration Utility" for more information about building the PL/SQL package.

TIPs are PL/SQL packages. Any time a package specification is recompiled, all objects which depend on that package are invalidated and implicitly recompiled as they are referenced, even if the specification did not change.

Objects which depend on a TIP specification include client applications that call the TIP to interact with remote host transactions.

It might be important to change the TIP body for the following reasons:

- Oracle Corporation ships maintenance which affects the TIP body
- Oracle Corporation ships maintenance for the UTL_RAW or UTL_PG conversion functions upon which the TIP body relies. Refer to Appendix D,
"The UTL_PG and UTL_RAW Interfaces" for more detailed information about these functions.

- if the remote host network or program location parameters have changed. Refer to "DEFINE TRANSACTION" in Chapter 2, "Procedural Gateway Administration Utility" for more information

Provided that the TIP specification does not need to change or be recompiled, the TIP body can be regenerated and recompiled to pick up changes without causing invalidation and implicit recompilation of client applications that call the TIP.

It is for this reason that PGAU now separates output TIPs into specification and body files. Refer to "GENERATE" in Chapter 2, "Procedural Gateway Administration Utility" for a discussion of file identification.

**Independent TIP Body Changes**

Independent TIP body changes are internal and require no change to the TIP specification. Examples of such changes include: a change in UTL_RAW or UTL_PG conversions, inclusion of diagnostics, or a change to network transaction parameters.

In these cases, when PGAU is used to regenerate the TIP, the new TIP specification file can be saved or discarded, but should not be recompiled. The new TIP body should be recompiled under SQL*Plus.

Provided that the TIP body change is independent, the new body compilation completes without errors and the former TIP specification remains valid. To determine if a specification has remained valid, issue the following from SQL*Plus:

```
SQL> column ddl_date format A22 heading 'LAST_DDL'
SQL> select object_name,
2   object_type,
3   to_char(last_ddl_time,'MON-DD-YY HH:MM:SS') ddl_date,
4   status
5   from all_objects where owner = 'PGAADMIN'
6   order by object_type, object_name;
```

<table>
<thead>
<tr>
<th>OBJECT_NAME</th>
<th>OBJECT_TYPE</th>
<th>LAST_DDL</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGADB2I</td>
<td>PACKAGE</td>
<td>NOV-24-1999 09:09:13</td>
<td>VALID</td>
</tr>
<tr>
<td>PGADB2I</td>
<td>PACKAGE BODY</td>
<td>NOV-24-1999 09:11:44</td>
<td>VALID</td>
</tr>
<tr>
<td>DBZIDRIV</td>
<td>PROCEDURE</td>
<td>DEC-30-1999 12:12:14</td>
<td>VALID</td>
</tr>
<tr>
<td>DBZIDRVM</td>
<td>PROCEDURE</td>
<td>DEC-30-1999 12:12:53</td>
<td>VALID</td>
</tr>
<tr>
<td>DBZIFORM</td>
<td>PROCEDURE</td>
<td>DEC-14-1999 11:12:24</td>
<td>VALID</td>
</tr>
</tbody>
</table>
The LAST_DDL column is the date and time at which the last DDL change against the object was done. It shows that the order of compilation was:

PGADB2I PACKAGE (the specification)
DB2IDRVM PROCEDURE (1st client application depending on PGADB2I)
DB2IFORM PROCEDURE (2nd client application depending on PGADB2I)
DB2IDRIV PROCEDURE (3rd client application depending on PGADB2I)
PGADB2I PACKAGE BODY (a recompilation of the body)

Note that the recompilation of the body does not invalidate its dependent object, the specification, or the client application indirectly.

Dependent TIP Body or Specification Changes

You can also change the data structures or call exchange sequences of the remote host transaction. However, this kind of change is exposed to dependent client applications because the public datatypes or functions in the TIP specification will also change and necessitate recompilation, which in turn causes the Oracle server to recompile such dependent client applications.

Note that the recompilation of the specification has invalidated its dependent objects, the three client applications in addition to the package body. To complete these changes, the body must be recompiled to bring it into compliance with the specification and then the three client applications could be compiled manually, or the Oracle server compiles them automatically as they are referenced.

If the client applications are recompiled by the Oracle server as they are referenced, there is a one-time delay during recompilation.
Recompilation errors in the client application, if any, are due to:

- customer changes in the client application source
- an altered PG DD definition for the TIP if the TIP has been regenerated
- the wrong version being generated from multiple transaction entry versions saved in the PG DD if the TIP has been regenerated

**Inadvertent Alteration of TIP Specification**

If you make a mistake when you generate a tip (for example, if you alter a PG DD transaction definition, or if you’ve inadvertently specified the wrong version during regeneration), then the recompiled body will not match the stored specification; as a result, the Oracle integrating server would invalidate the specification and any dependent client applications.

You may have to regenerate and recompile the TIP and its dependent client applications to restore correct operation.

Refer to "Listing Dependency Management Information," in the *Oracle9i Server Application Developer’s Guide* for more information.

**TIP Call to APPC Call Relationship**

To initialize and terminate a connection to the remote transaction program (RTP) and to process the buffers exchanged through the PGAXFER RPC, you are required to know:

- the remote transaction program’s data formats;
- the remote transaction program’s network interfaces; and
- gateway RPC calling conventions

Proper use of the UTL_RAW and UTL_PG functions is also required, to perform conversion of the data exchanged with the remote transaction. Refer to Appendix D, “The UTL_PG and UTL_RAW Interfaces” for detailed information about using these functions. Typically, this information is available to the remote transaction program developer and to the gateway administrator.

You are expected to know:

- PL/SQL datatypes
- database link, DBLINK, names assigned to gateway nodes
TIP Call to APPC Call Relationship

- PL/SQL procedure programming methods

The resulting TIP should provide the function calls you need. Once a TIP is developed and stored in the Oracle integrating server database, it can be called by other client applications which need to access and control the remote transaction program for which the TIP was written.

You are expected to know how to control the remote transaction program and step it through all proper sequences. For example, using EMPT for employee update might first require an input to select one of several possible employee tables to update. Then each such choice results in further inputs and outputs specific to the type of table being updated. You are expected to know how to navigate among these choices.

The TIP is a transaction access method or transaction call runtime library which assists the application in controlling EMPT, without requiring you to know all of the specifics of gateway RPCs and data conversion.

TIP Function Calls and Remote Transaction Program APPC Calls

A remote transaction program and its related TIP with client application must correspond on two key requirements:
- parameter datatype conversion
- APPC send/receive synchronization

Parameter Datatype Conversion

The requirement to synchronize parameter datatype conversion means that the TIP function logic must perform all data conversion and buffering into the format expected by the receiving remote transaction program. The TIP function logic must also perform all data unbuffering from the format supplied by the sending remote transaction program and then convert the data to PL/SQL datatypes returned to the TIP caller, the client application, as PL/SQL parameters.

Assume the remote transaction program receives and then transmits a 75-byte record containing three fields:
- IBM 370 packed decimal number 10 bytes long
- IBM EBCDIC fixed length character string 30 bytes long
- IBM EBCDIC variable length character string <= 35 bytes
Each field corresponds to a passed PL/SQL parameter. The TIP function must send logic, before calling the PGAXFER remote procedural call (RPC), to:

- convert the first passed PL/SQL IN parameter to 10 bytes of IBM packed decimal, and insert it in positions 1-10 of the send buffer.
- convert the second passed PL/SQL IN parameter to 30 bytes of IBM EBCDIC character data, probably blank padded on the right, and insert it in positions 11-40 of the send buffer.
- compute the length of the third passed PL/SQL IN parm, convert that value to a 2-byte IBM unsigned binary integer, and insert that in positions 41-42 of the send buffer.
- convert the third passed PL/SQL IN parameter to not more than 33 bytes of EBCDIC character data, and insert it in positions 43 through the end of the buffer. If fewer then 33 bytes of data are being sent, then the remainder of the buffer must be padded in compliance with the expectations of the remote transaction program.

Each of the above three conversions requires different UTL_RAW and UTL_PG functions depending on whether the PL/SQL parameter is NUMBER, CHAR, VARCHAR2, or DATE.

After the PGAXFER RPC call, the record received from the remote transaction program must be converted in the reverse steps for each field:

- extract raw bytes from predetermined positions in the buffer, based on positions and lengths as dictated by the remote transaction program.
- convert the extracted raw byte substrings into proper datatypes using the UTL_RAW and UTL_PG functions and assign the values to the PL/SQL OUT parameters being returned.

These conversions differ based on the returned datatypes (NUMBER, CHAR, VARCHAR2, or DATE).

**APPC Send/Receive Synchronization**

The requirement to synchronize APPC SENDs and RECEIVEs means that when the remote transaction program expects data parameters to be input, it issues APPC RECEIVEs to read the data parameters. Accordingly, the TIP must cause the gateway to issue APPC SENDs to write the data parameters to the remote transaction program. The reverse is also true; the TIP must cause the gateway to issue APPC RECEIVEs when the remote transaction program issues APPC SENDs.
The number of data parameters exchanged between the TIP and the gateway on each call can vary at the user’s discretion, as long as the remote transaction program’s SEND/RECEIVE requests are satisfied. For example, the remote transaction program data exchange sequence might be:

- **APPC SEND** 5 fields (field1-field5)
- **APPC RECEIVE** 1 fields (field6)
- **APPC SEND** 1 field (field7)
- **APPC RECEIVE** 3 fields (field8 - field10)

The resulting TIP/application call sequence could be:

```plaintext
tip_call1(parm1 OUT, <-- APPC SEND field1 from remote TP
parm2 OUT, <-- APPC SEND field2 from remote TP
parm3 OUT); <-- APPC SEND field3 from remote TP

Tip_call2(parm4 OUT, <-- APPC SEND field4 from remote TP
parm5 OUT); <-- APPC SEND field5 from remote TP
```

**Note:** At this point, the total data length expected by PL/SQL parameters 1-5 must equal the total data length sent in remote transaction program fields 1-5. Because an Oracle NUMBER datatype and a COBOL datatype can be different lengths for the same numeric value (their internal representations differ), a simple sum of lengths does not suffice. The internal representation of the remote data format must be known and embodied in the TIP conversion logic, as discussed in Step 4 of “TIP Body” on page B-11.

```plaintext
tip_call3(parm6 IN OUT); --> APPC RECEIVE field6 in remote TP
 <-- APPC SEND field7 from remote TP

Tip_call4(parm8 IN, --> APPC RECEIVE field8 into remote TP
parm9 IN, --> APPC RECEIVE field9 into remote TP
parm10 IN); --> APPC RECEIVE field10 into remote TP
```

**Note:** At this point, the total data length expected by PL/SQL parameters 8-10 must equal the total data length received in remote transaction program fields 8-10.
Notice that the remote transaction program’s first five written fields are read by two separate TIP function calls (tip_call1 and tip_call2). This could also have been equivalently accomplished with five TIP function calls of one OUT parameter each, or a single TIP function call with five OUT parameters. Then the remote transaction program’s first read field (field6) and subsequent written field (field7) corresponds to a single TIP function call (tip_call3) with a single IN OUT parameter (parm6).

This use of a single IN OUT parameter implies that the remote transaction program’s datatype for field6 and field7 are both the same and correspond to the conversion performed for the datatype of parm6. If field6 and field7 were of different datatypes, then they would have to correspond to different PL/SQL parameters (for example, parm6 IN and parm7 OUT). However, they could still be exchanged as two parameters on a single TIP call or one parameter each on two TIP calls.

Lastly, the remote transaction program’s remaining three RECEIVE fields are supplied by tip_call4 parms 8-10. They also could have been done with three TIP calls passing one parameter each or two TIP calls passing one parameter on one call and two parameters on the other, in either order. This flexibility permits the user to define the correspondence between the remote transaction program’s operation and the TIP function calls in whatever manner best suits the user.

Oracle Procedural Gateway for APPC Restrictions

- The PGAXFER RPC processes the send buffers first, followed by the receive buffers.

Thus, a remote transaction program expecting to send one field, then receive one field must correspond to separate TIP calls which do:

```
tip_call0(parm0 OUT); <-- APPC SEND outfield from remote transaction program
```

PGAXFER RPC checks first for parameters to send but finds none and proceeds to receive parameters.

```
tip_call1(parml IN);  --> APPC RECEIVE infield to remote transaction program
```

PGAXFER RPC processes parameters to send and then checks for parameters to receive, but finds none and completes.

A single TIP call with an OUT parameter followed by an IN parameter does not work because the IN parameter is processed first regardless of its position in the parameter list.
An APPC send buffer, sent from the remote host, must be received in full; the expected length to be received must be at least as long as the data sent from the remote host. A greater length can be expected though a shorter length is actually received. The TIP must determine the actual length received (refer to the getrcvla subprogram in any of the sample TIPs), and adjust the conversions to PL/SQL variables accordingly, possibly right padding the data or using a VARCHAR2 depending on circumstances. Refer to "PGAXFER" in Appendix C, "Gateway RPC Interface" for a discussion of receive lengths. The gateway generates an error if the remote host attempts to send more data than was expected to be received by the TIP.

PGAU-generated TIPs also impose the restriction that each individual PL/SQL parameter, whether atomic or aggregate, correspond to one complete APPC send or receive at the remote host. A single parameter passed from the client application is mapped to a single APPC logical record which the gateway transfers in a single APPC send or receive. A parameter is neither split into multiple sends, nor filled from multiple receives.

The only exception is a parameter greater than 32K in length. In this case, the TIP builds additional send/receive "pseudo-parameters" internally to instruct the gateway to exchange additional buffers until the total length is satisfied. The remote host transaction must be programmed to expect and exchange multiple buffers for data fields exceeding 32K.

The exact count of bytes sent and received must correspond on both ends of the transfer before transmission in the other direction can begin.

For example, if the remote transaction program sends 100 bytes, then the user’s application and TIP calls must cause the PGAXFER RPC to receive 100 bytes. After the application and TIP have received 100 bytes from the PGAXFER RPC, then the next PGAXFER RPC call can send data to the remote transaction program, provided that the remote transaction program is ready to receive. The byte counts transferred to the remote transaction program must match exactly at both ends before reversing direction again.

**TIP Specification**

Use the following guidelines to develop the package specification.

1. Declare public functions which are called by applications.
   - remote transaction program initiate function
     Declare a transaction instance number as an OUT parameter.
Refer to "PGAINIT and PGAINIT_SEC" in Appendix C, "Gateway RPC Interface".

- remote transaction program terminate function
  Declare a transaction instance number as an IN parameter.
  Refer to "PGATERM" in Appendix C, "Gateway RPC Interface".

- as many remote transaction program data transfer functions as desired
  Declare a transaction instance number as an IN parameter.
  Declare user IN, IN OUT, or OUT parameters for data transfer.
  Refer to "PGAXFER" in Appendix C, "Gateway RPC Interface".

2. Declare public data parameters.
   - Declare PL/SQL record types to be passed as parameters on calls to public functions and which might be referenced in calling applications as package_name.rec_type.

TIP Body

Use the following guidelines to develop the package body.

1. Declare private control variables.
   - PL/SQL table of RAW(12) elements to store APPC conversation IDs, an index into the table, and a constant to set table entries to zero. For example:

     ```
     TYPE tran_cid_tbl IS TABLE OF RAW(12) NOT NULL INDEX BY BINARY_INTEGER;
     trannum  BINARY_INTEGER;
     cidraw0s RAW(12) := HEXTORAW('000000000000000000000000');
     trancid  tran_cid_tbl; /* APPC Conv ID returned */
     ```

   - constants to define remote transaction program name and network connections.

   - RAW(2048) variables for pre-generated NUMBER_TO_RAW or RAW_TO_NUMBER conversion formats.
     Refer to Appendix D, "The UTL_PG and UTL_RAW Interfaces".

   - gateway remote procedural call (RPC) parameters for the send/receive buffers.
any shared variables used by private internal functions.

2. Declare private internal functions used as common subroutines by the public functions.
   - update PGAXFER RPC send parameter lengths list (see updsndll in pgadb2i.sql).
   - update PGAXFER RPC receive parameter lengths list (see updrcvll in pgadb2i.sql).
   - extract PGAXFER RPC received parameter actual length (see getrcvla in pgadb2i.sql).
   - display conversion warnings issued from formatted NUMBER_TO_RAW or RAW_TO_NUMBER functions (see showwngs in pgadb2i.sql).

3. Public remote transaction program init function.
   a. Declare public function and public data parameter list with a transaction instance number as an OUT parameter.
   b. Initialize any private control variables as needed.
   c. Increment the APPC conversation ID table index.
   d. Call the PGAINIT RPC passing the remote transaction program name and network connection information of the targeted remote transaction program to be initiated.
      Specify the Oracle database link assigned to the gateway node which communicates with the remote transaction program.
      The call is of the form:
      ```sql
      PGAINIT@dblink(convid,
      tpname,
      luname,
      modename,
      profname,
      synchlevel);
      ```
      Refer to "PGAINIT and PGAINIT_SEC" remote procedural calls in Appendix C, "Gateway RPC Interface" for more information.
   e. Upon return from the PGAINIT RPC, save the APPC conversation ID in the current APPC conversation ID table entry and assign the table index to the transaction instance number OUT parameter returned to the calling
application so that the application might designate this particular conversation on future gateway RPC calls.

f. Return to the calling application.

4. Public remote transaction program data exchange functions.
   a. Declare public function and public data parameters exchanged on the call, including in the parameter list, redeclarations of the exact same type as made in the Package Specification section for:
      – a transaction instance number as an IN parameter.
      – user IN, IN OUT, or OUT parameters for data transfer.
   b. Reinitialize PGAXFER RPC parameters for the send/receive buffers.
   c. Assemble IN parameters into the PGAXFER RPC send buffer. Refer to:
      – \texttt{pgadb2i.sql} function \texttt{pgadb2i\_main} for an example of PGAXFER send buffer construction.
      – and refer to Appendix D, "The UTL\_PG and UTL\_RAW Interfaces" for a description of the UTL\_RAW and UTL\_PG functions. Also refer to Appendix C, "Gateway RPC Interface" for descriptions of the PGAXFER RPC calling conventions and parameter format.

For each IN or IN OUT parameter passed by the calling application:
   – convert the application parameter into the send buffer
   – put its length in the send lengths list
   – increment the send count

Local Application data format expected:
   – Atomic or discrete application parameters are converted separately, one at a time, into the send buffer. Aggregate application data, such as a PL/SQL record, or other RAW byte string containing fields, must have each field converted separately. An entire aggregate cannot be converted as a single item; it must be separated into its component fields first.
   – If the aggregate is a PL/SQL record, then its component fields can each be referenced as ‘record\_name.field\_name’ in the algorithms below. If the aggregate is a RAW, CHAR, or VARCHAR, then the SUBSTR function must be used to extract each component field based on predetermined offsets and lengths.
Remote Transaction Program data format expected:

- RAW data sent must be in the exact internal representation as expected by the remote transaction program, as if the data were native to the remote transaction program, and positioned in the send buffer exactly as the remote transaction program expects it. Aggregate data, like COBOL records or C structures, must have its groups, fields, and members positioned and aligned at the offsets and boundaries (with or without pad or filler bytes) exactly as the remote transaction program expects it.

Because of this requirement for composing data in its exact native format, you should seek advice or assistance from the remote transaction program developer. Remote transaction program listings of the data structures being exchanged are useful.

Reference atomic variables as passed, or PL/SQL record fields as "record_name.fieldname", or if other aggregate, use SUBSTR to extract needed fields. Then when converting to RAW (into send buffer) from a PL/SQL datatype of:

- CHAR, use the following in order:
  - CAST_TO_RAW - recast user CHAR data into raw
  - CONVERT - convert raw data to remote NLS codepage
  - OVERLAY - place the result in the send buffer
- VARCHAR2, use the following in order:
  - CAST_TO_RAW - recast user VARCHAR2 data into raw
  - CONVERT - convert raw data to remote NLS codepage
  - OVERLAY - place the result in the send buffer
- NUMBER, use the following in order
  - NUMBER_TO_RAW - change user NUMBER data to COBOL data
  - OVERLAY - place the result in the send buffer

Store the final RAW byte length of the converted parameter into its respective position in the send lengths list (this is the exact byte length as expected by the remote transaction program) and increment the send count.
d. Assemble OUT parameter lengths into the PGAXFER RPC receive buffer. Refer to:
   - `pgadb2i.sql` function `pgadb2i_main` for an example of PGAXFER receive buffer construction.
     For each IN OUT or OUT parameter passed by the calling application, put its length in the receive lengths list and increment the receive count.
   - Refer to Appendix C, "Gateway RPC Interface" for descriptions of the PGAXFER RPC calling conventions and parameter format.

e. Call the PGAXFER RPC.

Call the PGAXFER RPC passing the APPC conversation ID indexed by the transaction instance number of the targeted remote transaction program, and the send/receive parameters as previously initialized and built. Specify the Oracle database link assigned to the gateway node which communicates with the remote transaction program. The call is of the form:

```
PGAXFER@dblink (convid,  
    sndbuf,  
    sndbuf1,  
    sndlns,  
    rcvbuf,  
    rcvbuf1,  
    rcvlns) ;
```

Refer to Appendix C, "Gateway RPC Interface" for descriptions of the PGAXFER RPC.

f. Disassemble OUT parameters from the PGAXFER RPC receive buffer. Refer to:
   - `pgadb2i.sql` function `pgadb2i_main` for an example of PGAXFER receive buffer dissection.
     Previous processing should have built a receive lengths list that corresponds to IN OUT or OUT parameters passed from the calling application. The actual receive count and actual lengths of received parameters have been placed in the received lengths list by the PGAXFER RPC upon return.
     For the count of received parameters, extract each received parameter from the receive buffer using the actual length from the received lengths list,
computing its starting offset from the previous offset plus the previous length. Then convert the extracted native parameter into an Oracle PL/SQL variable as follows.

Received parameters can be either atomic or aggregate datatypes. Atomic implies one parameter per Oracle variable. Aggregates, such as records or structures, must have each of their component fields extracted and separately converted to a PL/SQL datatype, one field per Oracle variable.

Remote Transaction Program data format received:

- RAW data received is in the exact internal representation as processed by the remote transaction program, as if the data were native to the remote transaction program, and positioned in the receive buffer exactly as the remote transaction program uses it.
- Aggregate data, like COBOL records or C structures, has its groups, fields, and members positioned and aligned at the offsets and boundaries (with or without pad or filler bytes) exactly as the remote transaction program processes it.
- Because of this requirement for decomposing data from its native format, you should seek advice or assistance from the remote transaction program developer. Remote transaction program listings of the data structures being exchanged are useful.
- Local Application datatype expected:
  - Atomic or discrete application parameters are extracted separately, one at a time, from the receive buffer and then converted to PL/SQL variables.
  - Aggregate remote transaction program data, such as a COBOL record or C structure containing fields, must have each field converted separately. An entire aggregate cannot be converted to a PL/SQL record. The PL/SQL record must be composed from its component fields as each is converted.
  - If the aggregate is a PL/SQL record, then its component fields can each be referenced as "record_name.field_name" in the algorithms below. If the aggregate is a RAW, CHAR, or VARCHAR then the OVERLAY function must be used to insert each converted component field into the result aggregate based on predetermined offsets and lengths.

Use SUBSTR to extract the current receive parameter from the receive buffer, using the length from the receive lengths list, and computing the starting offset from the previous received parameter length plus 1.
Then when converting from RAW, from the receive buffer, to a PL/SQL
datatype of:

- CHAR, use the following in order:
  SUBSTR - extract raw data field from buffer
  CONVERT - convert raw data from remote NLS codepage
  CAST_TOVARCHAR2 - recast raw data as varchar2
  RPAD - right pad remainder of CHAR with blanks

- VARCHAR2, use the following in order:
  SUBSTR - extract raw data field from buffer
  CONVERT - convert raw data from remote NLS codepage
  CAST_TOVARCHAR2 - recast raw data as varchar2

- NUMBER, use the following in order:
  SUBSTR - extract raw COBOL field from buffer
  RAW_TO_NUMBER - change raw COBOL data to NUMBER

Then assign the result to an atomic variable as passed or to PL/SQL record
fields as "record_name.fieldname", or if other aggregate, use OVERLAY to
insert the result field.

- Refer to Appendix D, "The UTL_PG and UTL_RAW Interfaces". Refer also
to Appendix C, "Gateway RPC Interface" for descriptions of the PGAXFER
RPC calling conventions and parameter format.

g. Return to the calling application.

5. Public remote transaction program terminate function.
   a. Declare public function and public data parameter list with:
      - a transaction instance number as an IN parameter
      - a conversation termination type as an IN parameter
   b. Call the PGATERM RPC passing the APPC conversation ID indexed by the
      transaction instance number of the targeted remote transaction program to
      terminate and the type of termination to be performed.

      Specify the Oracle database link assigned to the gateway node which
      communicates with the remote transaction program.
The call is of the form:

\[ \text{PGATERM}@\text{dblink} \ (\text{convid}, '0') ; \]

Refer to Appendix C, "Gateway RPC Interface" for descriptions of the PGATERM RPC.

c. Upon return from the PGATERM RPC, reset the APPC conversation ID indexed by the transaction instance number to zeros.

d. Return to the calling application.

**TIP Initialization**

Use the following guidelines to initialize the package.

1. Initialize the private control variables declared in Step 1 of "TIP Body" on page B-11. These variables never change.
   a. Identify remote transaction program name and network information.
   b. Preprocess frequently used NUMBER_TO_RAW and RAW_TO_NUMBER datatype conversion formats.
      Refer to Appendix D, "The UTL_PG and UTL_RAW Interfaces" for descriptions of the following UTL_PG functions; Refer to \texttt{pgadb2i.sql} for examples of their use.
      - MAKE\_NUMBER\_TO\_RAW\_FORMAT
      - MAKE\_RAW\_TO\_NUMBER\_FORMAT
      - NUMBER\_TO\_RAW
      - RAW\_TO\_NUMBER

   APPC conversation ID table entries and gateway remote procedural call parameters change on every call. Package initialization of these variables is extra overhead and not sufficient during operation.
TIP Init and Term Functions

These function calls are always included in PGAU-generated TIPs. They issue the PGAINIT and PGATERM RPCs, respectively, to the gateway.

TIP Initialization Function

The client application calls the TIP initialization function as if it were any local PL/SQL function. For example:

```sql
mytran_number := 0;           /* first transaction started*/
rc = tip_init(mytran_number); /* and init transaction */
```

Note that the application does not pass any remote transaction program name or network connection information. The TIP has such information internally coded as constants. All the application needs is to call the proper TIP for the requested remote transaction program.

This call serves the following two purposes:

- To cause the PL/SQL package, the TIP, to be loaded and to perform the initialization logic programmed in the TIP initialization section.
  
  This includes:
  
  - setting remote transaction program name and network parameters.
  - preprocessing the formats for frequently used RAW_TO_NUMBER and NUMBER_TO_RAW conversions (refer to Appendix D, "The UTL_PG and UTL_RAW Interfaces").

- To cause the TIP init function to call the PGAINIT RPC, which in turn establishes communication with the remote transaction program and returns an APPC conversation ID to the TIP. The TIP then saves the APPC conversation ID in a table and returns the table index for the conversation to the application. This index is known to the application as a transaction instance number. Refer to "PGAINIT and PGAINIT_SEC" in Appendix C, "Gateway RPC Interface" for more information.

This transaction instance number must be passed to subsequent TIP transfer and terminate functions to identify to the gateway on which APPC conversation, and hence which iteration of a remote transaction program, the data is to be transmitted or communication terminated.

A single application can control multiple instances of the same remote transaction program (RTP) or multiple different RTPs, all concurrently. The APPC conversation
ID is the TIP and gateway mechanism for routing the applications call to the intended remote transaction program.

However, to simplify the application’s management of the transaction and to permit a single instance of a TIP to manage multiple conversations, the APPC conversation ID should be stored in an array of RAWs and the index to this particular element should be returned to the application as the transaction instance number.

The TIP must remember all active APPC conversation IDs and pass the correct one as the remote transaction program/conversation identifying parameter on every call. The TIP does this by using the transaction instance number passed from the application to index into the RAW array of APPC conversation IDs and passes the indexed APPC conversation ID to the gateway.

It is the application’s responsibility to remember the transaction instance number of each active transaction and pass the correct one to each TIP function called for that transaction.

**TIP Terminate Function**

The client application calls the TIP termination function as if it were any local PL/SQL function. For example:

```plsql
rc = tip_term(mytran_number); /* and terminate it */
mytran_number := 0; /* reset transaction instance to 0 */
```

After a transaction instance number has been passed on a TIP terminate call to terminate the transaction, or after the remote transaction program has abended, that particular transaction instance number might be forgotten and its corresponding entry in the RAW array of APPC conversation IDs reset.
To execute a remote transaction program using the Oracle Procedural Gateway for APPC, you must execute a PL/SQL program to call the gateway functions, using a remote procedural call (RPC). The gateway functions handle the initiation, data exchange and termination for an APPC conversation with the remote transaction program. The Oracle Procedural Gateway for APPC includes a tool, PGAU, to generate the PL/SQL packages (TIPs) automatically, based on definitions you provide in the form of COBOL record layouts and PGDL (Procedural Gateway Definition Language).

This appendix contains the following section:

- **Calling Gateway Functions to Execute Transaction Programs** on page C-2
Calling Gateway Functions to Execute Transaction Programs

The gateway functions are all executed through remote procedural calls (RPC). The functions are called from PL/SQL code as follows:

\[
\text{\texttt{function@dblink(parm1, parm2, ..., parmn);}}
\]

where:

- \textit{function} is the name of the function being called.
- \textit{dblink} is the name of a predefined database link to the gateway server on the UNIX system.
- \textit{parml, parm2, parmn} are the function-specific parameters described later in this appendix.

Calling a function in PL/SQL code with the \texttt{@dblink} notation following the function name is a remote procedural call.

PGAINIT and PGAINIT_SEC

PGAINIT and PGAINIT_SEC are remote procedural calls that initiate an APPC conversation with a specified transaction program. The difference between the two is that PGAINIT_SEC also includes the added capability of being able to set the APPC conversation security user ID and password to values other than the current Oracle user ID and password. Upon successful completion of either function, the conversation is ready to send data to the remote transaction program.

The following table presents the parameters that are passed to both PGAINIT and PGAINIT_SEC. It lists the type, datatype and description of each parameter:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONVID</td>
<td>OUT</td>
<td>RAW(12)</td>
<td>Conversation identifier returned by the PGAINIT function to be used to identify the conversation to the PGAXFER and PGATERM functions. After PGAINIT is called, this variable must never be modified, or results will be unpredictable.</td>
</tr>
</tbody>
</table>
### Calling Gateway Functions to Execute Transaction Programs

#### Gateway RPC Interface

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPNAME</td>
<td>IN</td>
<td>VARCHAR2(64)</td>
<td>Transaction program name of the remote transaction program with which a conversation is to be established. For most OLTPs, the name must be the transaction name as defined to the OLTP. This name can be from 1 to 64 characters in length.</td>
</tr>
<tr>
<td>LUNAME</td>
<td>IN</td>
<td>VARCHAR2(17)</td>
<td>The LU name of the OLTP under which the remote transaction program executes. This parameter is the fully-qualified LU name or alias and can be from 1 to 17 characters in length.</td>
</tr>
<tr>
<td>MODENAME</td>
<td>IN</td>
<td>VARCHAR2(8)</td>
<td>Logmode entry name of the logmode table entry on the remote system which defines the session characteristics for the APPC conversation. This name can be from 1 to 8 characters in length.</td>
</tr>
<tr>
<td>PROFNAME</td>
<td>IN</td>
<td>VARCHAR2(8)</td>
<td>Profile name of the SNA Side Information profile which defines the conversation. This name can be from 1 to 8 characters in length.</td>
</tr>
</tbody>
</table>

---

Gateway RPC Interface  C-3
Sync level for this conversation. This value must be either '0', '1', or '2'.

Sync level 0 indicates that the remote transaction program has no synchronization capabilities.

Sync level 1 indicates that the remote transaction program is capable of responding to CONFIRM requests and is used to ensure data integrity when the remote transaction program is making updates to a database on the remote system.

Sync level 2 indicates that the remote transaction program is capable of responding to SYNC requests and is used to implement two-phase commit. Only one sync level 1 or 2 conversation can be active at any time from a single gateway session.

Note that not all platforms support sync level 2 with the gateway. Refer to the Oracle Procedural Gateway for APPC Installation and Configuration Guide for your platform for information on whether or not your platform supports sync level 2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNLEVEL</td>
<td>IN</td>
<td>CHAR(1)</td>
<td>Sync level for this conversation. This value must be either '0', '1', or '2'. Sync level 0 indicates that the remote transaction program has no synchronization capabilities. Sync level 1 indicates that the remote transaction program is capable of responding to CONFIRM requests and is used to ensure data integrity when the remote transaction program is making updates to a database on the remote system. Sync level 2 indicates that the remote transaction program is capable of responding to SYNC requests and is used to implement two-phase commit. Only one sync level 1 or 2 conversation can be active at any time from a single gateway session. Note that not all platforms support sync level 2 with the gateway. Refer to the Oracle Procedural Gateway for APPC Installation and Configuration Guide for your platform for information on whether or not your platform supports sync level 2.</td>
</tr>
</tbody>
</table>
The following table lists the types, datatypes and descriptions of additional parameters which are passed only to PGAINIT_SEC:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERID</td>
<td>IN</td>
<td>VARCHAR2(8)</td>
<td>Conversation security user ID to be passed to the target OLTP. The value must be from 1 to 8 characters in length.</td>
</tr>
<tr>
<td>PASSWORD</td>
<td>IN</td>
<td>VARCHAR2(8)</td>
<td>Conversation security password to be passed to the target OLTP. The value must be from 1 to 8 characters in length.</td>
</tr>
</tbody>
</table>

There is an interrelationship between PROFNAME and LUNAME/TPNAME/MODENAME. If PROFNAME is set to blanks or a null value, the LUNAME, TPNAME, and MODENAME parameters are all required to be non-blank values. If they are not all set to non-blank values, an exception is generated. However, if PROFNAME is set to a valid Side Information Profile name, the LUNAME, TPNAME, and MODENAME parameters can be null or blank, because the Side Information profile specifies all the information necessary to establish the conversation. In this case, any non-blank, non-null values specified for LUNAME, TPNAME, or MODENAME override values set in the Side Information profile.

**PGAXFER**

PGAXFER is called to transfer data to and from a remote transaction program on an APPC conversation initialized by PGAINIT. The function sends and/or receives data items based on the calling parameters. The following table list the types, datatypes and descriptions of parameters that are passed to PGAXFER:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONVID</td>
<td>IN</td>
<td>RAW(12)</td>
<td>Conversation identifier returned by the PGAINIT function to be used to identify the conversation.</td>
</tr>
</tbody>
</table>
SENDBUF IN RAW(32763) Buffer containing all the data items to be sent to the remote transaction program. The data items are sent as is, with no changes. Data items must appear in the buffer in the exact order in which the remote transaction program expects to receive them. The total size of all the data items cannot exceed the maximum size for a single APPC send, which is 32,763 bytes for a mapped APPC conversation.

SENBUFL IN BINARY_INTEGER Total length of the data items contained in SENDBUF. The range is 0-32,763 bytes. A value of '0' is used when there are no data items to send.

SENDLNS IN RAW(1024) Buffer containing an array of up to 256 4-byte integer values. The first integer value specifies the number of data items contained in the send buffer (SENDBUF). Following that data item count is a series of integer values specifying the lengths of the data items. There must be an exact match between the data item count and the number of data item length values. Up to 255 data items can be described by this array. The sum of all the data item lengths cannot exceed the total length in SENBUFL.

RECVBUF OUT RAW(32763) Buffer to contain all the data items received from the remote transaction program. The data items are stored in this buffer in the exact order in which the remote transaction program sends them. The total size of all the data items cannot exceed the maximum size for a single APPC receive, which is 32,763 bytes for a mapped APPC conversation.
RECVBUFL  IN  BINARY_INTEGER  Total length of the receive buffer. The range is 0-32,763 bytes. A value of '0' is used when there are no data items to receive.

RECVLNS  INOUT  RAW(1024)  Buffer containing an array of up to 256 4-byte integer values. The first integer value specifies the number of data items to be received into the receive buffer (RECVBUF). Following the data item count is a series of integer values specifying the maximum lengths of the data items to be received. On output, these values are replaced with the actual lengths of the data items received. There must be an exact match between the data item count and the number of data item length values. Up to 255 data items can be described by this array. The sum of all the data item lengths cannot exceed the total length of the receive buffer (RECVBUFL).
When PGAXFER is called, either or both of SENDBUFL and RECVBUFL must be nonzero; in other words, at least one data item must be sent to or received from the remote transaction program. If PGAXFER is called with no data items to send or receive, it generates an exception.

---

**Note:** On each PGAXFER call, all send processing occurs first, followed by all receive processing. If a transaction operates in a manner that requires multiple sets of send and receives, then PGAXFER can be called more than once to accommodate the transaction. If more than 32,763 bytes of data are to be sent or received, multiple calls to PGAXFER must be made.

---

**PGATERM**

PGATERM is called to terminate an APPC conversation that was initiated by a previous call to PGAINIT. Upon successful completion of this function, the conversation is deallocated and all storage associated with it is freed. The following table presents the types, datatypes and descriptions of PGATERM parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONVID</td>
<td>IN</td>
<td>RAW(12)</td>
<td>Conversation identifier returned by the PGAINIT function to be used to identify the conversation.</td>
</tr>
<tr>
<td>TERMTYPE</td>
<td>IN</td>
<td>CHAR(1)</td>
<td>Type of termination to be performed. '0' indicates normal completion and '1' indicates abnormal termination, which is only requested if there is an error.</td>
</tr>
</tbody>
</table>

---

**PGATCTL**

PGATCTL is called to manipulate the PGA trace control flags. These are the same flags set at gateway initialization time by the SET TRACE_LEVEL PGA parameter. Using PGATCTL, the trace level can be changed dynamically from within a PL/SQL stored procedure. This facility is useful when debugging a new PL/SQL application.

---

C-8 Oracle Procedural Gateway for APPC User’s Guide
The following table presents the types, datatypes and descriptions of parameters in PGATCTL:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONVID</td>
<td>IN</td>
<td>RAW(12)</td>
<td>Conversation identifier returned by the PGAINIT function to be used to identify the conversation. If this parameter is set to binary zeroes, the new trace settings affect only conversations started after this call to PGATCTL. If this parameter is set to binary ones, the new trace settings affect all currently active conversations. Otherwise, the new trace settings affect only the specified conversation.</td>
</tr>
<tr>
<td>TRFUNC</td>
<td>IN</td>
<td>CHAR(1)</td>
<td>Trace control function to be performed. The valid values are: ‘S’ - set trace flags to the exact value specified by the TRFLAGS parameter. ‘E’ - enable the trace flags specified by the TRFLAGS parameter, without changing any other flags. ‘D’ - disable the trace flags specified by the TRFLAGS parameter, without changing any other flags.</td>
</tr>
<tr>
<td>TRFLAGS</td>
<td>IN</td>
<td>BINARY_INTEGER</td>
<td>Trace flags. For valid values, refer to the discussion of the SET TRACE_LEVEL parameter in Appendix A of the Oracle Procedural Gateway for APPC Installation and Configuration Guide for your platform.</td>
</tr>
</tbody>
</table>

**PGATRAC**

This function is called to write a line of user data into the PGA trace file. Using PGATRAC, the flow within a PL/SQL procedure can be traced, along with the events traced based on the SET TRACE_LEVEL settings. This is a useful debugging tool when developing a new PL/SQL application.

The following table presents the type, datatype and description of the PGATRAC parameter:
## Calling Gateway Functions to Execute Transaction Programs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRDATA</td>
<td>IN</td>
<td>VARCHAR2(120)</td>
<td>Line of user data to be written into the gateway trace file. The contents must be printable characters.</td>
</tr>
</tbody>
</table>
The Oracle Procedural Gateway for APPC requires the use of the RAW datatype to transfer data to and from PL/SQL without any alteration by Oracle Net. This is necessary because only the PL/SQL applications have information about the format of the data being sent to and received from the remote transaction programs. Oracle Net only has information about the systems where the PL/SQL application and the gateway server are running. If Oracle Net is allowed to perform translation on the data flowing between PL/SQL and the gateway, the data can end up in the wrong format.

This appendix contains the following sections:

- UTL_RAW Functions on page D-2
- UTL_PG Functions on page D-17
- NUMBER_TO_RAW and RAW_TO_NUMBER Argument Values on page D-37
The UTL_RAW package is an extension to PL/SQL that provides a full complement of RAW data manipulation functions. Using these functions, data sent to remote transaction programs can be converted into the correct format by the PL/SQL application, and data received from remote transaction programs can be converted back into Oracle formats.

All of the functions listed in this section are called in the standard PL/SQL manner, which is `package_name.function_name(arguments)`. In the case of the UTL_RAW routines, this is `UTL_RAW.function_name(arguments)`.

For each function listed below, the function name, arguments and their datatypes, and the return value datatype are provided. Unless otherwise specified, the parameters are IN, not OUT, parameters.

**BIT_AND**

BIT_AND performs a bitwise logical AND operation of the values in `r1` and `r2` and returns the resulting value.

If `r1` and `r2` have different lengths, then the AND operation is terminated after the last byte of the shorter of the two RAW values. The unprocessed portion of the longer RAW value is appended to the partial result to produce the final result returned. The result length equals the longer of the two input RAW values.

**Syntax**

```plaintext
function BIT_AND (r1 IN RAW, r2 IN RAW) RETURN RAW;
```

The following table describes the syntax:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>r1</code></td>
<td>is a RAW value to be combined with <code>r2</code>.</td>
</tr>
<tr>
<td><code>r2</code></td>
<td>is a RAW value to be combined with <code>r1</code>.</td>
</tr>
</tbody>
</table>

**Defaults**

None

**Return Value**

A RAW value which is the bitwise logical AND of `r1` and `r2`. Or a null value if either `r1` or `r2` is null.
Errors
None

BIT_COMPLEMENT

BIT_COMPLEMENT performs a bitwise logical COMPLEMENT operation of the value \( r \) and returns the resulting RAW value. The length of the result equals the length of the input RAW value \( r \).

Syntax
function BIT_COMPLEMENT (r IN RAW) RETURN RAW;

where:
\( r \) is the RAW value on which to perform the COMPLEMENT operation.

Defaults
None

Return Value
A RAW value which is the bitwise logical COMPLEMENT of \( r \). Or a null value if the input value \( r \) is null.

Errors
None

BIT_OR

BIT_OR performs a bitwise logical OR operation of the values in \( r1 \) and \( r2 \) and returns the resulting value.

If \( r1 \) and \( r2 \) have different lengths, then the OR operation is terminated after the last byte of the shorter of the two RAW values. The unprocessed portion of the longer RAW value is appended to the partial result to produce the final result returned. The resulting length equals the longer of the two input values, \( r1 \) and \( r2 \).

Syntax
function BIT_OR (r1 IN RAW, r2 IN RAW) RETURN RAW;
The following table describes the syntax:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>is a RAW value to OR with r2.</td>
</tr>
<tr>
<td>r2</td>
<td>is a RAW value to OR with r1.</td>
</tr>
</tbody>
</table>

**Defaults**

None

**Return Value**

A RAW value which is the bitwise logical OR of r1 and r2. Or a null value if both r1 and r2 is null.

**Errors**

None

**BIT_XOR**

BIT_XOR performs a bitwise logical EXCLUSIVE OR operation of the values in r1 and r2 and returns the resulting value.

If r1 and r2 have different lengths, then the EXCLUSIVE OR operation is terminated after the last byte of the shorter of the two RAW values. The unprocessed portion of the longer RAW value is appended to the partial result to produce the final result returned. The result length equals the longer of the two input RAW values.

**Syntax**

function BIT_XOR (r1 IN RAW, r2 IN RAW) RETURN RAW

The following table describes the syntax:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>is the RAW value to XOR with r2.</td>
</tr>
<tr>
<td>r2</td>
<td>is the RAW value to XOR with r1.</td>
</tr>
</tbody>
</table>
UTL_RAW Functions

Defaults
None

Return Value
A RAW value which is the bitwise logical XOR of r1 and r2. Or a null value if r1 and r2 have identical values.

Errors
None

CAST_TO_RAW
CAST_TO_RAW converts a VARCHAR2 c into a RAW with the same number of bytes.
The input is treated as if it is composed of single 8-bit bytes, not characters. Multibyte character boundaries are ignored. The data is not modified in any way, it is just changed to a RAW datatype.

Syntax
function CAST_TO_RAW (c IN VARCHAR2) RETURN RAW;

where:

  c           is a VARCHAR2 value to be changed to a RAW value.

Defaults
None

Return Value
A RAW value having the same data and byte length as the input VARCHAR2 value. Or a null value if c is null.

Errors
None
CAST_TO_VARCHAR2

CAST_TO_VARCHAR2 converts a RAW \textit{r} into a VARCHAR2 with the same number of data bytes.

The result is treated as if it is composed of single 8-bit bytes, not characters. Multibyte character boundaries are ignored. The data is not modified in any way, it is just changed to a VARCHAR2 datatype.

\textbf{Syntax}

function CAST\_TO\_VARCHAR2 (r IN RAW) RETURN VARCHAR2;

where:

\textit{r} is a RAW value to be changed to a VARCHAR2 value.

\textbf{Defaults}

None

\textbf{Return Value}

A VARCHAR2 value having the same data as the RAW input value. Or a null value if \textit{r} is null.

\textbf{Errors}

None

COMPARE

COMPARE compares RAW \textit{r1} with RAW \textit{r2}.

If they are identical, COMPARE returns zero. Otherwise, COMPARE returns the position of the first byte that does not match. If the input values are of different length, the shorter RAW value is padded on the right with the byte specified by \textit{pad}.

\textbf{Syntax}

function COMPARE (r1 IN RAW, r2 IN RAW, pad RAW DEFAULT NULL) RETURN NUMBER;
The following table describes the syntax:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>Is the first RAW value to be compared. This may be null and/or have a length of 0.</td>
</tr>
<tr>
<td>r2</td>
<td>Is the second RAW value to be compared. This might be null and/or have a length of 0.</td>
</tr>
<tr>
<td>pad</td>
<td>Is a 1 byte value used to pad the shorter RAW value.</td>
</tr>
</tbody>
</table>

**Defaults**

pad  optional and defaults to x'00'.

**Return Value**

A value of 0 if the strings are null or identical. Or the position, numbered from 1, of the first mismatched byte.

**Errors**

None

**CONCAT**

CONCAT concatenates a set of up to 12 RAW values (r1 - r12) into a single RAW and returns it.

Input RAW values are appended together in the resulting RAW, left to right, in the order they appear in the parameter list. Input values need not be contiguous. Null input RAW values are skipped and the concatenation continues with the next non-null input RAW value. If the sum of the lengths of the input RAWs exceeds the maximum allowable length for a RAW (32767), an error is returned.

**Syntax**

```sql
function CONCAT (r1 IN RAW DEFAULT NULL,
                 r2 RAW DEFAULT NULL,
                 r3 RAW DEFAULT NULL,
                 r4 RAW DEFAULT NULL,
                 r5 RAW DEFAULT NULL,
                 r6 RAW DEFAULT NULL,
                 r7 RAW DEFAULT NULL,
                 r8 RAW DEFAULT NULL,
                 r9 RAW DEFAULT NULL,
                 r10 RAW DEFAULT NULL,
                 r11 RAW DEFAULT NULL,
                 r12 RAW DEFAULT NULL)
```
UTL_RAW Functions

r8 RAW DEFAULT NULL,
r9 RAW DEFAULT NULL,
r10 RAW DEFAULT NULL,
r11 RAW DEFAULT NULL,
r12 RAW DEFAULT NULL) RETURN RAW;

where:

r1 – r12 are the RAW items to be concatenated.

Defaults
None

Return Value
A RAW value with the concatenated items.

Errors
A VALUE_ERROR exception is raised if the sum of the input value lengths exceeds 32767.

CONVERT

CONVERT converts RAW r from character set from_charset to character set to_charset and returns the resulting RAW value.

Both from_charset and to_charset must specify supported character sets defined to the Oracle server.

Syntax

function CONVERT (r IN RAW,
to_charset IN VARCHAR2,
from_charset IN VARCHAR2) RETURN RAW;

The following table describes the syntax:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>Is the RAW byte-string to be converted.</td>
</tr>
<tr>
<td>to_charset</td>
<td>Is the NLS character set to convert r to.</td>
</tr>
<tr>
<td>from_charset</td>
<td>Is the NLS character set that r is currently using.</td>
</tr>
</tbody>
</table>
Defaults
None

Return Value
A RAW string of bytes converted according to the specified character set.

Errors
VALUE_ERROR exception raised when:
- $r$ is null, or 0 in length
- from_charset or to_charset missing, null, or 0 in length
- from_charset or to_charset names invalid or unsupported

COPIES

COPIES returns $n$ copies of RAW $r$ concatenated together.

Syntax
function COPIES (r IN RAW,
   n IN NUMBER) RETURN RAW;

The following table describes the syntax:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r$</td>
<td>Is the RAW value to be copied.</td>
</tr>
<tr>
<td>$n$</td>
<td>Is the number of times to copy the RAW value. This must be a positive value.</td>
</tr>
</tbody>
</table>

Defaults
None

Return Value
The RAW value copied $n$ times.

Errors
A VALUE_ERROR exception is raised if:
- $r$ is null or has a length of 0
n < 1
the length of the result exceeds the maximum length of a RAW value (32767)

LENGTH

LENGTH returns the length in bytes of RAW r.

Syntax

function LENGTH (r IN RAW) RETURN NUMBER;

where:

r is the byte stream to be measured.

Defaults

None

Return Value

The current length of the RAW input value.

Errors

None

OVERLAY

OVERLAY replaces the specified portion of RAW target with RAW overlay, beginning at byte position pos of target, and proceeding for len bytes.

If overlay has fewer than len bytes, then overlay is padded to len bytes using the byte specified by pad. If overlay has more than len bytes, then the extra bytes in overlay are ignored. If len bytes beginning at position pos of target exceeds the length of target, target is extended to contain the entire length of overlay. If pos exceeds the length of target, target is padded with pad bytes to position pos and then target is further extended with overlay bytes.

Syntax

function OVERLAY (overlay IN RAW,
target IN RAW,
pos IN BINARY_INTEGER DEFAULT 1,
len IN BINARY_INTEGER DEFAULT NULL,
   pad IN RAW DEFAULT NULL) RETURN RAW;

The following table describes the syntax:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>overlay</td>
<td>Is a byte-string used to overlay the target. Bytes are always selected from the overlay RAW beginning with the leftmost byte.</td>
</tr>
<tr>
<td>target</td>
<td>Is the byte-string to be overlayed.</td>
</tr>
<tr>
<td>pos</td>
<td>Is the position within the target RAW, numbered from 1, at which to begin overlaying. This value must be greater than zero. This parameter is optional.</td>
</tr>
<tr>
<td>len</td>
<td>Is the number of bytes to overlay. This must be greater than or equal to zero. This parameter is optional.</td>
</tr>
<tr>
<td>pad</td>
<td>Is a single byte value used to pad when len exceeds overlay length or pos exceeds target length. This parameter is optional.</td>
</tr>
</tbody>
</table>

**Defaults**

The following table describes the defaults:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
<td>Defaults to 1.</td>
</tr>
<tr>
<td>len</td>
<td>Defaults to length of overlay.</td>
</tr>
<tr>
<td>pad</td>
<td>Defaults to x'00'.</td>
</tr>
</tbody>
</table>

**Return Value**

The target byte-string overlayed as specified.

**Errors**

A VALUE_ERROR exception is raised if:

- overlay is null or has a length of 0
- target is undefined
■ the length of \texttt{target} exceeds the maximum length of a RAW value (32767)
■ \texttt{len} is less than 0
■ \texttt{pos} is less than 1

\textbf{REVERSE}

Reverse the byte sequence in RAW \texttt{r} from end-to-end. For example, \texttt{x'0102F3'} would be reversed into \texttt{x'F30201'} and \texttt{'xyz'} would be reversed to \texttt{'zyx'}. The result length is the same as the input RAW length.

\textbf{Syntax}

\begin{verbatim}
function REVERSE (r IN RAW) RETURN RAW;
end function REVERSE;
\end{verbatim}

where:

\begin{itemize}
\item \texttt{r} is the RAW value to reverse.
\end{itemize}

\textbf{Defaults}

None

\textbf{Return Value}

A RAW value containing the reverse of the input RAW value.

\textbf{Errors}

A VALUE\_ERROR exception is raised if \texttt{r} is null or has a length of 0.

\textbf{SUBSTR}

SUBSTR extracts a portion of RAW \texttt{r}, starting at byte position \texttt{pos} and including \texttt{len} bytes.

If \texttt{pos} is positive, SUBSTR counts from the beginning of \texttt{r} to find the first byte. If \texttt{pos} is negative, SUBSTR counts backwards from the end of \texttt{r}. If \texttt{len} is not specified, SUBSTR returns all bytes to the end of \texttt{r}.

\textbf{Syntax}

\begin{verbatim}
function SUBSTR (r IN RAW,
    pos IN BINARY\_INTEGER,
    len BINARY\_INTEGER DEFAULT NULL) RETURN RAW;
end function SUBSTR;
\end{verbatim}
The following table describes the syntax:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>Is the RAW byte-string from which a portion is to be extracted.</td>
</tr>
<tr>
<td>pos</td>
<td>Is the byte position from which to start extraction. This value cannot be zero. If this value is negative, SUBSTR counts backwards from the end of r.</td>
</tr>
<tr>
<td>len</td>
<td>Is the number of bytes from pos to extract from r. This value must be greater than zero. This parameter is optional.</td>
</tr>
</tbody>
</table>

**Defaults**

Defaults to the length of position pos to the end of r.

**Return Value**

The portion of r beginning at pos for len bytes. Or a null value if r is null.

**Errors**

A VALUE_ERROR exception is raised if:

- pos is 0
- len is less than 0

**TRANSLATE**

TRANSLATE returns the RAW r after changing the bytes in from_set according to bytes in to_set.

Successive bytes in r are looked up in from_set and if found, the byte at the same offset in to_set is copied to the result or omitted from the result if the offset exceeds the length of to_set. Bytes that appear in r but not in from_set are copied to the result. Only the first (leftmost) occurrence of a byte in from_set is used and subsequent duplicate occurrences are ignored.

If from_set contains more bytes than to_set, the extra bytes at the end of from_set have no corresponding bytes in to_set. Any bytes in r matching such uncorresponded from_set bytes are omitted from the resulting RAW value.
TRANSLATE differs from TRANSLITERATE in the following ways:

- translation RAWs have no defaults
- r bytes undefined in the to_set translation RAW are omitted
- resulting RAW value can be shorter than the input RAW value

**Syntax**

```
function TRANSLATE (r IN RAW,
                   from_set IN RAW,
                   to_set IN RAW) RETURN RAW;
```

The following table describes the syntax:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>Is the RAW source byte-string to be changed.</td>
</tr>
<tr>
<td>from_set</td>
<td>Is the RAW byte-codes to be matched, if present in r.</td>
</tr>
<tr>
<td>to_set</td>
<td>Is the RAW byte-codes to which corresponding from_set bytes are changed.</td>
</tr>
</tbody>
</table>

**Defaults**

None

**Return Value**

A RAW translated byte-string.

**Errors**

A VALUE_ERROR exception is raised if:

- r is null or has a length of 0
- from_set is null or has a length of 0
- to_set is null or has a length of 0

**TRANSLITERATE**

TRANSLITERATE returns the RAW r after replacing all occurrences of any bytes in from_set with the corresponding bytes in to_set.
Successive bytes in \( r \) are looked up in \( \text{from}_\text{set} \) and, if not found, are copied unaltered to the resulting \( \text{RAW} \) value. If found, they are replaced in the resulting \( \text{RAW} \) value by either the byte at the same offset in \( \text{to}_\text{set} \), or the \( \text{pad} \) byte when the offset exceeds the \( \text{to}_\text{set} \) length. Bytes found in \( r \) but not found in \( \text{from}_\text{set} \) are copied to the result. Only the first (leftmost) occurrence of a byte in \( \text{from}_\text{set} \) is used; subsequent duplicate occurrences are ignored. The result of \( \text{TRANSLITERATE} \) is always the same length as \( \text{RAW} \) \( r \).

If \( \text{to}_\text{set} \) is shorter than \( \text{from}_\text{set} \), then the \( \text{pad} \) byte is placed in the resulting \( \text{RAW} \) value when a selected \( \text{from}_\text{set} \) byte has no corresponding byte in \( \text{to}_\text{set} \).

\( \text{TRANSLITERATE} \) differs from \( \text{TRANSLATE} \) in the following ways:

- \( r \) bytes undefined in \( \text{to}_\text{set} \) are padded
- the resulting \( \text{RAW} \) value is always the same length as the input \( \text{RAW} \) value

**Syntax**

\[
\text{function TRANSLITERATE (r IN RAW,}
\text{ to\_set IN RAW DEFAULT NULL,}
\text{ from\_set IN RAW DEFAULT NULL,}
\text{ pad IN RAW DEFAULT NULL) RETURN RAW;}
\]

The following table describes the syntax:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r )</td>
<td>Is the ( \text{RAW} ) source byte-string to be changed.</td>
</tr>
<tr>
<td>( \text{to}_\text{set} )</td>
<td>Is the ( \text{RAW} ) byte-codes to which corresponding ( \text{from}_\text{set} ) bytes are changed. This value can be of any valid ( \text{RAW} ) length.</td>
</tr>
<tr>
<td>( \text{from}_\text{set} )</td>
<td>Is the ( \text{RAW} ) byte-codes to be matched, if present in ( r ). This value can be of any valid ( \text{RAW} ) length.</td>
</tr>
<tr>
<td>( \text{pad} )</td>
<td>Is a 1 byte value used when ( \text{to}<em>\text{set} ) is shorter than ( \text{from}</em>\text{set} ).</td>
</tr>
</tbody>
</table>

**Defaults**

The following table describes the defaults:
Return Value
A RAW transliterated byte-string.

Errors
A VALUE_ERROR exception is raised if:
- \( r \) is null or has a length of 0

XRANGE
XRANGE returns a RAW containing all valid 1-byte encodings in succession beginning with the value \( \text{start}_\text{byte} \) and ending with the value \( \text{end}_\text{byte} \). If \( \text{start}_\text{byte} \) is greater than \( \text{end}_\text{byte} \), the succession of result bytes begin with \( \text{start}_\text{byte} \), wrap from \('ff'\) to \('00'\), and end at \( \text{end}_\text{byte} \). If specified, \( \text{start}_\text{byte} \) and \( \text{end}_\text{byte} \) must be single-byte RAW values.

Syntax
function XRANGE (start_byte IN RAW DEFAULT NULL,
                 end_byte IN RAW DEFAULT NULL) RETURN RAW;

The following table describes the syntax:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_byte</td>
<td>Is the 1-byte beginning byte-code value of the resulting sequence.</td>
</tr>
<tr>
<td>end_byte</td>
<td>Is the 1-byte ending byte-code value of the resulting sequence.</td>
</tr>
</tbody>
</table>

Defaults
The following table describes the defaults:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>to_set</td>
<td>Defaults to a null value, and effectively extended with pad to the length of from_set as necessary.</td>
</tr>
<tr>
<td>from_set</td>
<td>Defaults to ('00' through ('ff').</td>
</tr>
<tr>
<td>pad</td>
<td>Defaults to ('00').</td>
</tr>
</tbody>
</table>
Return Value
A RAW value containing a succession of 1-byte hexadecimal encodings.

Errors
None

UTL_PG Functions
The UTL_PG package is an extension to PL/SQL that provides a full set of functions for converting COBOL number formats into Oracle numbers and Oracle numbers into COBOL number formats.

UTL_PG conversion format RAWs are not portable in this release. Additionally, generation of conversion format RAWs on one system and transfer to another system is not supported.

The functions listed in this section are called in the standard PL/SQL manner:

```
package_name.function_name(arguments)
```

Specifically for UTL_PG routines, this is:

```
UTL_PG.function_name(arguments)
```

For each function listed below, the function name, arguments and their datatypes, and the return value datatype are provided. Unless otherwise specified, the parameters are IN, not OUT, parameters.

Common Parameters
The following UTL_PG functions require several similar parameters:

- RAW_TO_NUMBER
- MAKE_NUMBER_TO_RAW_FORMAT
- MAKE_RAW_TO_NUMBER_FORMAT
- NUMBER_TO_RAW
These similar parameters are documented in detail here and then referenced only by name for each UTL_PG function.

**Input Parameters**
The following table describes the input parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>Is the compiler datatype mask. This is the datatype to be converted, specified in the source language of the named compiler (compname). This implies the internal format of the data as encoded according to the compiler and host platform.</td>
</tr>
<tr>
<td>maskopts</td>
<td>The compiler datatype mask options or NULL. These are additional options associated with the mask, as allowed or required, and are specified in the source language of compname. These can further qualify the type of conversion as necessary.</td>
</tr>
<tr>
<td>envrnmnt</td>
<td>The compiler environment clause or NULL. These are additional options associated with the environment in which the remote data resides, as allowed or required, and is specified in the source language of compname. This parameter typically supplies aspects of data conversion dictated by customer standards, such as decimal point or currency symbols if applicable.</td>
</tr>
<tr>
<td>compname</td>
<td>The compiler name. The only supported value is IBMVSCOBOLII.</td>
</tr>
<tr>
<td>compopts</td>
<td>The compiler options or NULL.</td>
</tr>
<tr>
<td>nlslang</td>
<td>The zoned decimal code page specified in Oracle NLS format, language_territory.charset. This defaults to AMERICAN_AMERICA.WE8EBCDIC37C.</td>
</tr>
<tr>
<td>wind</td>
<td>The warning indicator. A Boolean indicator which controls whether conversion warning messages are to be returned in the wmsgblk OUT parameter.</td>
</tr>
</tbody>
</table>
The UTL_PG and UTL_RAW Interfaces

Output Parameters
The following table describes the output parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wmsgbsiz</td>
<td>The warning message block declared size in bytes. It is a BINARY_INTEGER set to the byte length of wmsgblk. The warning message block must be at least 512 and not more than 8192 bytes in length. When declaring wmsgblk, plan on approximately 512 bytes for each warning returned, depending on the nature of the requested conversion.</td>
</tr>
</tbody>
</table>

RAW_TO_NUMBER
RAW_TO_NUMBER converts a RAW byte-string r from the remote host internal format specified by mask, maskopts, envrmnt, compname, compopts, and nlslang into an Oracle number.

Warnings are issued, if enabled, when the conversion specified conflicts with the conversion implied by the data or when conflicting format specifications are supplied.
For detailed information about the mask, maskopts, envrnmnt, compname, and compopts arguments, refer to "NUMBER_TO_RAW and RAW_TO_NUMBER Argument Values" on page D-37.

Syntax

function RAW_TO_NUMBER (r IN RAW,
mask IN VARCHAR2,
maskopts IN VARCHAR2,
envrnmnt IN VARCHAR2,
compname IN VARCHAR2,
compopts IN VARCHAR2,
nls-lang IN VARCHAR2,
wind IN BOOLEAN,
wmsgbsiz IN BINARY_INTEGER,
wmsgblk OUT RAW) RETURN NUMBER;

The following table describes the syntax:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>Is the remote host data to be converted.</td>
</tr>
<tr>
<td>mask</td>
<td>Is the compiler datatype mask.</td>
</tr>
<tr>
<td>maskopts</td>
<td>Are the compiler datatype mask options or NULL.</td>
</tr>
<tr>
<td>envrnmnt</td>
<td>Is the compiler environment clause or NULL.</td>
</tr>
<tr>
<td>compname</td>
<td>Is the compiler name.</td>
</tr>
<tr>
<td>compopts</td>
<td>Are the compiler options or NULL.</td>
</tr>
<tr>
<td>nls-lang</td>
<td>Is the zoned decimal code page in Oracle NLS format.</td>
</tr>
<tr>
<td>wind</td>
<td>Is a warning indicator.</td>
</tr>
<tr>
<td>wmsgbsiz</td>
<td>Is the warning message block size in bytes.</td>
</tr>
<tr>
<td>wmsgblk</td>
<td>Is the warning message block. This is an OUT parameter.</td>
</tr>
</tbody>
</table>

Defaults and Optional Parameters

The following table describes the default and optional parameters:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maskopts</td>
<td>null allowed, no default value</td>
</tr>
</tbody>
</table>
Return Value
An Oracle number corresponding in value to \( r \).

Errors
ORA-8401 invalid compiler name
ORA-8414 error encountered
ORA-8429 raw data has invalid digit display type data
ORA-8430 raw data missing leading sign
ORA-8431 raw data missing zero as defined in picture
ORA-8433 invalid picture type in convert raw to number
ORA-8434 raw data has invalid trailing sign
ORA-8435 picture mask missing leading sign
ORA-8436 raw data has invalid sign digit
ORA-8437 invalid picture type in picture mask
ORA-8441 closed parenthesis is missing in picture mask
ORA-8443 syntax error in BLANK WHEN ZERO clause in mask options
ORA-8444 syntax error in JUSTIFIED clause in mask options
ORA-8445 syntax error in SIGN clause in mask options
ORA-8446 syntax error in SYCHRONIZED clause in mask options
ORA-8447 syntax error in USAGE clause in mask options
ORA-8448 syntax error in DECIMAL-POINT clause in mask options
ORA-8449 invalid numeric symbol found in picture mask
ORA-8450 invalid specification of CR in picture mask
ORA-8451 invalid specification of DB in picture mask
ORA-8452 specification of E in picture mask unsupported

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>envrmnt</td>
<td>null allowed, no default value</td>
</tr>
<tr>
<td>compopts</td>
<td>null allowed, no default value</td>
</tr>
</tbody>
</table>
ORA-8453 more than one V symbol specified in picture mask
ORA-8454 more than one S symbol specified in picture mask
ORA-8455 syntax error in CURRENCY SIGN environment clause
ORA-8456 no sign in picture mask but SIGN clause in mask options
ORA-8457 syntax error in SEPARATE CHARACTER option of SIGN clause
ORA-8458 syntax error in environment clause
ORA-8460 invalid environment clause in environment parameter
ORA-8462 raw buffer contains invalid decimal data
ORA-8463 overflow converting decimal number to Oracle number
ORA-8464 input raw decimal data contains more than 42 digits
ORA-8466 raw buffer length len too short for spec
ORA-8468 mask option option is not supported

Warnings
The following warnings can be issued, if warnings are enabled:
ORA-8498 picture mask overrides mask option USAGE... to DISPLAY
ORA-8499 picture mask options ignored by UTL_PG
NUMBER_TO_RAW

NUMBER_TO_RAW converts an Oracle number \( n \) of declared precision and scale into a RAW byte-string in the remote host internal format specified by \( \text{mask} \), \( \text{maskopts} \), \( \text{envrnmnt} \), \( \text{compname} \), \( \text{compopts} \), and \( \text{nlslang} \).

Warnings are issued, if enabled, when the conversion specified conflicts with the conversion implied by the data or when conflicting format specifications are supplied.

For detailed information about the \( \text{mask} \), \( \text{maskopts} \), \( \text{envrnmnt} \), \( \text{compname} \), and \( \text{compopts} \) arguments, refer to "NUMBER_TO_RAW and RAW_TO_NUMBER Argument Values" on page D-37.

Syntax

```sql
function NUMBER_TO_RAW (n IN NUMBER,
  mask IN VARCHAR2,
  maskopts IN VARCHAR2,
  envrnmnt IN VARCHAR2,
  compname IN VARCHAR2,
  compopts IN VARCHAR2,
  nlslang IN VARCHAR2,
  wind IN BOOLEAN,
  wmsgbsiz IN BINARY_INTEGER,
  wmsgblk OUT RAW) RETURN RAW;
```

The following table describes the syntax:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Is the Oracle number to be converted.</td>
</tr>
<tr>
<td>mask</td>
<td>Is the compiler datatype mask.</td>
</tr>
<tr>
<td>maskopts</td>
<td>Are the compiler datatype mask options or NULL.</td>
</tr>
<tr>
<td>envrnmnt</td>
<td>Is the compiler environment clause or NULL.</td>
</tr>
<tr>
<td>compname</td>
<td>Is the compiler name.</td>
</tr>
<tr>
<td>compopts</td>
<td>Are the compiler options or NULL.</td>
</tr>
<tr>
<td>nlslang</td>
<td>Is the zoned decimal code page in Oracle NLS format.</td>
</tr>
<tr>
<td>wind</td>
<td>Is a warning indicator.</td>
</tr>
<tr>
<td>wmsgbsiz</td>
<td>Is the warning message block size in bytes.</td>
</tr>
</tbody>
</table>
The following table describes the defaults and optional parameters:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wmsgblk</td>
<td>Is the warning message block. This is an OUT parameter.</td>
</tr>
</tbody>
</table>

**Return Value**

A RAW value corresponding in value to n.

**Errors**

ORA-8401 invalid compiler name
ORA-8414 error encountered
ORA-8437 invalid picture type in picture mask
ORA-8441 closed parenthesis is missing in picture mask
ORA-8443 syntax error in BLANK WHEN ZERO clause in mask options
ORA-8444 syntax error in JUSTIFIED clause in mask options
ORA-8445 syntax error in SIGN clause in mask options
ORA-8446 syntax error in SYCHRONIZED clause in mask options
ORA-8447 syntax error in USAGE clause in mask options
ORA-8448 syntax error in DECIMAL-POINT clause in mask options
ORA-8449 invalid numeric symbol found in picture mask
ORA-8450 invalid specification of CR in picture mask
ORA-8451 invalid specification of DB in picture mask
ORA-8452 specification of E in picture mask unsupported
ORA-8453 more than one V symbol specified in picture mask
ORA-8454 more than one S symbol specified in picture mask
ORA-8455 syntax error in CURRENCY SIGN environment clause
ORA-8456 no sign in picture mask but SIGN clause in mask options
ORA-8457 syntax error in SEPARATE CHARACTER option of SIGN clause
ORA-8460 invalid environment clause in environment parameter
ORA-8466 raw buffer length \textit{len} to short for \textit{spec}
ORA-8467 error encountered converting Oracle number to \textit{type}
ORA-8468 mask option \textit{option} is not supported

**Warnings**
The following warnings can be issued, if warnings are enabled:
ORA-8498 picture mask overrides mask option USAGE... to DISPLAY
ORA-8499 picture mask options ignored by UTL_PG
MAKE_RAW_TO_NUMBER_FORMAT

MAKE_RAW_TO_NUMBER_FORMAT makes a RAW_TO_NUMBER format conversion specification used to convert a RAW byte-string from the remote host internal format specified by mask, maskopts, envrnmnt, compname, compopts, and nlslang into an Oracle number of comparable precision and scale.

Warnings are issued, if enabled, when the conversion specified conflicts with the conversion implied by the data or when conflicting format specifications are supplied.

This function returns a RAW value containing the conversion format which can be passed to UTL_PG.RAW_TO_NUMBER_FORMAT.

For detailed information about the mask, maskopts, envrnmnt, compname, and compopts arguments, refer to “NUMBER_TO_RAW and RAW_TO_NUMBER Argument Values” on page D-37.

Syntax

function MAKE_RAW_TO_NUMBER_FORMAT (mask IN VARCHAR2,
maskopts IN VARCHAR2,
envrnmnt IN VARCHAR2,
compname IN VARCHAR2,
compopts IN VARCHAR2,
nlslang IN VARCHAR2,
wind IN BOOLEAN,
wmsgbsiz IN BINARY_INTEGER,
wmsgblk OUT RAW) RETURN RAW;

The following table describes the syntax:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>Is the compiler datatype mask.</td>
</tr>
<tr>
<td>maskopts</td>
<td>Are the compiler datatype mask options or NULL.</td>
</tr>
<tr>
<td>envrnmnt</td>
<td>Is the compiler environment clause or NULL.</td>
</tr>
<tr>
<td>compname</td>
<td>Is the compiler name.</td>
</tr>
<tr>
<td>compopts</td>
<td>Are the compiler options or NULL.</td>
</tr>
<tr>
<td>nlslang</td>
<td>Is the zoned decimal code page in Oracle NLS format.</td>
</tr>
<tr>
<td>wind</td>
<td>Is a warning indicator.</td>
</tr>
</tbody>
</table>
The UTL_PG and UTL_RAW Interfaces

Defaults and Optional Parameters
The following table describes the defaults and optional parameters:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wmsgbsiz</td>
<td>Is the warning message block size in bytes.</td>
</tr>
<tr>
<td>wmsgblk</td>
<td>Is the warning message block. This is an OUT parameter.</td>
</tr>
</tbody>
</table>

Return Value
A RAW(2048) format conversion specification for RAW_TO_NUMBER.

Errors
ORA-8401 invalid compiler name
ORA-8414 error encountered
ORA-8433 invalid picture type in convert raw to number
ORA-8437 invalid picture type in picture mask
ORA-8441 closed parenthesis is missing in picture mask
ORA-8443 syntax error in BLANK WHEN ZERO clause in mask options
ORA-8444 syntax error in JUSTIFIED clause in mask options
ORA-8445 syntax error in SIGN clause in mask options
ORA-8446 syntax error in SYCHRONIZED clause in mask options
ORA-8447 syntax error in USAGE clause in mask options
ORA-8448 syntax error in DECIMAL-POINT clause in mask options
ORA-8449 invalid numeric symbol found in picture mask
ORA-8450 invalid specification of CR in picture mask
ORA-8451 invalid specification of DB in picture mask
ORA-8452 specification of E in picture mask unsupported
ORA-8453 more than one V symbol specified in picture mask
ORA-8454 more than one S symbol specified in picture mask
ORA-8455 syntax error in CURRENCY SIGN environment clause
ORA-8456 no sign in picture mask but SIGN clause in mask options
ORA-8457 syntax error in SEPARATE CHARACTER option of SIGN clause
ORA-8458 invalid format parameter
ORA-8459 invalid format parameter length
ORA-8460 invalid environment clause in environment parameter
ORA-8467 error encountered converting Oracle number to type
ORA-8468 mask option option is not supported

Warnings
The following warnings can be issued, if warnings are enabled:
ORA-8498 picture mask overrides mask option USAGE... to DISPLAY
ORA-8499 picture mask options ignored by UTL_PG
MAKE_NUMBER_TO_RAW_FORMAT

MAKE_NUMBER_TO_RAW_FORMAT makes a NUMBER_TO_RAW format conversion specification used to convert an Oracle number of declared precision and scale to a RAW byte-string in the remote host internal format specified by mask, maskopts, envrnmnt, compname, compopts, and nlslang.

Warnings are issued, if enabled, when the conversion specified conflicts with the conversion implied by the data or when conflicting format specifications are supplied.

This function returns a RAW value containing the conversion format which can be passed to UTL_PG.NUMBER_TO_RAW_FORMAT. The implementation length of the result format RAW is 2048 bytes.

For detailed information about the mask, maskopts, envrnmnt, compname, and compopts arguments, refer to "NUMBER_TO_RAW and RAW_TO_NUMBER Argument Values" on page D-37.

Syntax

function MAKE_NUMBER_TO_RAW_FORMAT (mask IN VARCHAR2,
maskopts IN VARCHAR2,
envrnmnt IN VARCHAR2,
compname IN VARCHAR2,
compopts IN VARCHAR2,
nlslang IN VARCHAR2,
wind IN BOOLEAN,
wmsgbsiz IN BINARY_INTEGER,
wmsgblk OUT RAW) RETURN RAW;

The following table describes the syntax:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>is the compiler datatype mask.</td>
</tr>
<tr>
<td>maskopts</td>
<td>are the compiler datatype mask options or NULL.</td>
</tr>
<tr>
<td>envrnmnt</td>
<td>is the compiler environment clause or NULL.</td>
</tr>
<tr>
<td>compname</td>
<td>is the compiler name.</td>
</tr>
<tr>
<td>compopts</td>
<td>are the compiler options or NULL.</td>
</tr>
<tr>
<td>nlslang</td>
<td>is the zoned decimal code page in Oracle NLS format.</td>
</tr>
</tbody>
</table>
Defaults and Optional Parameters
The following table describes the defaults and optional parameters:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wind</td>
<td>is a warning indicator.</td>
</tr>
<tr>
<td>wmsgbsiz</td>
<td>is the warning message block size in bytes.</td>
</tr>
<tr>
<td>wmsgblk</td>
<td>is the warning message block. This is an OUT parameter.</td>
</tr>
</tbody>
</table>

Return Value
A RAW(2048) format conversion specification for NUMBER_TO_RAW.

Errors
ORA-8401 invalid compiler name
ORA-8414 error encountered
ORA-8437 invalid picture type in picture mask
ORA-8441 closed parenthesis is missing in picture mask
ORA-8443 syntax error in BLANK WHEN ZERO clause in mask options
ORA-8444 syntax error in JUSTIFIED clause in mask options
ORA-8445 syntax error in SIGN clause in mask options
ORA-8446 syntax error in SYCHRONIZED clause in mask options
ORA-8447 syntax error in USAGE clause in mask options
ORA-8448 syntax error in DECIMAL-POINT clause in mask options
ORA-8449 invalid numeric symbol found in picture mask
ORA-8450 invalid specification of CR in picture mask
ORA-8451 invalid specification of DB in picture mask
ORA-8452 specification of E in picture mask unsupported
ORA-8453 more than one V symbol specified in picture mask
ORA-8454 more than one S symbol specified in picture mask
ORA-8455 syntax error in CURRENCY SIGN environment clause
ORA-8456 no sign in picture mask but SIGN clause in mask options
ORA-8457 syntax error in SEPARATE CHARACTER option of SIGN clause
ORA-8458 invalid format parameter
ORA-8459 invalid format parameter length
ORA-8460 invalid environment clause in environment parameter
ORA-8467 error encountered converting Oracle number to type
ORA-8468 mask option option is not supported

Warnings
The following warnings can be issued, if warnings are enabled:
ORA-8498 picture mask overrides mask option USAGE... to DISPLAY
ORA-8499 picture mask options ignored by UTL_PG
RAW_TO_NUMBER_FORMAT

RAW_TO_NUMBER_FORMAT converts, according to the RAW_TO_NUMBER conversion format r2nfmt, a RAW byte-string rawval in the remote host internal format into an Oracle number.

**Syntax**

```sql
function RAW_TO_NUMBER_FORMAT (rawval IN RAW,
r2nfmt IN RAW) RETURN NUMBER;
```

where:

The following table describes the syntax:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rawval</td>
<td>Is the remote host data to be converted.</td>
</tr>
<tr>
<td>r2nfmt</td>
<td>Is a RAW(2048) format specification returned from MAKE_RAW_TO_NUMBER_FORMAT.</td>
</tr>
</tbody>
</table>

**Defaults**

None

**Return Value**

An Oracle number corresponding in value to r.

**Errors**

ORA-8414 error encountered
ORA-8429 raw data has invalid digit in display type data
ORA-8430 raw data missing leading sign
ORA-8431 raw data missing zero as defined in picture
ORA-8434 raw data has invalid trailing sign
ORA-8436 raw data has invalid sign digit
ORA-8458 invalid format parameter
ORA-8459 invalid format parameter length
ORA-8462 raw buffer contains invalid decimal data
ORA-8463 overflow converting decimal number to Oracle number
ORA-8464 input raw decimal data contains more than 42 digits
ORA-8466 raw buffer length len to short for spec
ORA-8467 error encountered converting Oracle number to type

**NUMBER_TO_RAW_FORMAT**

NUMBER_TO_RAW_FORMAT converts, according to the NUMBER_TO_RAW conversion format n2rfmt, an Oracle number numval of declared precision and scale into a RAW byte-string in the remote host internal format.

**Syntax**

```sql
function NUMBER_TO_RAW_FORMAT (numval IN NUMBER, n2rfmt IN RAW) RETURN RAW;
```

The following table describes the syntax:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numval</td>
<td>Is the Oracle number to be converted.</td>
</tr>
<tr>
<td>n2rfmt</td>
<td>Is a RAW(2048) format specification returned from MAKE_NUMBER_TO_RAW_FORMAT.</td>
</tr>
</tbody>
</table>

**Defaults**

None

**Return Value**

A RAW value corresponding in value to n.

**Errors**

ORA-8414 error encountered
ORA-8458 invalid format parameter
ORA-8459 invalid format parameter length
ORA-8467 error encountered converting Oracle number to type
WMSGCNT

WMSGCNT tests a wmsgblk to determine how many warnings, if any, are present.

Syntax

    function WMSGCNT (wmsgblk IN RAW) RETURN BINARY_INTEGER;

where:

    wmsgblk is the warning message block returned from one of the following functions:
    - MAKE_NUMBER_TO_RAW_FORMAT
    - MAKE_RAW_TO_NUMBER_FORMAT
    - NUMBER_TO_RAW
    - RAW_TO_NUMBER

Defaults

None

Return Value

A BINARY_INTEGER value equal to the count of warnings present in the RAW wmsgblk. The following table lists possible returned values:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;0</td>
<td>Indicating a count of warnings present in wmsgblk.</td>
</tr>
<tr>
<td>0</td>
<td>Indicating no warnings are present in wmsgblk.</td>
</tr>
</tbody>
</table>

Errors

-2    indicating an invalid message block.
WMSG

WMSG extracts a warning message specified by wmsgitem from wmsgblk.

**Syntax**

```sql
function WMSG (wmsgblk IN RAW,
               wmsgitem IN BINARY_INTEGER,
               wmsgno OUT BINARY_INTEGER,
               wmsgtext OUT VARCHAR2,
               wmsgfill OUT VARCHAR2) RETURN BINARY_INTEGER;
```

The following table describes the syntax:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wmsgblk</td>
<td>is a RAW warning message block returned from one of the following functions:</td>
</tr>
<tr>
<td></td>
<td>■ MAKE_NUMBER_TO_RAW_FORMAT</td>
</tr>
<tr>
<td></td>
<td>■ MAKE_RAW_TO_NUMBER_FORMAT</td>
</tr>
<tr>
<td></td>
<td>■ NUMBER_TO_RAW</td>
</tr>
<tr>
<td></td>
<td>■ RAW_TO_NUMBER</td>
</tr>
<tr>
<td>wmsgitem</td>
<td>is a BINARY_INTEGER value specifying which warning message to extract, numbered from 0 for the first warning through n minus 1 for the nth warning.</td>
</tr>
<tr>
<td>wmsgno</td>
<td>is an OUT parameter containing the BINARY_INTEGER (hexadecimal) value of the warning number. This value, after conversion to decimal, is documented in the Oracle9i Error Messages manual.</td>
</tr>
<tr>
<td>wmsgtext</td>
<td>is a VARCHAR2 OUT parameter value containing the fully-formatted warning message in ORA-xxxxx format, where xxxxx is the decimal warning number documented in the Oracle9i Error Messages manual.</td>
</tr>
<tr>
<td>wmsgfill</td>
<td>is a VARCHAR2 OUT parameter value containing the list of warning message parameters to be substituted into a warning message in the following format: warnparm1;;warnparm2;;...;;warnparmn</td>
</tr>
</tbody>
</table>
**UTL_PG Functions**

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>where each warning parameter is delimited by a double semicolon.</td>
</tr>
</tbody>
</table>

**Defaults**
None

**Return Value**
A BINARY_INTEGER value containing a status return code. The following are possible return codes:

- 0 indicating that wmsgno, wmsgtext, and wmsgfill are assigned and valid.

**Errors**
The following table describes the errors:

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Indicating the warning specified by wmsgitem was not found in wmsgblk.</td>
</tr>
<tr>
<td>-2</td>
<td>Indicating an invalid message block.</td>
</tr>
<tr>
<td>-3</td>
<td>Indicating wmsgblk is too small to contain the warning associated with wmsgitem. A partial or no warning message might be present for this particular wmsgitem.</td>
</tr>
<tr>
<td>-4</td>
<td>Indicating there are too many substituted warning parameters.</td>
</tr>
</tbody>
</table>
NUMBER_TO_RAW and RAW_TO_NUMBER Argument Values

This table lists the valid values for the format arguments for NUMBER_TO_RAW and RAW_TO_NUMBER and related functions.

Below are examples of some valid COBOL picture masks. Any valid IBM VS COBOL II picture mask may be used. For a complete explanation of COBOL picture masks, refer to the IBM VS COBOL II Application Programming Language Reference manual.

**mask**: COBOL picture mask

| PIC 9(n) | where 1 <= n <= 18 |
| PIC S9(n) | where 1 <= n <= 18 |
| PIC 9(n)V9(s) | where 1 <= n+s <= 18 |
| PIC S9(n)V9(s) | where 1 <= n+s <= 18 |
| PIC 9999999V99 |
| PIC V99999 |
| PIC SV9(5) |
| PIC 999.00 |
| PIC 99/99/99 |
| PIC +999.99 |
| PIC 999.99+ |
| PIC -999.99 |
| PIC 999.99- |
| PIC $$$$$,$$$99 |
| PIC 99999.99DB |
| PIC 99999.99CR |

**maskopts**: COBOL picture mask options

| COMP USAGE IS COMP |
| COMP USAGE IS COMPUTATIONAL |
| COMP-3 USAGE IS COMP-3 |
| COMP-3 USAGE IS COMPUTATIONAL-3 |
| COMP-4 USAGE IS COMP-4 |
| COMP-4 USAGE IS COMPUTATIONAL-4 |
| DISPLAY USAGE IS DISPLAY |
| SIGN IS LEADING |
SIGN IS LEADING SEPARATE
SIGN IS LEADING SEPARATE CHARACTER
SIGN IS TRAILING
SIGN IS TRAILING SEPARATE
SIGN IS TRAILING SEPARATE CHARACTER

envrmnt: COBOL environment clause

CURRENCY SIGN IS X where X is a valid currency sign character
DECIMAL-POINT IS COMMA

compname: COBOL compiler name

IBMVSCOBOLII

compopts: COBOL compiler options

(no values are supported at this time)
Use the following sample input statements and report output for the Procedural Gateway Administration Utility to guide you in designing your own PGAU statements.

This appendix contains these sample PGAU statements:

- Sample PGAU DEFINE DATA Statements on page E-2
- Sample PGAU DEFINE CALL Statements on page E-3
- Sample PGAU DEFINE TRANSACTION Statement on page E-3
- Sample PGAU GENERATE Statement on page E-3
- Sample Implicit Versioning Definitions on page E-4
- Sample PGAU REDEFINE DATA Statements on page E-8
- Sample PGAU UNDEFINE Statements on page E-9
- Sample PGAU REPORT Output on page E-9
- Sample PGAU Script File (pgadb2i.ctl) on page E-14
- Sample TIP Content Documentation on page E-16
- Sample TIP Trace Output on page E-19
- Sample TIP Output on page E-21
Sample PGAU DEFINE DATA Statements

DEFINE DATA EMPNO
   PLSNAME (EMPNO)
   USAGE (PASS)
   LANGUAGE (IBMVSCOBOLII)
   ( 01 EMP-NO PIC X(6).
   );

DEFINE DATA EMPREC
   PLSNAME (DCLEMP)
   USAGE (PASS)
   LANGUAGE (IBMVSCOBOLII)
   INFILE("emp.cob");

where the file emp.cob contains the following:

  01 DCLEMP.
   10 EMPNO   PIC X(6).
   10 FIRSTNME.
      49 FIRSTNME-LEN PIC S9(4) USAGE COMP.
      49 FIRSTNME-TEXT PIC X(12).
   10 MIDINIT  PIC X(1).
   10 LASTNAME.
      49 LASTNAME-LEN PIC S9(4) USAGE COMP.
      49 LASTNAME-TEXT PIC X(15).
   10 WORKDEPT PIC X(3).
   10 PHONENO  PIC X(4).
   10 HIREDATE PIC X(10).
   10 JOB     PIC X(8).
   10 EDLEVEL  PIC S9(4) USAGE COMP.
   10 SEX     PIC X(1)
   10 BIRTHDATE PIC X(10).
   10 SALARY  PIC S9999999V99 USAGE COMP-3.
   10 BONUS   PIC S9999999V99 USAGE COMP-3.
   10 COMM    PIC S9999999V99 USAGE COMP-3.

DEFINE DATA DB2INFO
   PLSNAME (DB2)
   USAGE (PASS)
   LANGUAGE (IBMVSCOBOLII)
   INFILE("db2.cob");

E-2   Oracle Procedural Gateway for APPC User's Guide
where the file db2.cob contains the following:

```cobol
01 DB2.
  05 SQLCODE          PIC S9(9) COMP-4.
  05 SQLERRM.
    49 SQLERRML       PIC S9(4) COMP-4.
    49 SQLERRT        PIC X(70).
  05 DSNERRM.
    49 DSNERRML       PIC S9(4) COMP-4.
    49 DSNERRMT       PIC X(240) OCCURS 8 TIMES INDEXED BY ERROR-INDEX
```

Sample PGAU DEFINE CALL Statements

```cobol
DEFINE CALL DB2IMAIN
  PKGCALL (PGADB2I_MAIN)
  PARMS ( (EMPNO      IN ),
          (EMPREC     OUT) );

DEFINE CALL DB2IDIAG
  PKGCALL (PGADB2I_DIAG)
  PARMS ( (DB2INFO    OUT) );
```

Sample PGAU DEFINE TRANSACTION Statement

```cobol
DEFINE TRANSACTION DB2I
  CALL (   DB2IMAIN,
          DB2IDIAG )
  SIDEPROFILE(CICSPROD)
  TPNAME(DB2I)
  LOGMODE(ORAPLU62)
  SYNCLEVEL(0)
  NLS_LANGUAGE("AMERICAN_AMERICA.WE8EBCDIC37C");
```

Sample PGAU GENERATE Statement

```cobol
GENERATE DB2I
  PKGNAME(PGADB2I)
  OUTFILE("pgadb2i");
```

A user’s high-level application now uses this TIP by referencing these PL/SQL datatypes passed and returned. The following table provides a description:
The examples are sample definitions of DATA, CALL, and TRANSACTION entries with implicit versioning.

This example creates a new DATA version of ‘EMPREC’ because ‘EMPREC’ DATA was defined previously:

```
DEFINE DATA EMPREC
  PLSDNAME (NEWEMP)
  USAGE (PASS)
  LANGUAGE (IBMVSCOBOLII)
  INFILE("emp2.cob");
```

where the file `emp2.cob` contains the following:

```
01 NEWEMP.
  10 EMPNO     PIC X(6).
  10 FIRSTNME.
      49 FIRSTNME-LEN PIC S9(4) USAGE COMP.
      49 FIRSTNME-TEXT PIC X(12).
  10 MIDINIT   PIC X(1).
  10 LASTNAME.
      49 LASTNAME-LEN PIC S9(4) USAGE COMP.
      49 LASTNAME-TEXT PIC X(15).
  10 WORKDEPT  PIC X(3).
  10 PHONENO   PIC X(3).
```

and the application calls:

```
PGADB2I.PGADB2I_INIT(trannum);
PGADB2I.PGADB2I_MAIN( trannum, empno, emprec );
PGADB2I.PGADB2I_DIAG( trannum, db2 );
PGADB2I.PGADB2I_TERM(trannum, termtype);
```
To determine which DATA version number was assigned, this SQL query can be issued:

```
SELECT MAX(pd.version)
FROM pga_data pd
WHERE pd.dname = 'EMPREC';
```

To determine additional information related to the updated version of ‘EMPREC’ this query can be used:

```
SELECT *
FROM pga_data pd
WHERE pd.dname = 'EMPREC';
```

This example creates a new CALL version of ‘DB2IMAIN’ because the ‘DB2IMAIN’ CALL was defined previously:

```
DEFINE CALL DB2IMAIN
    PKGCALL (PGADB2I_MAIN)
    PARMS ( (EMPNO IN                   ),
                (EMPREC OUT VERSION(ddddd) )  );
```

where ddddd is the version number of the EMPREC DATA definition queried after the previous DEFINE DATA updated EMPREC.

To determine which call version number was assigned, this SQL query can be issued:

```
SELECT MAX(pc.version)
FROM pga_call pc
WHERE pc.cname = 'DB2IMAIN';
```
To determine additional information related to the updated version of 'DB2IMAIN' this query can be used:

```
SELECT *
FROM pga_call pc
WHERE pc.cname = 'DB2IMAIN';
```

The DEFINE TRANSACTION example creates a new TRANSACTION version of 'DB2I' because the 'DB2I' TRANSACTION was defined previously. The essential difference of the new version of the DB2I transaction is that the first call uses a new PL/SQL record format "NEWEMP" (which corresponds to the COBOL NEWEMP format) to query the employee data.

---

**Warning:** Record format changes like that discussed above must be synchronized with the requirements of the remote transaction program. Changes to the PGA TIP alone result in errors. A new remote transaction program with the corequisite changes could be running on a separate CICS system and started through the change from "CICSPROD" to "CICSTEST" in the SIDEPROFILE parameter below.

---

DEFINE TRANSACTION DB2I
   CALL ( DB2IMAIN VERSION (ccccc),
          DB2IDIAG )
   SIDEPROFILE(CICSTEST)
   TPNAME(DB2I)
   LOGMODE(ORAPLU62)
   SYNCFLEVEL(0)
   NLS_LANGUAGE("AMERICAN_AMERICA.WE8EBCDIC37C");

where cccccc is the version number of the DB2IMAIN CALL definition queried after the previous DEFINE CALL updated DB2IMAIN.

There are two versions of the DB2I transaction definition in the PGA DD. The original uses the old "DCLEMP" record format and starts transaction "DB2I" on the production CICS system. The latest uses the "NEWEMP" record format and starts transaction "DB2I" on the test CICS system.

To determine which transaction version number was assigned, this SQL query can be issued:

```
SELECT MAX(pt.version)
FROM pga_trans pt
```
WHERE pt.tname = 'DB2I';

To determine additional information related to the updated version of 'DB2I' this query can be used:

SELECT *
FROM pga_trans pt
WHERE pt.tname = 'DB2I';

This example generates a new package using the previously defined new versions of the TRANSACTION, CALL, and DATA definitions:

GENERATE DB2I
VERSION(ttttt)
PKGNAME(NEWDB2I)
OUTFILE("pgadb2i");

where ttttt is the version number of the DB2I TRANSACTION definition queried after the previous DEFINE TRANSACTION updated DB2I.

Note that the previous PL/SQL package files pgadb2i.pkh and pgadb2i.pkb are overwritten. To keep the new package separate, change the output file specification. For example:

GENERATE DB2I
VERSION(ttttt)
PKGNAME(NEWDB2I)
OUTFILE("newdb2i");

A user's high-level application now uses this TIP by referencing the PL/SQL datatypes passed and returned, described in the following table:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEWDB2I.EMPNO</td>
<td>Is a PL/SQL variable corresponding to COBOL EMPNO.</td>
</tr>
<tr>
<td>NEWDB2I.NEWEMP</td>
<td>Is a PL/SQL RECORD corresponding to COBOL NEWEMP.</td>
</tr>
<tr>
<td>NEWDB2I.DB2</td>
<td>Is a PL/SQL RECORD corresponding to COBOL DB2.</td>
</tr>
</tbody>
</table>

and the application calls:

NEWDB2I.PGADB2I_INIT(trannum);
NEWDB2I.PGADB2I_MAIN( trannum, empno, newemp );
NEWDB2I.PGADB2I_DIAG( trannum, db2 );
NEWDB2I.PGADB2I_TERM(trannum, terctype);

**Sample PGAU REDEFINE DATA Statements**

Single-field redefinition in which EDLEVEL USAGE becomes COMP-3:

```cobol
REDEFINE DATA EMPREC
    PLSNAME(DCLEMP)
    LANGUAGE(IBMVSCOBOLII)
    FIELD(EDLEVEL)
    PLSFNAME(PLSRECTYPE)
    {
        10 EDLEVEL PIC S9(4) USAGE IS COMP-3.
    };
```

By default, this redefines the latest version of EMPREC which implicitly affects the latest call and transaction definitions which refer to it.

Sample multi-field redefinition in which the employee's first and last name fields are expanded and the employee's middle initial is removed.

```cobol
REDEFINE DATA EMPREC
    VERSION(1)
    PLSNAME(DCLEMP)
    LANGUAGE(IBMVSCOBOLII)
    INFILE("emp1.cob");

where the file emp1.cob contains the following:

```cobol
01 DCLEMP.
  10 EMPNO                     PIC X(6).
  10 FIRSTNME.
      49 FIRSTNME-LEN         PIC S9(4) USAGE COMP.
      49 FIRSTNME-TEXT        PIC X(15).
  10 LASTNAME.
      49 LASTNAME-LEN         PIC S9(4) USAGE COMP.
      49 LASTNAME-TEXT        PIC X(20).
  10 WORKDEPT                  PIC X(3).
  10 PHONENO                   PIC X(4).
  10 HIREDATE                  PIC X(10).
  10 JOB                       PIC X(8).
  10 EDLEVEL                   PIC S9(4) USAGE COMP.
  10 SEX                       PIC X(1).
  10 BIRTHDATE                 PIC X(10).
  10 SALARY                    PIC S99999999V99 USAGE COMP-3.
  10 BONUS                     PIC S99999999V99 USAGE COMP-3.
```
The assumption is that version 1 of the data definition for 'EMPREC' is to be redefined. This causes a redefinition of the first 'EMPREC' sample data definition without changing the version number. Thus, existing call and transaction definitions which referenced version 1 of 'EMPREC' automatically reflect the changed 'EMPREC'. This change becomes effective when a TIP is next generated for a transaction that references the call which referenced version 1 of 'EMPREC'.

This implicitly affects both versions of the transaction because both refer to EMPREC in the second call to update the employee data.

Sample PGAU UNDEFINE Statements

These samples illustrate the deletion of a specific version of a definition which has multiple versions, followed by deletion of all versions of a specific named definition.

Deletion of DATA Definitions:

UNDEFINE DATA EMPREC VERSION (ddddd);
UNDEFINE DATA EMPREC VERSION (ALL);
UNDEFINE CALL DB2IMAIN VERSION (cccccc);
UNDEFINE CALL DB2IMAIN VERS (all);
UNDEFINE TRANSACTION DB2I vers (tttttt);
UNDEFINE TRANSACTION DB2I vers (all);

Note that the previous UNDEFINE statements leave the DATA definition for EMPNO and the CALL definition for DB2IDIAG in the PGA DD.

Sample PGAU REPORT Output

/* Transaction DB2I version 298 created by PGAADMIN on FEB 14, 1995 17:38.02 */
DEFINE TRANSACTION DB2I
CALL (DB2IMAIN VERSION(672),
      DB2IDIAG VERSION(673))
SIDEPROFILE(CICSPGA)
TPNAME(DB2I)
LOGMODE(CRAPLU62)
SYNCREQ(0)
NLS_LANGUAGE(americanamerica.we8ebcdic37c);
Sample PGAU REPORT Output

PGAU> report transaction db2i with calls;

/* Call DB2IMAIN version 672 created by PGAADMIN on FEB 14, 1995 17:38.01 */
DEFINE CALL DB2IMAIN PKGCALL(PGADB2I_MAIN)
PARMS(EMPNO IN VERSION(638),
    EMPREC OUT VERSION(639));

/* Call DB2IDIAG version 673 created by PGAADMIN on FEB 14, 1995 17:38.02 */
DEFINE CALL DB2IDIAG PKGCALL(PGADB2I_DIAG)
PARMS(DB2INFO OUT VERSION(640));

/* Transaction DB2I version 298 created by PGAADMIN on FEB 14, 1995 17:38.02 */
DEFINE TRANSACTION DB2I
    CALL (DB2IMAIN VERSION(672),
          DB2IDIAG VERSION(673))
    SIDEPROFILE(CICSPGA)
    TPNAME(DB2I)
    LOGMODE(ORAPLU62)
    SYNCELEVEL(0)
    NLS_LANGUAGE(american_america.e8ebcdic37c);

PGAU> report transaction db2i with data;

/* Data Definition EMPNO version 638 created/updated by PGAADMIN */
/* on FEB 14, 1995 17:02.58 */
DEFINE DATA EMPNO LANGUAGE(IBMVSCOBOLII) USAGE(PASS)
    (01 EMPNO PICTURE IS X(6));

/* Data Definition EMPREC version 639 created/updated by PGAADMIN */
/* on FEB 14, 1995 17:02.59 */
DEFINE DATA EMPREC PLSDNAME(DCLEMP) LANGUAGE(IBMVSCOBOLII) USAGE(PASS)
    (01 DCLEMP.
        10 EMPNO PICTURE IS X(6).
        10 FIRSTNME.
        * Field FIRSTNME-LEN has PL/SQL name FIRSTNME_LEN
          49 FIRSTNME-LEN USAGE IS COMP PICTURE IS S9(4).
        * Field FIRSTNME-TEXT has PL/SQL name FIRSTNME_TEXT
          49 FIRSTNME-TEXT PICTURE IS X(12).
        10 MIDINIT PICTURE IS X(1)).
10 LASTNAME.
  * Field LASTNAME-LEN has PL/SQL name LASTNAME_LEN
  49 LASTNAME-LEN USAGE IS COMP PICTURE IS S9(4).
  * Field LASTNAME-TEXT has PL/SQL name LASTNAME_TEXT
  49 LASTNAME-TEXT PICTURE IS X(15).
  10 WORKDEPT PICTURE IS X(3).
  10 PHONENO PICTURE IS X(4).
  10 HIREDATE PICTURE IS X(10).
  10 JOB PICTURE IS X(8).
  10 EDLEVEL USAGE IS COMP PICTURE IS S9(4).
  10 SEX PICTURE IS X(1).
  10 BIRTHDATE PICTURE IS X(10).
  10 SALARY USAGE IS COMP-3 PICTURE IS S9999999V99.
  10 BONUS USAGE IS COMP-3 PICTURE IS S9999999V99.
  10 COMM USAGE IS COMP-3 PICTURE IS S9999999V99.

); /* Call DB2IMAIN version 672 created by PGAADMIN on FEB 14, 1995 17:38.01*/
DEFINE CALL DB2IMAIN PKGCALL(PGADB2I_MAIN)
  PARMs(EMPNO IN VERSION(638),
         EMPREC OUT VERSION(639));

/* Data Definition DB2INFO version 640 created/updated by PGAADMIN */
/* on FEB 14,1995 17:02.00*/
DEFINE DATA DB2INFO PLSDNAME(DB2) LANGUAGE(IBMVSCOBOLII) USAGE(PASS)
(
  01 DB2.
    05 SQLCODE USAGE IS COMP-4 PICTURE IS S9(9).
    05 SQLERRM.
      49 SQLERRM USAGE IS COMP-4 PICTURE IS S9(4).
      49 SQLERRM PICTURE IS X(70).
    05 DSNERRM.
      49 DSNERRM USAGE IS COMP-4 PICTURE IS S9(4).
      49 DSNERRM PICTURE IS X(240) OCCURS 8 TIMES.
); /* Call DB2IDIAG version 673 created by PGAADMIN on FEB 14, 1995 17:38.02*/
DEFINE CALL DB2IDIAG PKGCALL(PGADB2I_DIAG)
  PARMs(DB2INFO OUT VERSION(640));

/* Transaction DB2I version 298 created by PGAADMIN on FEB 14, 1995 17:38.02*/
DEFINE TRANSACTION DB2I
  CALL (DB2IMAIN VERSION(672),
         DB2IDIAG VERSION(673))
  SIDEPROFILE(CICSPGA)
TPNAME(DB2I)
LOGMODE(ORAPLU62)
SYNCELEVEL(0)
NLS_LANGUAGE(american_america.we8ebcidic37c);

PGAU> report transaction db2i with data with debug;

/* Data Definition EMPNO version 638 created/updated by PGAADMIN */
/* on FEB 14, 1995 17:02.58 */
DEFINE DATA EMPNO /* d#=565 */ LANGUAGE(IBMVSOCOBOLII) USAGE(PASS)
  {
    * f#=4005
      01 EMPNO PICTURE IS X(6).
  };

/* Data Definition EMPREC version 639 created/updated by PGAADMIN */
/* on FEB 14,1995 17:02.59 */
DEFINE DATA EMPREC /* d#=566 */ PLSDNAME(DCLEMP) LANGUAGE(IBMVSOCOBOLII) USAGE(PASS)
  {
    * f#=4006
      01 DCLEMP.
    * f#=4007
      10 EMPNO PICTURE IS X(6).
    * f#=4008
      10 FIRSTNME.
    * f#=4009
      Field FIRSTNME-LEN has PL/SQL name FIRSTNME_LEN
      49 FIRSTNME-LEN USAGE IS COMP PICTURE IS S9(4).
    * f#=4010
      Field FIRSTNME-TEXT has PL/SQL name FIRSTNME_TEXT
      49 FIRSTNME-TEXT PICTURE IS X(12).
    * f#=4011
      10 MIDINIT PICTURE IS X(1).
    * f#=4012
      10 LASTNAME.
    * f#=4013
      Field LASTNAME-LEN has PL/SQL name LASTNAME_LEN
      49 LASTNAME-LEN USAGE IS COMP PICTURE IS S9(4).
    * f#=4014
      Field LASTNAME-TEXT has PL/SQL name LASTNAME_TEXT
      49 LASTNAME-TEXT PICTURE IS X(15).
    * f#=4015
      10 WORKDEPT PICTURE IS X(3).
    * f#=4016
10 PHONENO PICTURE IS X(4).
* f#=4017
10 HIREDATE PICTURE IS X(10).
* f#=4018
10 JOB PICTURE IS X(8).
* f#=4019
10 EDLEVEL USAGE IS COMP PICTURE IS S9(4).
* f#=4020
10 SEX PICTURE IS X(1).
* f#=4021
10 BIRTHDATE PICTURE IS X(10).
* f#=4022
10 SALARY USAGE IS COMP-3 PICTURE IS S9999999V99.
* f#=4023
10 BONUS USAGE IS COMP-3 PICTURE IS S9999999V99.
* f#=4024
10 COMM USAGE IS COMP-3 PICTURE IS S9999999V99.
);

/* Call DB2IMAIN version 672 created by PGAADMIN on FEB 14, 1995 17:38.01*/
DEFINE CALL DB2IMAIN PKGCALL(PGADB2I_MAIN) /* c#=672 */
PARMS(EMPNO IN VERSION(638) /* d#=565 */, EMBED不锈(EMPNO IN EMBED(638) /* d#=565 */);

/* Data Definition DB2INFO version 640 created/updated by PGAADMIN */
/* on FEB 14,1995 17:02.00*/
DEFINE DATA DB2INFO /* d#=567 */ PLSDNAME(DB2) LANGUAGE(IBMVCLOBOLII) USAGE(PASS)
{
    * f#=4025
    01 DB2.
    * f#=4026
    05 SQLCODE USAGE IS COMP-4 PICTURE IS S9(9).
    * f#=4027
    05 SQLERRM.
    * f#=4028
    49 SQLERRML USAGE IS COMP-4 PICTURE IS S9(4).
    * f#=4029
    49 SQLERRMT PICTURE IS X(70).
    * f#=4030
    05 DSNERRM.
    * f#=4031
    49 DSNERRML USAGE IS COMP-4 PICTURE IS S9(4).
    * f#=4032
    49 DSNERRMT PICTURE IS X(240) OCCURS 8 TIMES.
};
Sample PGAU Script File (pgadb2i.ctl)

/* Call DB2IDIAG version 673 created by PGAADMIN on FEB 14, 1995 17:38.02*/
DEFINE CALL DB2IDIAG PKGCALL(PGADB2I_DIAG) /* c#=673 */
P ARMS(DB2INFO OUT VERSION(640) /* d#=567 */);

/* Transaction DB2I version 298 created by PGAADMIN on FEB 14, 1995 17:38.02*/
DEFINE TRANSACTION DB2I /* t#=298 */
   CALL (DB2IMAIN VERSION(672) /* c#=672 */,
          DB2IDIAG VERSION(673) /* c#=673 */)
   SIDEPROFILE(CICSPGA) /* a#=2 */
   TPNAMES(DB2I) /* a#=3 */
   LOGMODE(ORAPLU62) /* a#=4 */
   SYNCLEVEL(0) /* a#=6 */
   NLS_LANGUAGE(american_america.we8ebcdic37c) /* a#=7 */;

PGAU>

PGAU> spool off

Sample PGAU Script File (pgadb2i.ctl)

# Copyright (c) 1994,1995 by Oracle Corporation. All rights reserved.
#
# NAME
#   pgadb2i.ctl
# DESCRIPTION
#   PGAU script to define and generate the Transaction Interface Package
#   for the DB2I CICS demo transaction.
# MODIFIED     (MM/DD/YY)
#
set echo on
#
# First, undefine any previously defined version(s) of the transaction.
#
undefine transaction db2i vers(all);
undefine call db2imain vers(all);
undefine call db2idiag vers(all);
undefine data empno vers(all);
undefine data emprec vers(all);
undefine data db2info vers(all);
#
# Define data items transferred by this transaction. For details on the
# actual data format, refer to the COBOL source files specified by the
#(infile parameters.
#define data empno plsdname(empno) usage(pass) language(ibmvscobolii)
    infile("empno.cob");
define data emprec plsdname(dclemp) usage(pass) language(ibmvscobolii)
    infile("emp.cob");
define data db2info plsdname(db2) usage(pass) language(ibmvscobolii)
    infile("db2.cob");
#
#define each set of data exchanges with the transaction. A separate
#define callable function will be generated in the TIF for each of these
#define exchanges.
#define call db2imain pkgcall(pgadb2i_main)
    parms((empno in),(emprec out));
define call db2idiag pkgcall(pgadb2i_diag)
    parms((db2info out));
#
#define the transaction as the set of data exchanges defined above, and
#define its LU6.2 information (SNA Side Profile defining the LU where the
#define transaction resides, TP name of the transaction, and MODE name used for
#define sessions).
#define transaction db2i call(db2imain,db2idiag)
    sideprofile(CICSPGA) tpname(DB2I) logmode(oraplu62) synclvel(0)
    nls_language("american_america.we8ebcdic37c");
#
#define Generate the PL/SQL Transaction Interface Package for the transaction,
#define and place it in the file specified by the outfile parameter.
#define generate db2i pkgname(pgadb2i) pganode(pga) outfile("pgadb2i");
Sample TIP Content Documentation

PG4APPC TIP pgadb2i contents:

TIP generation date: 15-FEB-02 10:14:17  
TIP generation user: PGAADMIN  
PG DD version: 9.2.0.1.0  
PGAU version: 9.2.0.1.0

/*------------------------------------------------------------------*/  
/*  TIP user-transaction definition                                */  
/*------------------------------------------------------------------*/

remote host transaction program name: DB2I  
remote host transaction sync level: 0  
remote host application LU name: ORAPLU62  
gateway CPI-C Side Profile name: CICSPGA  
remote host environment: IBM370  
remote host NLS name: AMERICAN_AMERICA.WE8EBCDIC37C

/*------------------------------------------------------------------*/  
/*  TIP user-transaction-data type declarations                    */  
/*------------------------------------------------------------------*/

PL/SQL Variable name   PL/SQL Variable Type

TYPE FIRSTNME_Typ is RECORD (  
    FIRSTNME_LEN  NUMBER(4,0),  
    FIRSTNME_TEXT  CHAR(12));

TYPE LASTNAME_Typ is RECORD (  
    LASTNAME_LEN  NUMBER(4,0),  
    LASTNAME_TEXT  CHAR(15));

TYPE DCLEMP_Typ is RECORD (  
    EMPNO  CHAR(6),  
    FIRSTNME  FIRSTNME_Typ,  
    MIDINIT  CHAR(1),  
    LASTNAME  LASTNAME_Typ,  
    WORKDEPT  CHAR(3),  
    PHONENO  CHAR(4),  
    HIREDATE  CHAR(10),  
    JOB  CHAR(8),  
    EDLEVEL  NUMBER(4,0),  

SEX CHAR(1),
BIRTHDATE CHAR(10),
SALARY NUMBER(9,2),
BONUS NUMBER(9,2),
COMM NUMBER(9,2));

TYPE SQLERRM_Typ is RECORD (
  SQLERRML NUMBER(4,0),
  SQLERRMT CHAR(70));

DSNERRMT_Key BINARY_INTEGER;

TYPE DSNERRMT_Tbl is TABLE of CHAR(240)
  INDEX by BINARY_INTEGER;

TYPE DSNERRM_Typ is RECORD (
  DSNERRML NUMBER(4,0),
  DSNERRMT DSNERRMT_Tbl);

TYPE DB2_Typ is RECORD (
  SQLCODE NUMBER(9,0),
  SQLERRM SQLERRM_Typ,
  DSNERRM DSNERRM_Typ);

/*------------------------------------------------------------------*/
/*  TIP default functions                                          */
/*------------------------------------------------------------------*/

TYPE override_Typ IS RECORD (
  tranname VARCHAR2(2000), /* Transaction Program */
  transync BINARY_INTEGER, /* RESERVED */
  trannls VARCHAR2(50),    /* RESERVED */
  oltpname VARCHAR2(2000), /* Logical Unit */
  oltpmode VARCHAR2(2000), /* LOG Mode Entry */
  netaddr VARCHAR2(2000));/* Side Profile */

FUNCTION pgadb2i_init( /* init standard */
  tranuse IN OUT BINARY_INTEGER)
RETURN INTEGER;

FUNCTION pgadb2i_init( /* init override */
  tranuse IN OUT BINARY_INTEGER,
  override IN override_Typ)
RETURN INTEGER;
FUNCTION pgadb2i_init( /* init diagnostic */
    tranuse IN OUT BINARY_INTEGER,
    tipdiag IN CHAR)
RETURN INTEGER;

FUNCTION pgadb2i_init( /* init over-diag */
    tranuse IN OUT BINARY_INTEGER,
    override IN override_Typ,
    tipdiag IN CHAR)
RETURN INTEGER;

FUNCTION pgadb2i_term( /* terminate */
    tranuse IN BINARY_INTEGER,
    termtype IN BINARY_INTEGER)
RETURN INTEGER;

/*----------------------------------------*/
/* TIP user-transaction-call function definitions */
/*----------------------------------------*/
FUNCTION PGADB2I_MAIN(
    tranuse IN BINARY_INTEGER,
    EMPNO IN CHAR,
    DCLEMP OUT DCLEMP_Typ)
RETURN INTEGER;

FUNCTION PGADB2I_DIAG(
    tranuse IN BINARY_INTEGER,
    DB2 OUT DB2_Typ)
RETURN INTEGER;

/*----------------------------------------*/
/* TIP field variables */
/*----------------------------------------*/

PL/SQL name                                      PL/SQL type
EMPNO                                             CHAR(6)
DCLEMP                                             DCLEMP_Typ
DCLEMP.EMPNO                                       CHAR(6)
DCLEMP.FIRSTNME                                    FIRSTNME_Typ
DCLEMP.FIRSTNME.FIRSTNME_LEN                       NUMBER(4,0)
DCLEMP.FIRSTNME.FIRSTNME_TEXT                      CHAR(12)
DCLEMP.MIDINIT                                     CHAR(1)
DCLEMP.LASTNAME                                   LASTNAME_Typ
Sample TIP Trace Output

SQL> set serveroutput on size 20000
SQL> exec db2idriv('000340','11110000');
Name
JASON R GOUNOT
Sex Birthdate Edlevel
M 05/17/1926 16
Empno Dept Job Phone Hiredate Salary Bonus Commission
000340 E21 FIELDREP 5698 05/05/1947 23840.00 500.00 1907.00

PL/SQL procedure successfully completed.

SQL> exec rtrace('PGADB2I');
UTL_PG warnings enabled
function entry/exit trace enabled
data conversion trace enabled
gateway exchange trace enabled
enter PGADB2I_init diagnostic

time 22-MAR-1995 13:54:47
from PGAADMIN
exit PGADB2I_init diagnostic
enter PGADB2I_MAIN
enter updsndlt
sendcnt, parmlenm => 1, 6
sendllst(sendcnt) => 0
sendllst(sendcnt) => 6
exit updsndlt
EMPNO CHAR(6)
at pos, off, len => 1, 1, 6
enter updrcvlt
recvnt, parmlenm => 1, 91
recvllst(recvnt) => 0
recvllst(recvnt) => 91
exit updrcvlt
enter dataxfer
tranuse => 2
enter bldsxfrl
sendmax => 1
sendbufl => 6
sxfrllst => 0000000100000006
exit bldsxfrl
recvmax => 1
enter bldrxfrl
recvbufl => 91
rxfrllst => 000000010000005B
exit bldrxfrl
enter xfersr
tranuse => 2
transid => 200EBE08200F2E1800000000
sendbufl => 6
sxfrllst => 000000010000006000000000
recvbufl => 91
rxfrllst => 000000010000005B00000000
recvbufl => 91
rxfrllst => 000000010000005B00000000
exit xfersr
exit dataxfer
enter getrcvla
recvnt => 1
recvvalst(recvnt) => 0
recvvalst(recvnt) => 91
numplen => 91
exit getrcvla
DCLEMP.EMPNO CHAR(6)
at pos, off, len => 1, 1, 6
DCLEMP.FIRSTNME.FIRSTNME_LEN NUMBER(4,0)
at pos, off, len => 7, 7, 2
DCLEMP.FIRSTNME.FIRSTNME_TEXT CHAR(12)
at pos, off, len => 9, 9, 12
DCLEMP.MIDINIT CHAR(1)
at pos, off, len => 21, 21, 1
DCLEMP.LASTNAME.LASTNAME_LEN NUMBER(4,0)
at pos, off, len => 22, 22, 2
DCLEMP.LASTNAME.LASTNAME_TEXT CHAR(15)
at pos, off, len => 24, 24, 15
DCLEMP.WORKDEPT CHAR(3)
at pos, off, len => 39, 39, 3
DCLEMP.PHONENO CHAR(4)
at pos, off, len => 42, 42, 4
DCLEMP.HIREDATE CHAR(10)
at pos, off, len => 46, 46, 10
DCLEMP.JOB CHAR(8)
at pos, off, len => 56, 56, 8
DCLEMP.EDLEVEL NUMBER(4,0)
at pos, off, len => 64, 64, 2
DCLEMP.SEX CHAR(1)
at pos, off, len => 66, 66, 1
DCLEMP.BIRTHDATE CHAR(10)
at pos, off, len => 67, 67, 10
DCLEMP.SALARY NUMBER(9,2)
at pos, off, len => 77, 77, 5
DCLEMP.BONUS NUMBER(9,2)
at pos, off, len => 82, 82, 5
DCLEMP.COMM NUMBER(9,2)
at pos, off, len => 87, 87, 5
exit PGADB2I_MAIN
enter PGADB2I_term
exit PGADB2I_term
time 22-MAR-1995 13:54:49
from PGAADMIN
no more PGADB2I trace messages

PL/SQL procedure successfully completed.

SQL> spool off

Sample TIP Output

SQL> exec db2idriv('000340');
Sample TIP Output

Name
JASON R GOUNOT
Sex Birthdate Edlevel
M 05/17/1926 16
Empno Dept Job Phone Hiredate Salary Bonus Commission
000340 E21 FIELDREP 5698 05/05/1947 23840.00 500.00 1907.00

PL/SQL procedure successfully completed.

SQL> spool off
The first sample program in this appendix contains the COBOL source code. All APPC low-level interface code is provided by CICS. The second sample program (pgacics.sql) in this appendix shows a simple PL/SQL procedure that invokes the CICS FLIP transaction; this was shipped as part of the gateway’s Installation Verification Procedure.

This appendix contains the following sections:

- Description of Sample CICS APPC Transaction on page F-2
- COBOL Source Code on page F-2
- Sample pgacics.sql File on page F-13
Description of Sample CICS APPC Transaction

The program uses the following CICS calls for APPC conversation management:

- **EXEC CICS ASSIGN FACILITY**
  This call extracts the APPC conversation ID for use in subsequent APPC calls.

- **EXEC CICS RECEIVE**
  This call receives data from the conversation partner through the APPC interface.

- **EXEC CICS SEND**
  This call sends data to the conversation partner through the APPC interface.

- **EXEC CICS RETURN**
  This call deallocates the APPC conversation if necessary.

For further information about the EXEC CICS interface to APPC, refer to IBM publication *CICS/ESA Application Program Reference*.

The PL/SQL programs used to drive this transaction and act as the conversation partner can be found in the $ORACLE_HOME/pg4appc/demo/CICS directory in files *pgadb2i.pkh* and *pgadb2id.pkb*. The TIP is in the first file and the driver program is in the second file.

COBOL Source Code

This source code is in file *pgadb2i.cob* in the $ORACLE_HOME/pg4appc/demo/CICS directory.

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                               *
*  Copyright (c) 1993, 1994 by Oracle Corporation.               *
*  All rights reserved.                                         *
*                                                               *
*  This program is distributed on an as-is basis. The user is    *
*  free to modify and use this program as long as it is not      *
*  distributed to other parties. Such modification and use is   *
*  the sole responsibility of the user and not of Oracle        *
*  Corporation.                                                 *
*                                                               *
*  This program is a sample CICS transaction to show the use of *
*  the Oracle Procedural Gateway for APPC to retrieve data from *
*  DB2 into an Oracle application environment.                   *
* *
* The program uses DB2 sample employee table (DSN8230.EMP for * DB2 V2.3). This sample table must be installed in your DB2 * system before using this program. Refer to the IBM DB2 * documentation for more information on installing the sample * EMP table.

* The SQL qualifier used in this program for the sample data- * base is DSN8230, which is distributed with DB2 V2.3. If you * are running a different version of DB2, the SQL qualifier * is probably different. You should change the SQL statements * in this program to specify the correct SQL qualifier for the * sample database.

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
EJECT
IDENTIFICATION DIVISION.
PROGRAM-ID. PGADB2I.
AUTHOR. PGADEV.
DATE-COMPILED.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-370.
OBJECT-COMPUTER. IBM-370.

DATA DIVISION.
WORKING-STORAGE SECTION.

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* Declared SQL Interface Variables                              *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

* SQL Communications Area

   EXEC SQL INCLUDE SQLCA END-EXEC.

* DCLGEN for referenced table EMP

   EXEC SQL INCLUDE EMP END-EXEC.

EJECT
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* Working storage                                             *
* Input area for receiving employee number from partner, and
* length field for passing lengths to/from CICS.

01 INPUT-AREA PIC X(6).
77 IN-LENGTH PIC S9(4) COMP.

* Flags: TERM is set to 'Y' when the program should terminate.
* DB2-DIAG is set to 'Y' when DB2 diagnostic information needs
* to be sent to the partner.

77 TERM PIC X VALUE 'N'.
77 DB2-DIAG PIC X VALUE 'N'.

* APPC conversation ID, CICS response code, and CICS state code

77 WS-CONVID PIC X(4).
77 WS-RESP PIC S9(8) COMP.
77 WS-STATE PIC S9(7) COMP VALUE +0.

* DB2 Diagnostic information area (1998 bytes)

01 DB2-INFO.
  05 DB2-SQLCODE PIC S9(9) COMP.
  05 DB2-SQLERRRM.
    49 DB2-SQLERRML PIC S9(4) COMP.
    49 DB2-SQLERRMC PIC X(70).
  05 DB2-DSNERRRM.
    07 DB2-DSNERRML PIC S9(4) COMP VALUE +1920.
    07 DB2-DSNERRMT PIC X(240) OCCURS 8 TIMES
       INDEXED BY ERROR-INDEX.
  77 DB2-DSNERRMT-LEN PIC S9(9) COMP VALUE +240.

* Debug message text areas

01 RESP-MSG.
  05 FILLER PIC X(13)
     VALUE 'CICS RESP IS '.
  05 DISP-RESP PIC 9(8).

01 STATE-MSG.
  05 FILLER PIC X(14)
     VALUE 'CICS STATE IS '.
05 DISP-STATE PIC 9(7).
77 DEBUG-TEXT PIC X(40).

PROCEDURE DIVISION.

* Main logic is here. The diagram below shows the relationship
* between the partner application, the Procedural Gateway for
* APPC, and this program.

*                  PL/SQL                      Procedural Gateway      DB2I CICS
*     Partner                   for APPC               Transaction
* --------------------------- ----------------------- ----------------------
* Call PGAINIT  +---->               Set up control blocks
*                  Issue APPC ALLOCATE
*                  <----+                  Program initiates
* Call PGAXFER  +---->               <----+ Issue APPC RECEIVE
*  to send EMPNO and
*  receive EMP row
*  Issue APPC SEND
*  +----> RECEIVE completes
*  Issue APPC RECEIVE  Perform DB2 query
*  Issue APPC SEND
*  Issue CICS RETURN
*  (causes APPC
*  DEALLOCATE)
*  Receive completes
*  <----+
* Process EMP row
* Call PGATERM  +---->               Clean up control blocks
*  <----+
*  
* Perform all initialization

PERFORM DB2I-INITIALIZE.

* Receive employee number from the partner
PERFORM DB2I-RECEIVE-EMPNO.

* Open a DB2 cursor for the EMP table
  PERFORM DB2I-OPEN-CURSOR.

* Fetch the EMP row and send it back to the partner
  PERFORM DB2I-FETCH-ROW.

* Close the DB2 cursor
  PERFORM DB2I-CLOSE-CURSOR.

* Terminate (deallocates the conversation)
  EXEC CICS RETURN
  END-EXEC.

EJECT

DB2I-INITIALIZE.

* This routine performs all initialization, including the
  * APPC conversation initialization.
  *
  INITIALIZE INPUT-AREA.

* Reinitialize PGADB2I temporary storage queue for debugging
  EXEC CICS IGNORE CONDITION QIDERR
  END-EXEC.

  EXEC CICS DELETEQ TS QUEUE('PGADB2I')
  END-EXEC.

  EXEC CICS HANDLE CONDITION QIDERR
  END-EXEC.

* Get APPC conversation ID
  EXEC CICS ASSIGN FACILITY (WS-CONVID)
    RESP (WS-RESP)
EJECT

DB2I-RECEIVE-EMPNO.

* This routine receives the employee number of the DB2 EMP row requested from the partner. If the data received is the wrong length, or an APPC error occurs, the transaction is abended.

EXEC CICS IGNORE CONDITION LENGERR
END-EXEC.

* Receive the employee number of the DB2 EMP row to be retrieved. This was sent from the partner application via a PGAXFER call to the Procedural Gateway for APPC.

MOVE LENGTH OF EMPNO TO IN-LENGTH.

EXEC CICS RECEIVE
  CONVID(WS-CONVID)
  INTO(INPUT-AREA)
  LENGTH(IN-LENGTH)
  STATE(WS-STATE)
  RESP(WS-RESP)
END-EXEC.

* Check for APPC errors and abend if one is detected

IF WS-RESP NOT = DFHRESP(NORMAL) AND
  WS-RESP NOT = DFHRESP(EOC) THEN
  MOVE 'FAILURE RECEIVING EMPNO' TO DEBUG-TEXT
  PERFORM ABEND
ELSE
  IF EIBNODAT = HIGH-VALUES THEN
    MOVE 'RECEIVED NO EMPNO' TO DEBUG-TEXT
    PERFORM ABEND
END-IF
IF EIBERR = HIGH-VALUES THEN
   MOVE 'FAILURE RECEIVING EMPNO' TO DEBUG-TEXT
   PERFORM ABEND
END-IF
END-IF.

* Check length of input received and abend if incorrect

IF IN-LENGTH NOT = LENGTH OF EMPNO THEN
   MOVE 'INCORRECT EMPNO LENGTH RECEIVED' TO DEBUG-TEXT
   PERFORM ABEND
END-IF.

EJECT

DB2I-OPEN-CURSOR.

* *
* This routine declares and opens a SQL cursor for the DB2
* EMP table access.
*

EXEC SQL
   DECLARE CURSOR1 CURSOR FOR
   SELECT VALUE(EMPNO,''),
      VALUE(FIRSTNME,'No 1st Name'),
      VALUE(MIDINIT,''),
      VALUE(LASTNAME,'No Last Name'),
      VALUE(WORKDEPT,''),
      VALUE(PHONENO,''),
      VALUE(CHAR(HIREDATE),''),
      VALUE(JOB,''),
      VALUE(EDLEVEL,0),
      VALUE(SEX,''),
      VALUE(CHAR(BIRTHDATE),''),
      VALUE(SALARY,0), VALUE(BONUS,0), VALUE(COMM,0)
   FROM DSN8230.EMP
   WHERE EMPNO = :INPUT-AREA
END-EXEC.

EXEC SQL
   OPEN CURSOR1
END-EXEC.

IF SQLCODE NOT = +0 THEN


PERFORM DB2I-SQL-ERROR
PERFORM ABEND
END-IF.

EJECT

DB2I-CLOSE-CURSOR.

*  This routine closes the SQL cursor.
*

EXEC SQL
   CLOSE CURSOR1
END-EXEC.
IF SQLCODE NOT = +0 THEN
   PERFORM DB2I-SQL-ERROR
   PERFORM ABEND
   END-IF.
EJECT

DB2I-FETCH-ROW.

*  This routine fetches the requested row from DB2 and sends
*  it to the partner. If the row is not found, a blank row is
*  sent to the partner. If a DB2 SQL error (other than no more
*  rows) occurs, then DB2 diagnostic information is formatted
*  and sent to the partner.
*
EXEC SQL
   FETCH CURSOR1
   INTO :EMPNO, :FIRSTNME, :MIDINIT, :LASTNAME,
   :WORKDEPT, :PHONENO, :HIREDATE,
   :JOB, :EDLEVEL, :SEX, :BIRTHDATE, :SALARY,
   :BONUS, :COMM
END-EXEC.

*  Check query results and format error information if the SQL
*  code is any non-zero value other than +100, which indicates
*  ‘row not found’.
*
   IF SQLCODE NOT = +0 THEN
      MOVE ’Y’ TO TERM
MOVE SPACES TO EMPNO MIDINIT FIRSTNME LASTNAME WORKDEPT
   PHONENO HIREDATE JOB SEX BIRTHDATE
MOVE ZEROS TO FIRSTNME-LEN LASTNAME-LEN
   EDLEVEL SALARY BONUS COMM
IF SQLCODE NOT = +100 THEN
   MOVE 'ERROR ' TO EMPNO
   MOVE 'Y' TO DB2-DIAG
   PERFORM DB2I-SQL-ERROR
END-IF
END-IF.

* Send the row back to the partner application. This APPC send
* will satisfy the receive portion of a PGAXFER call to the
* Procedural Gateway for APPC issued by the partner application.

EXEC CICS SEND
   CONVID(WS-CONVID)
   STATE(WS-STATE)
   FROM(DCLEMP)
   LENGTH(LENGTH OF DCLEMP)
END-EXEC.

* Check for errors and abend if one is detected.

IF (WS-RESP NOT = DFHRESP(NORMAL) AND
   WS-RESP NOT = DFHRESP(EOC))
OR EIBERR = HIGH-VALUES THEN
   MOVE 'FAILURE SENDING EMP ROW' TO DEBUG-TEXT
   PERFORM ABEND
END-IF.

* Send DB2 diagnostic information if a DB2 error occurred.
* The partner application will issue another PGAXFER call to
* the Procedural Gateway for APPC to receive this data if the
* EMPNO field of the row returned was set to 'ERROR'. Note
* that the LAST option on this send causes the conversation
* to be deallocated immediately following the send.

IF DB2-DIAG = 'Y' THEN
   EXEC CICS SEND
      CONVID(WS-CONVID)
      STATE(WS-STATE)
      FROM(DB2-INFO)
      LENGTH(LENGTH OF DB2-INFO)
      LAST
* Check for errors and abend if one occurred.

    IF (WS-RESP NOT = DFHRESP(NORMAL) AND
        WS-RESP NOT = DFHRESP(EOC))
    OR EIBERR = HIGH-VALUES THEN
        MOVE 'FAILURE SENDING DB2 DIAGNOSTICS’ TO DEBUG-TEXT
        PERFORM ABEND
    END-IF
    END-IF.

EJECT

DB2I-SQL-ERROR.

*  This routine is invoked when any DB2 SQL error occurs to
  format the DB2 diagnostic information and write it to the
  PGADDB2I TS QUEUE. This information is also sent to the
  partner application (if possible) by the caller of this
  routine.

    MOVE SQLCODE TO DB2-SQLCODE.
    MOVE SQLERRM TO DB2-SQLERRM.

    EXEC CICS WRITEQ TS QUEUE('PGADDB2I')
        FROM(DB2-SQLERRM)
    END-EXEC.

*  Call the DB2 TIAR program to format the error information.

    CALL 'DSNTIAR' USING SQLCA DB2-DSNERRM DB2-DSNERRMT-LEN.

    IF RETURN-CODE = ZERO THEN
        PERFORM VARYING ERROR-INDEX
            FROM 1 BY 1 UNTIL ERROR-INDEX GREATER THAN 8
            EXEC CICS WRITEQ TS QUEUE('PGADDB2I')
                FROM(DB2-DSNERRMT(ERROR-INDEX))
        END-EXEC
    END-PERFORM
    END-IF.

EJECT
ABEND.

* This routine is called to write diagnostic information to
  * the PGADB2I TS queue, and abend the transaction with a DB2I
  * abend code.

EXEC CICS WRITEQ TS QUEUE('PGADB2I')
  FROM(DEBUG-TEXT)
END-EXEC.

MOVE WS-RESP TO DISP-RESP.
EXEC CICS WRITEQ TS QUEUE('PGADB2I')
  FROM(RESP-MSG)
END-EXEC.

MOVE WS-STATE TO DISP-STATE.
EXEC CICS WRITEQ TS QUEUE('PGADB2I')
  FROM(STATE-MSG)
END-EXEC.

EXEC CICS ABEND ABCODE('DB2I')
END-EXEC.

EXEC CICS RETURN
END-EXEC.
Sample pgacics.sql File

Rem
Rem Copyright (c) 1994 by Oracle Corporation. All rights reserved.
Rem
Rem NAME
Rem pgacics.sql - Procedural Gateway for APPC IVP
Rem DESCRIPTION
Rem This is a simple PL/SQL procedure to invoke the CICS FLIP transaction
Rem shipped as part of the gateway Installation Verification Procedure.
Rem The transaction takes a character string from 1 to 256 bytes in length,
Rem reverses it, and returns it to the caller.
Rem RETURNS
Rem Input message reversed
Rem NOTES
Rem The DBMS_OUTPUT package must be enabled in order to use this procedure.
Rem Also, the SQL*Plus SERVEROUTPUT variable must be set to ON in order for
Rem the output to be displayed.
Rem
Rem The database link name PGA should be replaced with the name of the
Rem database link to the gateway which you have defined to your integrating
Rem server.
Rem
Rem The tpname FLIP should be replaced with the name you have chosen for
Rem CICS transaction distributed as FLIP.
Rem
Rem The sidename CICSPGA should be replaced with the name of the SNA side
Rem information profile (defined on the gateway machine) that points to the
Rem CICS system into which the FLIP transaction has been installed. The
Rem profile should specify the partner LU name as the VTAM APPLID of the
Rem CICS system.
Rem
Rem The modename ORAPLU62 should be replaced with the mode name of the VTAM
Rem logmode entry you have defined for use with the gateway. This entry
Rem must be defined in the logmode table specified by the MODETAB parameter
Rem in the VTAM APPL definition for the CICS system.
Rem
Rem MODIFIED
Rem stalmod 01/20/2000 - created
Rem
CREATE or REPLACE PROCEDURE pgacics(msg VARCHAR2) AS
  tpname VARCHAR2(64) := 'FLIP';
  luname VARCHAR2(17) := '';
  modename VARCHAR2(8) := 'ORAPLU62';
  sidename VARCHAR2(8) := 'CICSPGA';
cid RAW(12) := HEXTORAW('000000000000000000000000');
sndbuf RAW(256);
sndbuf1 BINARY_INTEGER;
sndlns RAW(8);
rcvbuf RAW(256);
rcvbuf1 BINARY_INTEGER;
rcvlns RAW(8);
num2raw RAW(2048);
nlslang VARCHAR(50) := 'AMERICAN_AMERICA.WE8EBCDIC37C';
msgout VARCHAR2(256);
wngind BOOLEAN := FALSE;
wngblksz BINARY_INTEGER := 1;
wngblk RAW(1);
pad RAW(1) := HEXTORAW('00');
len NUMBER;
parms NUMBER;
numraw RAW(4) := HEXTORAW('00000000');

BEGIN

num2raw := UTL_PG.MAKE_NUMBER_TO_RAW_FORMAT('99999999',
'USAGE IS COMP',
'',
'IBMVSCOBOLII',
'',
nlslang,
wngind, wngblksz, wngblk);

PGAINIT@PGA(cid, tpname, luname, modename, sidename, '0');

parms := 1;
len := LENGTH(msg);

sndbuf := UTL_RAW.CAST_TO_RAW(msg);
sndbuf1 := len;
sndlns := HEXTORAW('000000000000000000000000');
umraw := UTL_PG.NUMBER_TO_RAW_FORMAT(parms,num2raw);
sndlns := UTL_RAW.OVERLAY(numraw,sndlns,1,4,pad);
umraw := UTL_PG.NUMBER_TO_RAW_FORMAT(len,num2raw);
sndlns := UTL_RAW.OVERLAY(numraw,sndlns,5,4,pad);
rcvbuf1 := 256;
rcvlns := UTL_RAW.CAST_TO_RAW(msg);
umraw := UTL_PG.NUMBER_TO_RAW_FORMAT(parms,num2raw);
rcvlns := UTL_RAW.OVERLAY(numraw,rcvlns,1,4,pad);
numraw := UTL_PG.NUMBER_TO_RAW_FORMAT(len, num2raw);
rcvlns := UTL_RAW.OVERLAY(numraw, rcvlns, 5, 4, pad);

PGAXFER@PGA(cid, sndbuf, sndbuf1, sndlns, rcvbuf, rcvbuf1, rcvlns);
msgout := UTL_RAW.CAST_TO_VARCHAR2(rcvbuf);
dbms_output.put_line(msgout);

PGATERM@PGA(cid, '0');

EXCEPTION
  WHEN OTHERS THEN
    PGATERM@PGA(cid, '1');
    RAISE;

END;
/
show errors

Rem Execute the procedure just created to send a message to the CICS FLIP
Rem transaction via the gateway and receive it back reversed. The expected
Rem result from this is:
Rem
Rem  ==> Congratulations, your gateway is communicating with CICS. <==
Rem
set serveroutput on

execute pgacics('==< .SCIC htiw gnitacinummoc si yawetag ruoy ,snoitalutargnoC
  >=');
You must convert datatypes and data formats properly when you are using the PGAU tool to generate TIPs and when you are developing a custom TIP using PL/SQL and the UTL_RAW and UTL_PG functions.

Read this appendix to learn about datatype conversion as it relates to TIPs.

This appendix contains the following sections:

- **Procedural Gateway for APPC Components** on page G-2
- **Length Checking** on page G-3
- **Conversion** on page G-4
Procedural Gateway for APPC Components

The Procedural Gateway for APPC product contains the following components:

- **PGA Server (termed the Gateway or Gateway Server)**
  This component is datatype independent, except data lengths cannot exceed 32K in a single transmission (refer to "PGAXFER" in Appendix C, "Gateway RPC Interface"). No datatype translations or reformatting is done by the Gateway. All such conversions are the responsibility of the application or the TIP which invokes the PGAXFER RPCs.

- **UTL_RAW PL/SQL package (the UTL_RAW functions)**
  This component is a series of data conversion functions for PL/SQL RAW variables and remote host data. The types of conversions performed depend on the language of the remote host data.

- **UTL_PG PL/SQL package (the UTL_PG functions)**
  This component is a series of COBOL numeric data conversion functions. Refer to "NUMBER_TO_RAW and RAW_TO_NUMBER Argument Values" in Appendix D, "The UTL_PG and UTL_RAW Interfaces" for supported numeric datatype conversions.

- **PG Data Dictionary (PG DD)**
  This component is a repository of remote host transaction and data definitions. The PGAU component accesses definitions in the PG DD when generating TIPs. The PG DD has datatype dependencies because it supports the PGAU and is not intended to be directly accessed by the customer. However, its data dependencies are effectively those imposed by the PGAU.

- **PG Administration Utility (PGAU)**
  This component generates TIPs which perform the conversion and reformatting of remote host data using PL/SQL and UTL_RAW/UTL_PG functions. Such conversions are dependent on the programming language of the remote host transaction and data, and the functions provided by PL/SQL and UTL_RAW/UTL_PG.
Length Checking

PGAU-generated TIPs perform length checking at the end of every parameter sent and received.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expected length</td>
<td>Is computed by PGAU when the TIP is generated.</td>
</tr>
<tr>
<td>convert length</td>
<td>Is summed by the TIP from each converted field.</td>
</tr>
<tr>
<td>send length</td>
<td>Is the transmitted send data length and is also equal to the actual length for send parameters.</td>
</tr>
<tr>
<td>receive length</td>
<td>Is the transmitted receive data length.</td>
</tr>
</tbody>
</table>

An exception is raised when the convert length of a sent parameter does not equal its expected length. This occurs if too many or too few send field conversions are performed.

An exception is raised when the convert length of a received parameter does not equal its received length. These length exceptions result when too few or too many conversions are performed.

A warning is issued when the expected length of a received parameter does not equal its convert or received length and data conversion tracing is enabled. This occurs when a maximum length record is expected, but a shorter record is transmitted and correctly converted.

Parameters Over 32K in Length

PGAU generates TIPs that support transmission of individual data parameters which exceed 32K bytes.

PGAU includes this support automatically when PGAU GENERATE processing detects the maximum length of a data parameter exceeding 32K.

This support is driven by the data definitions placed in the PG DD and cannot be selected by the user. To include the support, the data definition must actually or possible exceed 32K. To remove the support, you must decrease the parameter length to less than 32K, REDEFINE the data, and GENERATE the TIP again.

This support tests for field positions crossing the 32K buffer boundaries before and after conversion of those fields which lie across such boundaries. In the case of
repeating groups, This can be many fields, for repeating groups, or few fields in the case of simple linear records.

Each test and the corresponding buffer management logic adds overhead.

---

**Warning:** The target of a REDEFINE clause cannot reside in a previously processed buffer. Run-time TIP processing of the fields containing such REDEFINE clauses get unpredictable results.

---

**Conversion**

The PG DD and TIPs generated by PGAU support IBM VS COBOL II, specified as IBMVSCOBOLII when defining data.

**USAGE(PASS)**

When USAGE(PASS) has been specified on the PGAU DEFINE DATA statement, these datatype and format conversions are supported.

**Datatype Conversion**

PIC X

PGAU TIPs convert the COBOL X datatype to a PL/SQL CHAR datatype of the same character length. NLS character set translation is also performed.

Note: COBOL lacks a datatype specifically designated for variable length data. It is represented in COBOL as a subgroup containing a PIC 9 length field followed by a PIC X character field. For example:

10 NAME.

15 LENGTH PIC S9(4).

15 LETTERS PIC X(30).

Given this context, it cannot be guaranteed that all instances of an S9(4) field followed by an X field are always variable length data. Rather than PGAU TIPs converting the above COBOL group NAME to a VARCHAR, the TIPs instead construct a nested PL/SQL record as follows:

```plsql
TYPE NAME_typ is RECORD {
```
LENGTH NUMBER(4,0),
LETTERS CHAR(30));

TYPE ... is RECORD(
  ...
  NAME NAME_typ,
  ...

It is the client application’s responsibility (based upon specific knowledge of the remote host data) to extract NAME.LENGTH characters from NAME.LETTERS and assign the result to a PL/SQL VARCHAR, if a VARCHAR is desired.

Character set conversion is performed for single byte encoded:

- remote host character data, using either:
  - DEFINE TRANSACTION NLS_LANGUAGE character set for an entire transaction, or
  - REDEFINE DATA REMOTE_LANGUAGE character set for a single field, if specified.

- local Oracle character data, using either:
  - LANGUAGE character set of integrating server for an entire transaction, or
  - REDEFINE DATA LOCAL_LANGUAGE character set for a single field, if specified.

PIC G PGAU generated TIPs convert the COBOL G datatype to a PL/SQL VARCHAR2 datatype of the same length, allowing 2 bytes for every character position.

Character set conversion is performed for double-byte and multi-byte encoded:

- remote host character data, using either:
  - DEFINE TRANSACTION REMOTE_MBCS character set for an entire transaction, or
  - REDEFINE DATA REMOTE_LANGUAGE character set for a single field, if specified.

- local Oracle character data, using either:
- DEFINE TRANSACTION LOCAL_MBCS character set for an entire transaction, or
- REDEFINE DATA LOCAL_LANGUAGE character set for a single field, if specified.

Alphanumeric and DBCS Editing Field Positions
The following table illustrates how PGAU interprets COBOL symbols in datatype conversions.

<table>
<thead>
<tr>
<th>COBOL Symbol</th>
<th>Oracle Definition of COBOL Symbol s-Data Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>'B'</td>
<td>blank (1 byte SBCS or 2 bytes DBCS depending on USAGE)</td>
</tr>
<tr>
<td>'0'</td>
<td>zero (1 byte SBCS)</td>
</tr>
<tr>
<td>'/'</td>
<td>forward slash (1 byte SBCS)</td>
</tr>
<tr>
<td>'G'</td>
<td>double byte</td>
</tr>
</tbody>
</table>

Edited positions in COBOL statement data received from the remote host are converted by PGAU along with the entire field and passed to the client application in the corresponding PL/SQL VARCHAR2 output variable.

When editing symbols are present, they are interpreted to mean the remote host field contains the COBOL data content and length indicated. The editing positions are included in the length of the data field, but conversion of all field positions is processed by PGAU as a single string and no special scanning or translation is done for edited byte positions.

Edited positions in COBOL statement data sent to the remote host are converted by PGAU along with the entire PL/SQL VARCHAR2 input variable passed from the client application.

The following table provides an example:
Conversion

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC XXXBBXX</td>
<td>Is an alphanumeric field 7 bytes in length and would be converted in a single UTL_RAW.CONVERT call. No testing or translation is done on the contents of the byte positions indicated by 'B'. While VS COBOL II language rules indicate that these positions contain &quot;blank&quot; in the character set specified for the remote host, what data is actually present is the user’s responsibility.</td>
</tr>
<tr>
<td>PIC GGBGGG</td>
<td>Is a DBCS field 12 bytes in length and would be converted in a single UTL_RAW.CONVERT call. No testing or translation is done on the contents of the byte positions indicated by 'B'. While VS COBOL II language rules indicate that these positions contain &quot;blank&quot; in the character set specified for the remote host, what data is actually present is the user’s responsibility.</td>
</tr>
</tbody>
</table>
| PIC 9        | PGAU TIPs convert the COBOL 9 datatype to a PL/SQL NUMBER datatype of the same precision and scale. NLS character set translation is also performed on signs, currency symbols, and spaces. The following are supported:  
  - COMPUTATIONAL (binary)  
  - COMPUTATIONAL-3 (packed decimal)  
  - COMPUTATIONAL-4 (binary)  
  - DISPLAY (zoned decimal)  
For DISPLAY datatypes, the following sign specifications are supported:  
  - SEPARATE [CHARACTER]  
  - LEADING  
  - TRAILING  
Refer to "NUMBER_TO_RAW and RAW_TO_NUMBER Argument Values" in Appendix D for more information about numeric datatype conversions.  
COMPUTATIONAL-1 and COMPUTATIONAL-2 (floating point) datatypes are not supported. |
The following table describes format conversion:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JUSTIFIED</td>
<td>This causes remote host transaction data to be converted as a PL/SQL CHAR datatype according to character datatype, as discussed in &quot;Datatype Conversion&quot; on page 4, for both IN and OUT parameters.</td>
</tr>
<tr>
<td>JUSTIFIED RIGHT</td>
<td>IN parameter data passed from the application is stripped of its rightmost blanks and left padded as required. Then it is sent to the remote host.</td>
</tr>
<tr>
<td></td>
<td>OUT parameter data is aligned as it is received from the remote host and padded with blanks as required on the left. Then it is passed to the application.</td>
</tr>
<tr>
<td>JUSTIFIED LEFT</td>
<td>This causes warnings to be issued during TIP generation. No alignment is performed. This is treated as documentation.</td>
</tr>
<tr>
<td></td>
<td>The remote host transaction data is converted as a PL/SQL CHAR datatype according to character datatype, as discussed in &quot;Datatype Conversion&quot; on page 4, for both IN and OUT parameters.</td>
</tr>
</tbody>
</table>
This is an Oracle extension to the data definition as stored in the PG DD. This extension exists only in the PGAU context and is not valid IBM VS COBOL II syntax.

The purpose of this extension is to provide a means for variable-length character data to be processed more efficiently by the TIP conversion logic. This is an alternative to defining a variable-length PIC X field as PIC X(1) OCCURS DEPENDING ON field-2, where field-2 is the length of the field. With this extension, the same field could be defined as PIC X(5000) LENGTH IS field-2, where field -2 is the length of the field. The TIP is able to pick up the length and do the character set conversion on the field with a single UTL_RAW.CONVERT call instead of using a loop to do the conversion one character at a time.

Note that the use of this construct does not affect the COBOL program. The PIC X (or PIC G) field is still fixed-length as far as COBOL is concerned, so the position of the data does not change, nor does the amount of data that is transferred between the gateway and the OLTP. However, if the field is the last field in a COBOL definition, then the COBOL program could be modified to send only the number of bytes required to satisfy the length set in the field-2 field referenced by the LENGTH IS clause.

The LENGTH IS clause can be specified only for PIC X and PIC G fields, and the picture mask for those fields cannot contain editing characters.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LENGTH IS field-2</td>
<td>This is an Oracle extension to the data definition as stored in the PG DD. This extension exists only in the PGAU context and is not valid IBM VS COBOL II syntax. The purpose of this extension is to provide a means for variable-length character data to be processed more efficiently by the TIP conversion logic. This is an alternative to defining a variable-length PIC X field as PIC X(1) OCCURS DEPENDING ON field-2, where field-2 is the length of the field. With this extension, the same field could be defined as PIC X(5000) LENGTH IS field-2, where field -2 is the length of the field. The TIP is able to pick up the length and do the character set conversion on the field with a single UTL_RAW.CONVERT call instead of using a loop to do the conversion one character at a time. Note that the use of this construct does not affect the COBOL program. The PIC X (or PIC G) field is still fixed-length as far as COBOL is concerned, so the position of the data does not change, nor does the amount of data that is transferred between the gateway and the OLTP. However, if the field is the last field in a COBOL definition, then the COBOL program could be modified to send only the number of bytes required to satisfy the length set in the field-2 field referenced by the LENGTH IS clause. The LENGTH IS clause can be specified only for PIC X and PIC G fields, and the picture mask for those fields cannot contain editing characters.</td>
</tr>
</tbody>
</table>
OCCURS  n TIMES  This causes conversion of exactly ‘n’ instances of a set of PL/SQL variables to or from a repeating group area within the remote host record, the size of which area equals the group length times ‘n’ repetitions. PGAU generated TIPs employ PL/SQL RECORDs of TABLEs to implement an array-like subscript on fields within a repeating group. PL/SQL supports a single dimension TABLE, and consequently PGAU supports only a single level of an OCCURS group. Nested OCCURS groups are not supported. The conversion and formatting performed are dictated by the COBOL datatype of each subfield defined within the repeating group, as documented in "Datatype Conversion" on page G-4 and "Format Conversion" on page G-8.
### Datatype Conversion

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCCURS m TO n TIMES DEPENDING ON field-2</td>
<td>This causes conversion of at least ‘m’ and not over ‘n’ instances of a set of PL/SQL variables to or from a repeating group area within the remote host record, the size of which area equals the group length times the repetition count contained in the named field. PGAU generated TIPs employ PL/SQL RECORDs of TABLEs to implement an array-like subscript on fields within a repeating group. PL/SQL supports a single dimension TABLE, and consequently PGAU supports only a single level of an OCCURS DEPENDING ON group. Nested OCCURS DEPENDING ON groups are not supported. The conversion and formatting performed are dictated by the COBOL datatype of each subfield defined within the repeating group, as documented in “Datatype Conversion” on page G-4 and “Format Conversion” on page G-8. Range conversion: PGAU-generated TIPs use a ‘FOR ... LOOP’ algorithm with a range of 1 to whatever TIMES upper limit was specified. When the TIP has been generated with the DIAGNOSE(PKGEX(DC)) option, the PL/SQL FOR statement which iterates an OCCURS DEPENDING ON repeating group is preceded by an IF test to ensure at TIP runtime that the DEPENDING ON field contains a number which lies within the specified range for which the lower limit need not be 1. An exception is raised if this test fails.</td>
</tr>
<tr>
<td>RENAMES item-2 THRU item-3</td>
<td>A single PL/SQL variable declaration corresponds to a RENAMES definition. If all the subfields covered by a RENAMES definition are PIC X, then the PL/SQL variable is a VARCHAR2. Otherwise any non-PIC X subfield causes the PL/SQL variable datatype to be RAW. Lengths of renamed fields do not contribute to the overall parameter data length because the original fields dictate the lengths.</td>
</tr>
</tbody>
</table>
The 'WHEN item-3=value' is an Oracle extension to the data definition as stored in the PG DD. This extension exists only in the PGA context and is not valid IBM VS COBOL II syntax.

The purpose of this extension is to provide a means for the gateway administrator or application developer to specify the criteria by which the redefinition is to be applied. For example, a record type field is often present in a record and different record formats apply depending on which record type is being processed. The specification of which type value applies to which redefinition is typically buried in the transaction programming logic, not in the data definition. To specify which conversion to perform on redefined formats in the TIP, the WHEN criteria was added to PGA data definitions.

PGAU generates PL/SQL nested record declarations which correspond in name and datatype to the subordinate elements covered by the REDEFINES definition. The standard PGAU datatype determination described in "Datatype Conversion" on page G-4.

LEVEL 01 REDEFINE is ignored:

This permits remote host copybooks to include definitions which REDEFINE other transaction working storage buffers without having to define such buffers in the TIP or alter the copybook used as input for the definition.

This causes the numeric field to be aligned on boundaries as dictated by the remote host environment, compiler language, and datatype.

Numeric conversion is performed on the aligned data fields according to numeric datatype, as discussed in "Datatype Conversion" on page G-4, for both IN and OUT parameters.

<table>
<thead>
<tr>
<th>Item Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REDEFINES item-2 WHEN item-3=value</td>
</tr>
<tr>
<td>The purpose of this extension is to provide a means for the gateway administrator or application developer to specify the criteria by which the redefinition is to be applied. For example, a record type field is often present in a record and different record formats apply depending on which record type is being processed. The specification of which type value applies to which redefinition is typically buried in the transaction programming logic, not in the data definition. To specify which conversion to perform on redefined formats in the TIP, the WHEN criteria was added to PGA data definitions. PGAU generates PL/SQL nested record declarations which correspond in name and datatype to the subordinate elements covered by the REDEFINES definition. The standard PGAU datatype determination described in &quot;Datatype Conversion&quot; on page G-4. LEVEL 01 REDEFINE is ignored: This permits remote host copybooks to include definitions which REDEFINE other transaction working storage buffers without having to define such buffers in the TIP or alter the copybook used as input for the definition. This causes the numeric field to be aligned on boundaries as dictated by the remote host environment, compiler language, and datatype. Numeric conversion is performed on the aligned data fields according to numeric datatype, as discussed in &quot;Datatype Conversion&quot; on page G-4, for both IN and OUT parameters.</td>
</tr>
</tbody>
</table>
Usage

**Usage(ASIS)**

When USAGE(ASIS) is specified on the PGAU DEFINE DATA statement, no conversion is performed. Consequently, each such field is simply copied to a PL/SQL RAW of the same byte length. No conversion, translation, or reformatting is done.

**Usage(SKIP)**

When USAGE(SKIP) is specified on the PGAU DEFINE DATA statement, no data exchange is performed. The data is skipped as if it did not exist. Consequently, such fields are not selected from the PG DD, not reflected in the TIP logic, and presumed absent from the data streams exchanged with the remote host. The purpose of "SKIP" is to have definitions in the PG DD, but not active, perhaps because a remote host has either removed the field or has yet to include the field. SKIP allows an existing data definition to be used even though some fields do not exist at the remote host.

**Pl/sql Naming Algorithms**

**Delimiters**

COBOL special characters in record, group, and element names are translated when PGAU DEFINE inserts definitions into the PG DD, and by PGAU GENERATE when definitions are selected from the PG DD. Special characters are translated as follows:

- hyphen is translated to underscore (_)
- period is deleted

---

### Datatype Conversions

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNCHRONIZED</td>
<td>This causes warnings to be issued during TIP generation and no realignment is performed. This is treated as documentation.</td>
</tr>
<tr>
<td>LEFT</td>
<td>Numeric conversion is performed on the aligned data fields according to numeric datatype, as discussed in &quot;Datatype Conversion&quot; on page G-4, for both IN and OUT parameters.</td>
</tr>
</tbody>
</table>
Qualified Compound Names

PL/SQL variable names are fully qualified and composed from:

- PL/SQL record name as the leftmost qualifier corresponding to level 01 or 77 COBOL record name.
- PL/SQL nested record names corresponding to COBOL group names.
- PL/SQL nested fields corresponding to COBOL elements of datatype:
  - CHAR or NUMBER corresponding to non-repeating COBOL elements.
  - TABLE corresponding to COBOL elements which fall within an OCCURS or OCCURS DEPENDING ON group (COBOL repeating fields correspond to PL/SQL nested RECORDs of TABLE’s).

Note that when referencing PL/SQL variables from calling applications, the TIP package name must be prefixed as the leftmost qualifier. Thus the fully qualified reference to the PL/SQL variable which corresponds to:

- **SKILL** is:
  
  \[\text{tipname.EMPREC_Typ.SKILL(SKILL\_Key)}\]

- **HOME\_ADDRESS ZIP** is:
  
  \[\text{tipname.EMPREC_Typ.HOME\_ADDRESS.ZIP.FIRST\_FIVE}
  \text{tipname.EMPREC_Typ.HOME\_ADDRESS.ZIP.LAST\_FOUR}\]

Truncated and Non-Unique Names

PGAU truncates field names and corresponding PL/SQL variable names when the name exceeds:

- 26 bytes for fields within an aggregate record or group
  - This is due to the need to suffix each field or PL/SQL variable name with:
    - "_Typ" for group names
    - "_Tbl" for element names with a repeating group
  
  or

- 30 bytes due to the PL/SQL limitation of 30 bytes for any name
  - The rightmost four characters are truncated. This imposes the restriction that names be unique to 26 characters.
Duplicate Names

COBOL allows repetitive definition of the same group or element names within a record, and the context of the higher level groups serves to uniquely qualify names. However, because PGAU-generated TIPs declare PL/SQL record variables which reference nested PL/SQL records for subordinate groups and fields, such nested PL/SQL record types can have duplicate names. Given the following COBOL definition, note that ZIP is uniquely qualified in COBOL, but the corresponding PL/SQL declaration would have a duplicate nested record type for ZIP.

```cobol
01 EMPREC.
   05 HIREDATE   PIC X(8).
   05 BIRTHDATE  PIC X(8).
   05 SKILL      PIC X(12) OCCURS 4.
   05 EMPNO      PIC 9(4).
   05 EMPNAME.
      10 FIRST-NAME PIC X(10).
      10 LAST-NAME  PIC X(15).
   05 HOME-ADDRESS.
      10 STREET    PIC X(20).
      10 CITY      PIC X(15).
      10 STATE     PIC XX.
      10 ZIP.
         15 FIRST-FIVE PIC X(5).
         15 LAST-FOUR  PIC X(4).
   05 DEPT      PIC X(45).
   05 OFFICE-ADDRESS.
      10 STREET    PIC X(20).
      10 CITY      PIC X(15).
      10 STATE     PIC XX.
      10 ZIP.
         15 FIRST-FIVE PIC X(5).
         15 LAST-FOUR  PIC X(4).
   05 JOBTITLE   PIC X(20).
```

PGAU avoids declaring duplicate nested record types, and generates the following PL/SQL:

```sql
SKILL_Key BINARY_INTEGER;
TYPE SKILL_Tbl is TABLE of CHAR(12)
   INDEX by BINARY_INTEGER;
TYPE EMPNAME_Typ is RECORD (
   FIRST_NAME   CHAR(10),
   LAST_NAME    CHAR(15));
TYPE ZIP_Typ is RECORD (}
FIRST_FIVE   CHAR(5),
LAST_FOUR   CHAR(4));

TYPE HOME_ADDRESS_Typ is RECORD {
STREET       CHAR(20),
CITY         CHAR(15),
STATE        CHAR(2),
ZIP          ZIP_Typ);

TYPE OFFICE_ADDRESS_Typ is RECORD {
STREET       CHAR(20),
CITY         CHAR(15),
STATE        CHAR(2),
ZIP          ZIP_Typ);

TYPE EMPREC_Typ is RECORD {
HIREDATE     CHAR(8),
BIRTHDATE    CHAR(8),
SKILL        SKILL_Tbl,
EMPNO        NUMBER(4,0),
EMPNAME      EMPNAME_Typ,
HOME_ADDRESS  HOME_ADDRESS_Typ,
DEPT         CHAR(45),
OFFICE_ADDRESS OFFICE_ADDRESS_Typ,
JOBTITLE     CHAR(20));

However, in the case where multiple nested groups have the same name but have different subfields (see ZIP following):

05 HOME-ADDRESS.
   10 STREET       PIC X(20).
   10 CITY         PIC X(15).
   10 STATE        PIC XX.
   10 ZIP.
      15 LEFTMOST-FOUR PIC X(4).
      15 RIGHMOST-FIVE PIC X(5).
05 DEPT         PIC X(45).
05 OFFICE-ADDRESS.
   10 STREET       PIC X(20).
   10 CITY         PIC X(15).
   10 STATE        PIC XX.
   10 ZIP.
      15 FIRST-FIVE  PIC X(5).
      15 LAST-FOUR   PIC X(4).
05 JOBTITLE     PIC X(20).
PGAU alters the name of the PL/SQL nested record type for each declaration in which the subfields differ in name, datatype, or options. Note the "02" appended to the second declaration (ZIP_Typ02), and its reference in OFFICE_ADDRESS.

TYPE EMPNAME_Typ is RECORD (
    FIRST_NAME            CHAR(10),
    LAST_NAME                       CHAR(15));

TYPE ZIP_Typ is RECORD (
    LEFTMOST_FOUR         CHAR(4),
    RIGHTMOST_FIVE             CHAR(5));

TYPE HOME_ADDRESS_Typ is RECORD (
    STREET                CHAR(20),
    CITY                  CHAR(15),
    STATE                 CHAR(2),
    ZIP                                  ZIP_Typ);

TYPE ZIP_Typ02 is RECORD (
    FIRST_FIVE            CHAR(5),
    LAST_FOUR                       CHAR(4));

TYPE OFFICE_ADDRESS_Typ is RECORD (
    STREET                CHAR(20),
    CITY                  CHAR(15),
    STATE                 CHAR(2),
    ZIP                                  ZIP_Typ02);

TYPE EMPREC_Typ is RECORD (
    HIREDATE              CHAR(8),
    BIRTHDATE             CHAR(8),
    SKILL                 SKILL_Tbl,
    EMPNO                 NUMBER(4,0),
    EMPNAME               EMPNAME_Typ,
    HOME_ADDRESS          HOME_ADDRESS_Typ,
    DEPT                  CHAR(45),
    OFFICE_ADDRESS        OFFICE_ADDRESS_Typ,
    JOBTITLE               CHAR(20));

And the fully qualified reference to the PL/SQL variable which corresponds to:

- **HOME_ADDRESS.ZIP** is:
  
  tipname.EMPREC_Typ.HOME_ADDRESS.ZIP.LEFTMOST_FOUR
  tipname.EMPREC_Typ.HOME_ADDRESS.ZIP.RIGHTMOST_FIVE

- **OFFICE_ADDRESS.ZIP** is:
  
  tipname.EMPREC_Typ.OFFICE_ADDRESS.ZIP.FIRST_FIVE
  tipname.EMPREC_Typ.OFFICE_ADDRESS.ZIP.LAST_FOUR
Note that the nested record type name ZIP_Typ02 is not used in the reference, but is implicit within PL/SQL’s association of the nested records.
This appendix documents the National Language Support (NLS) information for the gateway. More information about using NLS is in the Oracle9i Server Application Developer’s Guide.

This appendix includes the following sections:

- **Overview** on page H-2
- **Languages Supported for Messages** on page H-2
- **Languages Supported for Data Conversion** on page H-3
Overview

National Language Support is a technology that enables Oracle applications to interact with users in their native language, using their conventions for displaying data.

The Oracle NLS architecture is data-driven, enabling support for specific languages and character encoding schemes to be added without requiring any changes in source code.

Languages Supported for Messages

The Oracle Procedural Gateway for APPC is capable of issuing its messages in languages other than English. The following table shows the currently supported languages in the left column, and the right column presents each language’s corresponding language, territory, and recommended character set specifications for use in the language setting. Other character sets can be used, but care must be taken to ensure that the character set used contains the full set of characters required by the specified language.

<table>
<thead>
<tr>
<th>Language</th>
<th>Setting for language_territory.charset</th>
</tr>
</thead>
<tbody>
<tr>
<td>American English</td>
<td>american_america.us7ascii</td>
</tr>
<tr>
<td>Brazilian Portuguese</td>
<td>&quot;brazilian portuguese&quot;_brazil.we8iso8859p1</td>
</tr>
<tr>
<td>Czech</td>
<td>czech_czechoslovakian.ee8iso8859p2</td>
</tr>
<tr>
<td>Danish</td>
<td>danish_denmark.we8iso8859p1</td>
</tr>
<tr>
<td>Dutch</td>
<td>dutch_&quot;the Netherlands&quot;.we8iso8859p1</td>
</tr>
<tr>
<td>Finnish</td>
<td>finnish_finland.we8iso8859p1</td>
</tr>
<tr>
<td>French</td>
<td>french_france.we8iso8859p1</td>
</tr>
<tr>
<td>German</td>
<td>german_germany.we8iso8859p1</td>
</tr>
<tr>
<td>Greek</td>
<td>greek_greece.el8iso8859p7</td>
</tr>
<tr>
<td>Hungarian</td>
<td>hungarian_hungary.ee8iso8859p2</td>
</tr>
<tr>
<td>Italian</td>
<td>italian_italy.we8iso8859p1</td>
</tr>
<tr>
<td>Japanese</td>
<td>japanese_japan.ja16euc</td>
</tr>
<tr>
<td>Korean</td>
<td>korean_korea.ko16ksc5601</td>
</tr>
<tr>
<td>Norwegian</td>
<td>norwegian_norway.we8iso8859p1</td>
</tr>
</tbody>
</table>
The language used by the Procedural Gateway for APPC server is specified by the LANGUAGE parameter in the initsid.ora file for the gateway. The syntax for the LANGUAGE parameter is:

```
LANGUAGE=language[_territory.charset]
```

where language, territory, and charset are valid values from the preceding table. If no LANGUAGE parameter is specified in the initsid.ora file, the default language used is American English.

The language used by the Procedural Gateway Administration Utility, PGAU, is specified by the NLS_LANG environment variable, in the same syntax as described for the LANGUAGE parameter for the server. If the NLS_LANG environment variable is not set, then the default language used is American English.

### Languages Supported for Data Conversion

The Oracle Procedural Gateway for APPC does no data conversion in the gateway itself. Instead, all conversion of data flowing between the integrating server and the gateway is performed by the Oracle integrating server. This is accomplished through calls within the TIP to the UTL_RAW.CONVERT function, which converts data from the local character set at the integrating server to the character set of the OLTP system.
The character set of the remote OLTP system is specified for:

- single-byte encoded data, using either:
  - PGAU DEFINE TRANSACTION NLS_LANGUAGE character set for an entire transaction
  - PGAU REDEFINE DATA REMOTE_LANGUAGE character set for a single field, if specified.

- double-byte and multi-byte encoded data, using either:
  - PGAU DEFINE TRANSACTION REMOTE_MBCS character set for an entire transaction, or
  - PGAU REDEFINE DATA REMOTE_LANGUAGE character set for a single field, if specified.

This information is generated into the TIP automatically by PGAU.

---

**Attention:** It is extremely important to ensure that the character set or codepage of the OLTP data be specified for PGAU using these parameters:

- DEFINE TRANSACTION ... NLS_LANGUAGE
- DEFINE TRANSACTION ... REMOTE_MBCS
- REDEFINE DATA ... REMOTE_LANGUAGE

Languages and character sets supported by the UTL_RAW.CONVERT function are the same as those supported by the Oracle integrating server. Refer to the platform-specific documentation for your Oracle integrating server for information on which languages and character sets are supported. If your integrating server is on an ASCII platform, then the platform-specific documentation might not list any of the EBCDIC character sets. This does not necessarily mean they are not supported. You should contact Oracle Support Services in this case.

---

**Attention:** Integrating servers at Version 7.2 and later support loadable character sets, and support all ASCII and EBCDIC character sets provided by Oracle7 and later.
### Index

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPC</td>
<td>CALL correspondence, 4-16</td>
</tr>
<tr>
<td>communications function, 1-12</td>
<td>CAST_TO_RAW function, D-5</td>
</tr>
<tr>
<td>facilitates gateway access to systems, 1-2</td>
<td>CAST_TO_VARCHAR2 function, D-6</td>
</tr>
<tr>
<td>overview, 1-2</td>
<td>client application development</td>
</tr>
<tr>
<td>runtime, 4-30</td>
<td>calling a TIP, 4-20</td>
</tr>
<tr>
<td>SENDs and RECEIVEs</td>
<td>customized TIPs for remote host</td>
</tr>
<tr>
<td>TIP CALL correspondence, 4-16</td>
<td>transaction, 4-8</td>
</tr>
<tr>
<td>trace, 6-15</td>
<td>declaring TIP variables, 4-20</td>
</tr>
<tr>
<td>using with terminal-oriented transaction program, 4-36</td>
<td>error handling, 4-27</td>
</tr>
<tr>
<td>APPC conversation sharing, 4-28</td>
<td>examples and samples, 1-5</td>
</tr>
<tr>
<td>concepts, 4-29</td>
<td>exchanging data, 4-27</td>
</tr>
<tr>
<td>examples, 4-32</td>
<td>executing, 4-28</td>
</tr>
<tr>
<td>for too large TIPs, 4-31</td>
<td>granting execute authority, 4-28</td>
</tr>
<tr>
<td>overrides and diagnostics, 4-35</td>
<td>overriding TIP initializations, 4-24</td>
</tr>
<tr>
<td>TIP compatibility, 4-30</td>
<td>overview, 4-2</td>
</tr>
<tr>
<td>usage, 4-30</td>
<td>preparation, 4-3</td>
</tr>
<tr>
<td>architecture</td>
<td>remote host transaction types</td>
</tr>
<tr>
<td>commit-confirm, 5-9</td>
<td>multi-conversational transactions, 4-7</td>
</tr>
<tr>
<td>three components of gateway architecture, 1-7</td>
<td>one-shot transactions, 4-5</td>
</tr>
<tr>
<td>ASCII</td>
<td>persistent transactions, 4-6</td>
</tr>
<tr>
<td>automatic conversion, 1-7</td>
<td>See also, index entries for each transaction type</td>
</tr>
<tr>
<td></td>
<td>requirements, 4-9</td>
</tr>
<tr>
<td>BIT_AND function, D-2</td>
<td>declare RHT/TIP data to be exchanged, 4-9,</td>
</tr>
<tr>
<td>BIT_COMPLEMENT function, D-3</td>
<td>4-10, 4-12</td>
</tr>
<tr>
<td>BIT_OR function, D-3</td>
<td>exchange data with RHT using TIP user</td>
</tr>
<tr>
<td>BIT_XOR function, D-4</td>
<td>function, 4-10, 4-13</td>
</tr>
<tr>
<td></td>
<td>initialize RHT for multi-conversational applications, 4-12</td>
</tr>
<tr>
<td></td>
<td>initialize RHT using TIP initialization function, 4-9, 4-11</td>
</tr>
<tr>
<td></td>
<td>repetitively exchange data with RHT using</td>
</tr>
</tbody>
</table>
TIP user function, 4-11
terminate RHT using TIP termination function, 4-13
terminate RHT using TIP termination function, 4-10, 4-12
security considerations, 4-26
terminating the conversation, 4-27
TIP and remote transaction program correspondence, 4-14
TIP CALL correspondence, 4-16
TIP content and purpose, 4-4
TIP DATA correspondence, 4-14
TIP TRANSACTION correspondence, 4-18
COBOL, 1-14, 4-3, G-4, G-13
source code, F-2
command termination, 2-7
COMMIT command, 2-3
user responsibility, 2-3
COMMIT processing, 2-3
commit-confirm, 4-8, 5-2
application design requirements, 5-8
architecture, 5-9
components, 5-9
interactions, 5-10
components required, 5-3
configuring, 5-5
gateway initialization parameters, 5-6
OLTP, 5-7
Oracle server, 5-5
flow, 5-10
logic flow, step by step, 5-10
Oracle Global Transaction ID, 5-2
purpose, 5-1
relation to two-phase commit, 5-2
required components
OLTP commit-confirm transaction log, 5-4
OLTP forget/recovery transaction, 5-4
OLTP transaction logging code, 5-4
Oracle logging server, 5-3
procedural gateway for APPC, 5-3
sample applications, 5-14
supported OLTPs, 5-2
transaction log, 5-13, 5-14
communication, between mainframe and Oracle integrating server, 1-11
COMPARE function, D-6
compiling a TIP, 3-8
components
of procedural gateway for APPC, G-2
CONCAT function, D-7
configuring
commit-confirm, 5-5
Oracle integrating server, 5-5
CONNECT command, 2-4, 2-8
control file
creating, 1-14
conversation sharing, see APPC conversation sharing
CONVERT function, D-8
COPIES function, D-9
creating a TIP
(detailed), 3-1
(overview), 1-13
D
data conversion
errors, 6-5
languages supported, H-3
DATA correspondence, 4-14
data dictionary, see PG DD
data exchange (PGAXFER function), 1-10
data format conversion, G-1
database link, 1-9
datatype
RAW, D-1
datatype conversion, G-1
See USAGE (PASS), USAGE (ASIS), USAGE (SKIP), and PL/SQL Naming Algorithms
convert length, G-3
duplicate names, G-15
expected length, G-3
format conversion, G-8
length checking, G-3
parameters over 32K in length, G-3
PL/SQL, G-14
naming algorithms, G-13
receive length, G-3
removing support for parameters over 32K in length, G-3

Index-2
send length, G-3
truncated and non-unique names, G-14

DBCS
See double-byte character sets
DBMS_PIPE PL/SQL package, 3-2
debugging tool
PGATRAC function, C-9
DEFINE CALL statement ("command"), 1-14, 3-5, 4-31, E-3
DEFINE DATA statement ("command"), 1-14, 2-40, 3-5, 4-14, 4-31, A-12, E-2
DEFINE TRANSACTION statement ("command"), 3-6, 4-31, E-3
defining and generating a TIP, 3-7
DESCRIBE command, 2-18
DISCONNECT command, 2-19
documentation required, xiii
double byte character sets (DBCS)
in application development, 4-35
driver procedure, 1-15

E

EBCDIC
automatic conversion, 1-7
errors
causes of, 6-2
data conversion, 6-5
including exception handlers in your TIP, 4-27
MAKE_NUMBER_TO_RAW_FORMAT function, D-30
MAKE_RAW_TO_NUMBER_FORMAT function, D-27
NUMBER_TO_RAW function, D-24
NUMBER_TO_RAW_FORMAT function, D-33
PLS -00123
program too large, 4-31
RAW_TO_NUMBER function, D-21
RAW_TO_NUMBER_FORMAT function, D-32
truncation, 6-6
WMSG function, D-36
WMSGCNT function, D-34
examples
APPC conversation sharing, 4-32
EXECUTE command, 2-20
executing
client application development, 4-28
EXIT command, 2-21

F

file
initsid.ora, 1-2, 5-6, 6-14, 6-15, H-3
pgacau.sql, 5-5
pgacclg.asm, 5-14
pgacics.sql, F-1
pgadb2c.cob, 5-14
pgadb2i.cob, F-2
pgadb2id.sql, 4-27
pgadb2i.pk, 1-5, 1-15, 4-2, 4-5
pgadb2i.pk, 1-5, 1-15, 4-5
pgadb2i.sql, B-1, B-12
pgareco.asm, 5-14
pgau.trc, 2-24, 2-28
pgddausr.sql, 4-38
pgdcr8.sql, 2-6, A-4
tipname.doc, 3-9, 4-14, 4-20
tipname.pk, 6-8
tname.ctl, 3-6, 3-7
tnsnames.ora, 5-6
format conversion, G-8
function
PGAINIT, 1-2, 1-10, B-11, B-19
PGATERM, 1-2, 1-10, B-19, C-8
PGAXFER, 1-2, 1-10, 4-5, C-5
PGAXFER, restrictions, B-9
UTL_PG, D-17
UTL_RAW, D-2
functions
see remote procedural call (RPC)
see UTL_PG
see UTL_RAW

gateway
creating a TIP, 3-1
features, 1-6
functions, 1-11
initialization file, 1-2

Index-3
language support,  H-2
overview,  1-2
remote procedure, definition,  1-2
server trace output sample,  6-16
starting,  1-9
terms used,  1-2
three components of gateway architecture,  1-7
tracing,  6-13
gateway server,  5-9
function in commit-confirm architecture,  5-9
transaction log tables,  5-13
gateway server trace,  6-13, 6-14
output sample,  6-16
GENERATE statement ("command"),  1-14, 3-6, 3-8,
  4-14, B-2, E-3
granting privileges for creating TIPs,  3-2
GROUP statement (PGAU),  3-8

H
HOST command,  2-31

I
IBM VS COBOL II
datatype conversion supported by PG DD and
TIPs,  G-4
support for double byte character sets, PIC G
datatypes,  4-35
implicit APPC,  4-37
implicit versioning
  sample definitions,  E-4
implied APPC,  5-3
initialization file
definition,  1-2
initiating remote transactions,  1-10
initsid.ora file,  1-2, 5-6, 6-14, 6-15, H-3

J
JUSTIFIED,  G-8
JUSTIFIED LEFT,  G-8
JUSTIFIED RIGHT,  G-8

K
keywords
  PGAU,  2-7

L
LANGUAGE parameter,  H-3
LENGTH function,  D-10
LENGTH IS field-2,  G-9

M
MAKE_NUMBER_TO_RAW_FORMAT
  function,  D-29
  errors,  D-30
MAKE_RAW_TO_NUMBER_FORMAT
  function,  D-26
  errors,  D-27
MBCS See multi-byte character sets
messages
  UTL_RAW interface,  D-2
multi-byte character sets (MBCS),  4-35
  application development support,  4-35
multi-conversational transactions,  1-11, 4-7, 4-12

N
National Language Support (NLS),  3-4, H-1
  in the gateway,  H-2
  languages supported for data conversion,  H-3
  multi-byte character set support,  4-35
  on various platforms,  H-4
NLS LANGUAGE parameter
  syntax,  H-3
NUMBER_TO_RAW and RAW_TO_NUMBER
  argument values,  D-37
NUMBER_TO_RAW function,  D-23
  errors,  D-24
NUMBER_TO_RAW_FORMAT function,  D-33
  errors,  D-33

O
OCCURS DEPENDING ON,  G-11
OCCURS n TIMES,  G-10
OLTP
See online transaction processor
one-shot transactions, 1-11, 4-5, 4-9, 5-8
online transaction processor (OLTP)
character set, H-4
commit-confirm transaction log, 5-4
configuring, 5-7
definition, 1-3
forget/recovery transaction, 5-4
function in the gateway, 1-9
functional requirements of the gateway, 5-8
in commit-confirm, 5-2
logic flow for successful commit, 5-10
parameters needed for NLS, H-4
remote, 1-1
security considerations, 4-26
transaction logging code, 5-4
Oracle global transaction ID, 5-2, 5-4, 5-8, 5-13, 5-14
Oracle integrating server, 1-7, 1-9, 1-10, 4-2
and OLTP, 1-8
architecture, 1-7
calling RPC functions, 5-10
component of commit-confirm architecture, 5-9
configuring for commit-confirm, 5-5
definition, 1-3
interaction with gateway server, 5-10
multiple servers, 1-8
role in the gateway, 1-11
steps to communication between server and
mainframe, 1-12
stores PL/SQL, 1-4
support for NLS, H-4
Oracle logging server, 5-3, 5-9, 5-10
description, 5-9
interaction with gateway server, 5-10
Oracle Net, 1-3, 4-38, 5-6, D-1
function, 1-7
restrictions for data conversion, D-1
supported protocols, 1-8
Oracle Procedural Gateway for APPC
compatibility with version 3.4.0, 4-30
development environment, 1-7
features, 1-6
functions, 1-11
overview, 1-2
See also, gateway server
OVERLAY function, D-10
overrides, 4-25
LOGMODE, 4-24
LUname, 4-24
Side profile, 4-24
TPName, 4-24
package
UTL_PG, 3-2
parameters
see NLS LANGUAGE
see PGAU commands
see remote procedural call (RPC)
see SET LOG_DESTINATION
see SET TRACE_LEVEL
persistent transactions, 1-11, 4-6, 4-10, 5-8
PG DD (Data Dictionary), 2-2, 2-13
active dictionary, A-4
sequence numbers, A-5
versioning, A-4
active dictionary tables
pga_call, A-10
pga_call_parm, A-11
pga_data, A-12
pga_data_attr, A-16
pga_data_values, A-17
pga_fields, A-14
pga_trans, A-6
pga_trans_attr, A-6
pga_trans_calls, A-9
pga_trans_values, A-8
data definitions for parameters over 32K in
length, G-3
datatype conversion support for IBM VS COBOL
II, G-4
definition, 1-3
definition names
valid characters in, 2-5
diagnostic
options, 6-3
references, 6-3
tables, creating a TIP, 3-7
Index-5
environment dictionary, A-2  
environment dictionary tables, A-3  
pga_maint, A-3  
pga_modes, A-4  
pga_usage, A-4  
in writing PGAU statements, 3-5  
keyword form in storage, 2-7  
maintenance, 2-4  
overview, A-2  
preparing client application, 4-3  
purpose of REPORT command, 2-39  
relationship to PGAU, 2-2  
remote transaction definitions, 2-4  
ROLLBACK command, 2-3  
select scripts, 6-4  
storage of information needed for PGAU  
GENERATE to perform, 4-14  
transaction attributes, 4-25  
USAGE (SKIP), G-13  
version definition tables, 2-6  
PGA  
administrator, 2-2  
definition, 1-3  
PAG Server (also called, Gateway or Gateway  
Server), G-2  
pga_call table, A-10  
pga_call_parm table, A-11  
PAG_CC_PENDING table  
commit-confirm transaction log, 5-13  
pga_data table, A-12  
pga_data_attr, A-16  
pga_data_values table, A-17  
pga_fields table, A-14  
pga_maint constant, A-3  
pga_modes constant, A-4  
pga_trans table, A-6  
pga_trans_attr table, A-6  
pga_trans_calls table, A-9  
pga_trans_values table, A-8  
pga_usage, A-4  
ppga_useage constant, A-4  
pgccau.sql file, 5-5  
pgcclg.asm file, 5-14  
pgccics.sql file  
sample program, F-1  
pgadb2c.cob file, 5-14  
pgcd2i, 1-15  
pgcd2i.cob file, F-2  
pgcd2id.sql file, 4-5, 4-27  
pgcd2i.pkb, 1-5  
pgcd2i.pkb file, 1-15, 4-2, 4-5  
pgcd2i.pk file, 1-5, 1-15  
pgcd2i.sql file, 4-5, B-1, B-12  
PAGINIT, B-11, B-19, C-2  
PAGINIT function, 1-2, 1-10  
pagareco.asm file, 5-14  
PAGATCTL, C-8  
PAGTERM, B-19, C-8  
PAGTERM function, 1-2, 1-10  
PAGATRAC, C-9  
PAGU, 4-2  
commands- also called “statements”, 2-7  
COMMIT processing, 2-3  
defining and testing a TIP, 2-5  
definition, 2-1  
used to generate TIPS, 1-3  
definition names, 2-5  
definition versioning, 2-6  
definitions, 2-4  
functions, 2-2  
generate TIP, 1-9  
generation, 2-4  
ininvoking, 2-3, 2-4  
keywords, 2-7  
NLS Language usage, H-3  
overview, 2-2  
ROLLBACK processing, 2-3  
sample input, E-1  
terminating a statement, 2-7  
writing statements, 3-5  
PAGU commands, 1-14, 2-7  
CONNECT, 2-4, 2-8  
CONNECT, parameters, 2-8  
DEFINE CALL, 1-14, 2-2, 2-9, 2-41, 3-5, 4-31  
call list, 3-3  
parameters, 2-9  
sample, E-3  
DEFINE DATA, 1-14, 2-2, 2-10, 2-40, 3-5, 4-14,  
4-31, A-12  
parameters, 2-10
qualified compound names, G-14
parameter restrictions, B-10
parameters, 4-20, B-7
record, B-13, B-16
record format, E-6
record types, B-11
simple procedure, F-1
stored procedure, 1-12
transferring data
using RAW datatype, D-1
variable names, G-14
datatype conversion, G-14
variables, 3-9, G-10, G-11, G-14
PL/SQL package, 2-4, 2-9, 2-48, 2-50, 3-2, 4-5, 6-3, B-2, C-1
See TIP components, 4-4
contents
package specification, 4-4
DBMS_PIPE, 3-2
definition, 1-4
execute authority, 4-28
function, 1-9, 4-2
grants required, 3-2
loading the tip, B-19
paccics, 1-12
parameter, 2-26
specifying names, 4-19
stored procedure specification, 1-6
PL/SQL stored procedure, 5-5, 5-10
changing trace level, C-8
starting up communication with mainframe, 1-12
used for logging transactions, 5-5
PL/SQL stored procedure specification
See PL/SQL package
PRINT command, 2-32
privileges
needed to use TIPs, 4-37
problem analysis
of data conversion and truncation errors, 6-6
with PG DD diagnostic references, 6-3
with PG DD select scripts, 6-4
with TIP runtime traces, 6-8
procedural gateway administration (PGA), 1-3

Procedural Gateway Administration Utility - See
PGAU, 1-9
procedural gateway for APPC
components, G-2
See Also, Oracle Procedural Gateway for APPC
See gateway server, G-2
product publications, xiii

R

RAW, B-14, D-7
RAW_TO_NUMBER FORMAT function, D-32
RAW_TO_NUMBER function, D-19
errors, D-21
RAW_TO_NUMBER_FORMAT function errors, D-32
recompilation errors
causes, B-5
REDEFINE DATA statement, A-12, E-8
REDEFINES, G-12
REM command, 2-38
remote host transactions (RHT)
APPC conversation sharing, 4-28
attributes needed, 4-19
client application, 4-10
defined using the PGAU DEFINE
TRANSACTION statement, 4-18
evaluating, 3-2
multi-conversational, client applications, 4-12
one-shot, client applications, 4-9
persistent, client applications, 4-10
requirements
understanding, 4-4
steps involved in, 4-9
types, 1-11, 4-5
remote procedural call (RPC), A-10
calling the gateway, C-1
definition, 1-4
executing gateway functions, C-2
functions, 1-9
in TIP development, B-11
parameters, C-8
PGAINIT, B-19
PGAINIT and PGAINIT_SEC, C-2
PGAINIT and PGAINIT_SEC, parameters, C-2
PGATCTL, C-8
PGATERM, B-19, C-8
PGATERM, parameters, C-8
PGATRAC, C-9
PGATRAC, parameters, C-9
PGAXFER, C-5
    restrictions, B-9, B-10
PGAXFER, parameters, C-5
processing, 1-1
remote procedure
definition, 1-2
remote transaction program (RTP), 1-3, 4-8
    activities, 4-7
definition, 1-4
    in TIP development, B-11
purpose, 4-4
    restrictions, B-10
    synchronized APPC SEND and RECEIVE in
    TIPs, B-7
remote transaction termination
    PGATERM, 1-10
RENAMEs, G-11
REPORT statement, 3-6
restriction
gateway non-support of two-phase commit, 4-8
restrictions in Oracle Procedural Gateway for
    APPC, B-9
REVERSE function, D-12
RHT, See remote host transactions
ROLLBACK command, 2-3
ROLLBACK processing, 2-3
RPC interface
    PGAINIT, C-2
PGATCTL, C-8
PGATERM, C-8
PGATRAC, C-9
PGAXFER, C-5
See also, remote procedural call (RPC)
RPC, See remote procedural call
RTP, See remote transaction program
runtime traces, 6-8
    controls, 6-10
    conversion warnings, 6-11
    data conversion tracing, 6-11
gateway exchange tracing, 6-12
    runtime function entry/exit tracing, 6-11
S
sample
    applications, 5-14
    CICS APPC transaction, F-2
    PGAU DEFINE CALL command, E-3
    PGAU DEFINE DATA command, E-2
    PGAU DEFINE TRANSACTION command, E-3
    PGAU GENERATE command, E-3
    PGAU REDEFINE DATA command, E-8
    PGAU REPORT output, E-9
    PGAU UNDEFINE command, E-9
    TIP output, E-21
    TIP trace output, E-19
    sample definitions
        implicit versioning, E-4
    script file, 3-6
    SET command, 2-42
    SET LOG_DESTINATION parameter, 6-7, 6-8, 6-14
    SET TRACE_LEVEL parameter, 6-7, 6-14, 6-15,
        C-8, C-9
    Side Information Profile, 2-16, C-5
    simple PG4APPC communication, 1-11
    SNA APPC (Advanced Program-to-Program
        Communication), 1-2
    specification file, 1-15
    SPOOL command, 2-47
SQL*Plus
    connecting server and mainframe, 1-12
    invoking, 3-8
    recompiling TIP body changes, B-3
    running scripts, 4-37
test scripts, 6-5
    use in configuring Oracle integrating server, 5-5
starting the gateway, 1-9
    statements
        see PGAU commands
    storage measurements, xvi
    SUBSTR function, D-12
    supported languages, H-2
    SYNCHRONIZED, G-12
    SYNCHRONIZED LEFT, G-13
    SYNCHRONIZED RIGHT, G-12
terminal-oriented transactions
modifying, 4-36
terminating a TIP conversation, 4-27
terms, gateway terms defined, 1-2
TIP, 4-2
APPC conversation sharing, 4-29, 4-30
architecture, B-10, B-11, B-18
background references, B-2
body, B-11
CALL correspondence, 4-16
call correspondence order restrictions, 4-18
call, to initiate and terminate connection to RTP, B-5
calling
and controlling, 4-2
from the client application, 4-20
client application development
content and purpose, 4-4
compiling, 3-8
content documentation (tipname.doc), 3-9
content file sections
GENERATION Status, 3-9
TIP Default Calls, 3-9
TIP Transaction, 3-9
TIP User Calls, 3-9
TIP User Declarations, 3-9
TIP User Variables, 3-9
control file, 2-2
controlling runtime conversion warnings, 6-11
controlling runtime data conversion
tracing, 6-11
controlling runtime function tracing, 6-11
controlling runtime gateway exchange
tracing, 6-12
conversation sharing used to circumvent large TIPs, 4-31
conversion, 4-35
converting PL/SQL datatypes to RAW, 1-10
creating, 3-1
custom TIP writing, B-1
customized interface for each remote host transaction (RTP), 4-8
DATA correspondence, 4-14
datatype conversion support for IBM VS COBOL II, G-4
declaring variables to create a TIP, 4-20
defining and generating, 3-7
definition, 1-4
definition errors, 6-2
dependent TIP body or specification changes, B-4
diagnostic parameters, 4-35
driver procedures, 1-15
flexible call sequence, 4-17
four steps to generate, 1-13
functions, 1-9
generated by PGAU, 4-2
granting privileges to use, 3-2, 4-37
independent TIP body changes, B-3
initializations, 4-25, B-18
overriding, 4-24
initializing the conversation, 4-22
internals, B-1
operation, B-19
output, sample, E-21
override parameters, 4-35
overriding default attributes, 4-25
overview, 1-13
parameter length checking, G-3
privileges needed, 3-2, 4-37
programming, B-10, B-11, B-18
public functions
tip_init, 4-6
tip_inqr, 4-6
tip_mode, 4-6
tip_term, 4-6
tip_updt, 4-6
recompiling, B-3, B-4
remote transaction
 correspondence, 4-14
remote transaction initiation (PGAINIT), 1-10
requirements for corresponding with RHT, 4-14
requirements of the client application, 4-9
service, 4-29
specification file, 3-7
synchronizing SEND/RECEIVE, B-7
terminate function, B-20
terminating the conversation, 4-27
trace controls, 6-10
trace output sample, E-19
tracing, 6-7
TRANSACTION correspondence, 4-18
usage, B-19
TIP control file commands, 1-14
TIP specification, 1-9, 4-2, B-2, B-3, B-10
changes, B-4
errors, B-5
file, 1-15
TIP warnings and tracing
suppressing, 6-12
tipname.doc file, 3-9, 4-14, 4-20
tipname.pkb file, 6-8
tname.ctl file, 3-6, 3-7
tnsnames.ora file, 5-6
trace option, 6-1
TIP definition errors, 6-2
TRACE_LEVEL, 6-14
traces, 6-11, 6-12
diagnostic, 6-7
enable gateway server trace, 6-14
enabling APPC trace from PL/SQL, 6-15
gateway server, 6-13
purpose of initializing conversations, 4-22
runtime, 6-8
trace controls, 6-10
sample gateway server output, 6-16
suppressing, 6-12
TIP, 6-8, 6-12
TRANSACTION correspondence, 4-18
Transaction Interface Package
See TIP
TRANSLATE function, D-13
TRANSLITERATE function, D-14
transparency, 1-6
two-phase commit, 4-8
platform support for, 2-16
supported for AIX, 4-7
transactional support, 1-6

U
UNDEFINE statement, 3-6, E-9
USAGE(ASIS), G-13
USAGE(PASS), G-4
datatype conversion, G-4
FILLER, G-8
PIC 9, G-7
PIC G, G-5
PIC X, G-4
format conversion
OCCURS DEPENDING ON, G-11
OCCURS n TIMES, G-10
USAGE(SKIP), G-13
UTL_PG
input parameters
wmsgbbsiz, D-19
output parameters
wmsgblk, D-19
parameters (input and output), D-17
PL/SQL package, 3-2
UTL_PG function, B-18, D-17
MAKE_NUMBER_TO_RAW_FORMAT, D-29
MAKE_RAW_TO_NUMBER_FORMAT, D-26
NUMBER_TO_RAW, D-23
NUMBER_TO_RAW and RAW_TO_NUMBER
argument values, D-37
NUMBER_TO_RAW_FORMAT, D-33
RAW_TO_NUMBER, D-19
RAW_TO_NUMBER_FORMAT, D-32
WMSG, D-35
WMSGCNT, D-34
UTL_PG input parameters
compname, D-18
compopts, D-18
envrmnt, D-18
mask, D-18
maskopts, D-18
nlslang, D-18
wind, D-18
UTL_PG PL/SQL package
definition, 1-4
description, G-2
UTL_RAW function
BIT_AND, D-2
BIT_COMPLEMENT, D-3
BIT_OR, D-3
BIT_XOR, D-4
CAST_TO_RAW, D-5
CAST_TO_VARCHAR2, D-6
COMPARE, D-6
CONCAT, D-7
CONVERT, D-8
COPIES, D-9
LENGTH, D-10
OVERLAY, D-10
REVERSE, D-12
SUBSTR, D-12
TRANSLATE, D-13
TRANSLITERATE, D-14
XRANGE, D-16
UTL_RAW PL/SQL package, 3-2, D-1
definition, 1-4
description, G-2
functions, D-2

V
VARIABLE command, 2-51

W
WMSG function, D-35
ersors, D-36
WMSCNT function, D-34
WMSCNTfunction
ersors, D-34
writing PGAU statements, 3-5

X
XRANGE function, D-16