Contents

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

Preface ........................................................................................................................................ xiii

1 Introduction
   Introduction to Message Queuing ............................................................................................ 1-2
   Introduction to IBM MQSeries ............................................................................................... 1-2
   MQSeries Terms ..................................................................................................................... 1-3
   Introduction to the Gateway ................................................................................................. 1-4
   Developing Gateway Applications ......................................................................................... 1-4
   Gateway Terms ...................................................................................................................... 1-5
   Advantages of Using the Gateway ......................................................................................... 1-6
   Gateway Architecture ........................................................................................................... 1-7
   Component Descriptions ....................................................................................................... 1-8
   Gateway Structure ................................................................................................................ 1-9
   Starting the Gateway ............................................................................................................ 1-9
   Communication ....................................................................................................................... 1-9

2 Release Information
   The Product Set ....................................................................................................................... 2-2
   Changes and Enhancements ................................................................................................. 2-2
   Changes and Enhancements for Release 9.2.0.1.0 ............................................................. 2-2
   Changes and Enhancements for Release 9.0.1.0.1 ............................................................. 2-3
   Known Problems for Release 9.2.0.1.0 .............................................................................. 2-6
### Known Restrictions for Release 9.2.0.1.0

- 2-6

### 3 Requirements

- Hardware Requirements ................................................................. 3-2
- Software Requirements ................................................................. 3-3
- Oracle Integrating Server ............................................................. 3-4
- Recommended Documentation ...................................................... 3-4

### 4 Pre-installation

- Preinstallation Tasks ..................................................................... 4-2
  - MQSeries Software .................................................................... 4-2
  - Setting Environment Variables ..................................................... 4-3
- About Oracle Universal Installer .................................................. 4-7
  - oraInventory Directory .............................................................. 4-8
  - Starting Oracle Universal Installer ............................................. 4-9

### 5 Installation

- Installation .................................................................................... 5-2
- Running root.sh ........................................................................... 5-13

### 6 Deinstallation and Reinstallation

- Deinstallation ............................................................................... 6-2
  - Deinstalling Using Oracle Universal Installer .......................... 6-2
  - Deinstalling Oracle9i Procedural Gateway for IBM MQSeries ...... 6-2
- Reinstallation ................................................................................. 6-5

### 7 Configuration

- Configuration Overview .............................................................. 7-2
- Configuring the Gateway .............................................................. 7-3
  - Using the Gateway with the Default Values ......................... 7-3
  - Using the Gateway without the Default Values .................. 7-3
  - Changing Default Values ....................................................... 7-4
- Configuring Oracle Net for the Gateway .................................... 7-10
Using Oracle Net with Gateway Default Values ................................................................. 7-10
Using Oracle Net When Changing the Gateway Default Values........................................ 7-10

**Configuring Oracle Net for the Oracle Integrating Server** ............................................ 7-16
Using Gateway Default Values ......................................................................................... 7-16
Changing Gateway Default Values .................................................................................. 7-16

Creating a Transaction Log Queue .................................................................................... 7-19

**Administering Database Links** .................................................................................... 7-19
Creating Database Links .................................................................................................. 7-20
Dropping Database Links ................................................................................................. 7-22
Examining Available Database Links ............................................................................. 7-22
Limiting the Number of Active Database Links ............................................................... 7-22

**Installing the Visual Workbench Repository** ............................................................... 7-22
Preinstallation Tasks ........................................................................................................ 7-22
Visual Workbench Repository Installation Tasks ............................................................ 7-24
After the Repository is Created ...................................................................................... 7-26
Deinstall the Visual Workbench Repository .................................................................. 7-27

**Preparing the Production Oracle Server** ................................................................... 7-28
Introduction ...................................................................................................................... 7-28
Verifying and Installing PL/SQL Packages ................................................................... 7-29
Removing the PL/SQL Packages ................................................................................???? 7-30

---

### 8 Gateway Running Environment

**Security Models** ........................................................................................................... 8-2
Relaxed Model ................................................................................................................... 8-2
Strict Model ....................................................................................................................... 8-3
Authorization for MQSeries Objects .............................................................................. 8-4

**Transaction Support** .................................................................................................. 8-5
Non-Oracle Data Sources and Distributed Transactions ................................................. 8-5
Transaction Capability Types .......................................................................................... 8-6
Transaction Capability Types of Procedural Gateway for IBM MQSeries ....................... 8-7

**Troubleshooting** .......................................................................................................... 8-9
Message and Error Code Processing .............................................................................. 8-9
Common Error Codes ...................................................................................................... 8-11
Gateway Tracing ................................................................................................................ 8-15
Verifying Gateway Operation .......................................................................................... 8-17
## 9 Heterogeneous Message Propagation

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heterogeneous Message Propagation Option</td>
<td>9-2</td>
</tr>
<tr>
<td>Oracle Advanced Queuing (AQ)</td>
<td>9-2</td>
</tr>
<tr>
<td>Operational and Administrative Interfaces</td>
<td>9-2</td>
</tr>
<tr>
<td>Outbound Interfaces</td>
<td>9-2</td>
</tr>
<tr>
<td>Inbound Interfaces</td>
<td>9-3</td>
</tr>
<tr>
<td>Heterogeneous Outbound Propagation</td>
<td>9-3</td>
</tr>
<tr>
<td>Outbound Propagation in General</td>
<td>9-3</td>
</tr>
<tr>
<td>Heterogeneous Inbound Propagation</td>
<td>9-5</td>
</tr>
<tr>
<td>Inbound Propagation in General</td>
<td>9-6</td>
</tr>
</tbody>
</table>

## 10 Installing Heterogeneous Message Propagation

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle Software Requirement</td>
<td>10-2</td>
</tr>
<tr>
<td>Installing the Heterogeneous Message Propagation Option</td>
<td>10-2</td>
</tr>
<tr>
<td>PL/SQL Installation Scripts</td>
<td>10-2</td>
</tr>
<tr>
<td>Installation</td>
<td>10-2</td>
</tr>
<tr>
<td>Privileges and Access Control</td>
<td>10-3</td>
</tr>
<tr>
<td>Verifying the Heterogeneous Message Propagation Option</td>
<td>10-4</td>
</tr>
<tr>
<td>Deinstalling the Heterogeneous Message Propagation Option</td>
<td>10-5</td>
</tr>
</tbody>
</table>

## 11 Configuring Heterogeneous Outbound Message Propagation

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview of Outbound Message Propagation</td>
<td>11-2</td>
</tr>
<tr>
<td>Translation of Oracle AQ/IBM MQSeries Interfaces</td>
<td>11-2</td>
</tr>
<tr>
<td>Translation for Outbound Propagation for MQSeries</td>
<td>11-3</td>
</tr>
<tr>
<td>Configuring Outbound Message Propagation</td>
<td>11-8</td>
</tr>
<tr>
<td>Operational Interface for Individual Messages</td>
<td>11-9</td>
</tr>
<tr>
<td>Operational Interface for Queues</td>
<td>11-14</td>
</tr>
<tr>
<td>Oracle AQ Procedures and Structures</td>
<td>11-15</td>
</tr>
<tr>
<td>Administrative Interfaces</td>
<td>11-17</td>
</tr>
<tr>
<td>Procedures for Outbound Message Propagation</td>
<td>11-17</td>
</tr>
<tr>
<td>Example</td>
<td>11-22</td>
</tr>
<tr>
<td>Security</td>
<td>11-23</td>
</tr>
<tr>
<td>Exception Handling</td>
<td>11-24</td>
</tr>
<tr>
<td>Schedule Status</td>
<td>11-25</td>
</tr>
</tbody>
</table>
12 Configuring Heterogeneous Inbound Message Propagation

Overview of Inbound Message Propagation From MQSeries.............................................. 12-2
Translation of MQSeries/Oracle AQ Interfaces........................................................................ 12-2
  Translations for inbound propagation for MQSeries:.................................................. 12-3
Configuring Inbound Message Propagation........................................................................ 12-6
Operational Interface ........................................................................................................... 12-8
Administrative Interface ........................................................................................................ 12-11
  Procedures for Inbound Message Propagation......................................................... 12-12
  Example.................................................................................................................. 12-17
Security ............................................................................................................................... 12-18
Exception Handling ............................................................................................................. 12-19
  propagation exception queue ...................................................................................... 12-19
Schedule Status .................................................................................................................. 12-19
  Status of an Inbound Message Propagation Schedule ........................................... 12-19

A The PGM8, PGM_UTL8, and PGM_SUP Packages

PGM8 Package and PG4MQ Gateway Procedures .......................................................... A-2
  Summary of Procedures and Type Definitions.......................................................... A-2
  Procedure Conventions .............................................................................................. A-3
  MQI Calls Performed by the Gateway ...................................................................... A-4
  Unsupported MQI Calls ............................................................................................. A-4
  Migration Tips ............................................................................................................ A-5
  MQCLOSE Procedure .................................................................................................. A-8
  MQGET Procedure ...................................................................................................... A-9
    PGM8.MQMD Type Definition............................................................................... A-14
    PGM8.MQGMO Type Definition.......................................................................... A-19
  MQOPEN Procedure ..................................................................................................... A-20
    PGM8.MQOD Type Definition.............................................................................. A-22
  MQPUT Procedure ........................................................................................................ A-24
    PGM8.MQPMO Type Definition............................................................................ A-27
PGM_SUP Package ............................................................................................................. A-29
  PGM8.MQGMO Values............................................................................................... A-29
  PGM8.MQMD Values................................................................................................. A-30
### B  UTL_RAW Package

**Message Data Types** 

- PGM8.MQOD Values ................................................................. A-35
- PGM8.MQPMO Values .............................................................. A-36
- MQCLOSE Values ................................................................. A-36
- MQOPEN Values ................................................................. A-37
- Maximum Lengths for Fields of PGM8 Type Definitions ........ A-37
- Error Code Definitions ......................................................... A-39

**UTL_RAW Functions**

- UTL_RAW.TO_RAW ................................................................. B-2
- UTL_RAW.BIT_AND ............................................................... B-4
- UTL_RAW.BIT_COMPLEMENT ............................................... B-5
- UTL_RAW.BIT_OR ................................................................. B-5
- UTL_RAW.BIT_XOR ............................................................... B-6
- UTL_RAW.CAST_TO_RAW ....................................................... B-7
- UTL_RAW.CAST_TO_VARCHAR2 ........................................... B-7
- UTL_RAW_COMPARE .............................................................. B-8
- UTL_RAW.CONCAT ............................................................... B-8
- UTL_RAW.CONVERT .............................................................. B-9
- UTL_RAW.COPIES ................................................................. B-9
- UTL_RAW.LENGTH ................................................................. B-10
- UTL_RAW.OVERLAY .............................................................. B-10
- UTL_RAW.REVERSE ............................................................. B-11
- UTL_RAW.SUBSTR ............................................................... B-12
- UTL_RAW.TRANSLATE .......................................................... B-12
- UTL_RAW.TRANSLITERATE ................................................... B-14
- UTL_RAW.XRANGE ............................................................... B-15

### C  Gateway Initialization Parameters

**Gateway Initialization Parameters**

- Gateway Initialization File .................................................. C-2
- Gateway Parameters .......................................................... C-2
  - HS_DB_DOMAIN ............................................................... C-2
  - HS_DB_NAME ................................................................. C-2
  - LOG_DESTINATION ........................................................ C-3
D  Heterogeneous Message Propagation DBA Views

DBA_OUTBOUND_SCHEDULES: ................................................................. D-2

DBA_INBOUND_SCHEDULES: ............................................................... D-4

Index
Send Us Your Comments

Part No. A96196-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. 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 about this manual?

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

  infoibm_us@oracle.com

If you would like a reply, please give your name, address, telephone number, and electronic mail address (optional).

If you have problems with the software, please contact your local Oracle Support Services.
The Oracle9i release of the Oracle Procedural Gateway and Tools for IBM MQSeries provides access to MQSeries services. This gateway requires a host with or without multiple processors that is capable of running 64-bit applications. The UNIX platforms supported by this gateway release are: Sun Solaris Operating System (SPARC), IBM RS/6000 AIX-based system, Hewlett-Packard 9000 Series 700 or 800.

Read this guide if you are responsible for tasks such as:
- administering the gateway
- setting up gateway security
- using the gateway
- diagnosing gateway errors

Understand the fundamentals of your operating system, the procedural gateways, PL/SQL, the Oracle server, and MQSeries software before using this guide to install, configure, or administer the gateway.

**Intended Audience**

This guide is intended for anyone responsible for installing, configuring, or administering the Oracle9i Procedural Gateway for IBM MQSeries, and also for developers writing applications that access message queuing systems, especially for those who need to access queues owned by both IBM MQSeries and other non-Oracle message queuing systems as well as queues owned by Oracle Advanced Queuing (AQ).
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.

Product Name

The complete name for this product is Oracle Procedural Gateway and Tools for IBM MQSeries, also called Oracle9i Procedural Gateway for IBM MQSeries with this release, and also called PG4MQ.
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 actually appears. For input, the following conventions apply:

<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.</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>{}</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>Square brackets indicate that the enclosed arguments are optional. Do not enter the brackets themselves.</td>
</tr>
<tr>
<td></td>
<td>Vertical lines separate choices.</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>
</tbody>
</table>

Other punctuation, such as commas, quotes, or the pipe symbol `|`, must be entered as shown unless otherwise specified. Directory names, fileids, and so on appear in the required letter case in examples. The use of *italics* in a fileid indicates that those portions that are in *italics* can vary.

Gateway commands, fileids, 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 because they have reserved meanings within the Oracle system.

The PGM8 procedures follow a slightly different set of convention rules. Refer to "Procedure Conventions" on page A-3 for more information.
Related Publications

A supporting Oracle document, Oracle9i Heterogeneous Connectivity Administrator’s Guide, contains information common to all procedural gateways, including important information on functions, parameters, and error messages.

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 part of some UNIX directory names and should not be confused as a prompt character.

Storage Measurements

Storage measurements use the following abbreviations:

- K, for kilobyte, which equals 1,024 bytes
- M, for megabyte, which equals 1,048,576 bytes
- G, for gigabyte, which equals 1,073,741,824 bytes
Documents Referenced in this Guide

Oracle Call Interface Programmer’s Guide
Oracle9i Database Administrator’s Guide
Oracle9i Database Error Messages
Oracle9i Database Reference
Oracle9i Database Utilities
Oracle9i Heterogeneous Connectivity Administrator’s Guide
Oracle9i Net Services Administrator’s Guide
Oracle9i Net Services Reference Guide
Oracle9i Security Overview
Oracle9i SQL Reference
Oracle9i Supplied PL/SQL Packages and Types Reference
PL/SQL User’s Guide and Reference
Oracle9i Administrator’s Reference Release 2 (9.2.0.1.0) for UNIX Systems: AIX-Based Systems, Compaq Tru64 UNIX, HP 9000 Series HP-UX, Linux Intel, and Sun Solaris
Oracle9i Installation Guide Release 2 (9.2.0.1.0) for UNIX Systems: AIX-Based Systems, Compaq Tru64 UNIX, HP 9000 Series HP-UX, Linux Intel, and Sun Solaris
Oracle Visual Workbench for Oracle Procedural Gateways for IBM MQSeries Installation and User’s Guide
1

Introduction

This chapter provides an overview of message queuing, IBM MQSeries, and the gateway’s role when accessing MQSeries queues:

- Introduction to Message Queuing on page 1-2
- Introduction to IBM MQSeries on page 1-2
- Introduction to the Gateway on page 1-4
Introduction to Message Queuing

Message queuing allows distributed applications to communicate asynchronously by sending messages between the applications. The messages from the sending application are stored in a queue and are retrieved by the receiving application. The applications send or receive messages through a queue by sending a request to the message queuing system. Sending and receiving applications can use the same message queuing system or different ones, allowing the message queuing system to handle the forwarding of the messages from the sender queue to the recipient queue.

Queued messages can be stored at intermediate nodes until the system is ready to forward them to the next node. At the destination node, the messages are stored in a queue until the receiving application retrieves them from the queue. Message delivery is guaranteed even if the network or application fails. This provides for a reliable communication channel between the applications.

The complexity and details of the underlying model (to store and forward messages between different environments) are handled by the message queuing system. By maintaining this level of abstraction, distributed applications can be developed without the need to worry about the details of how the information is transported.

Because the sending and receiving applications operate independently of one another, the sending application is less dependent on the availability of the remote application, less dependent on the network between them, and less dependent on the computer system on which the receiving application runs. This leads to a higher level of availability for the participating applications.

Messages and message queue operations can be configured by the applications to operate in specific modes. For example, a sending application can specify that queued messages should survive system crashes. As another example, the receiving application can specify a maximum waiting period for a receiving operation from a queue (in case no messages are available yet on the receiving queue).

Introduction to IBM MQSeries

IBM MQSeries is a message queuing system based on a model of message queue clients and message queue servers. The applications run either on the server node where the queue manager and queues reside, or from a remote client node. Applications can send or retrieve messages only from queues owned by the queue manager to which they are connected.
## MQSeries Terms

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message queues</td>
<td>are storage areas for messages exchanged between applications.</td>
</tr>
<tr>
<td>Message queue interface (MQI)</td>
<td>is an application programming interface (API) for applications that want to send or receive messages through IBM MQSeries queues.</td>
</tr>
<tr>
<td>MQSeries client configuration</td>
<td>is an MQSeries configuration where the queue manager and message queues are located on a different (remote) computer or node than the application software. Client applications connect to the remote queue manager using IBM software that provides the necessary networking software to connect to the remote queue manager.</td>
</tr>
<tr>
<td>MQSeries server configuration</td>
<td>is an MQSeries configuration where the queue manager and message queues are located on the same (local) computer or node as the application software. Client applications connect to the local queue manager using MQI.</td>
</tr>
<tr>
<td>Queue manager</td>
<td>provides the message queuing facilities that applications use, and manages the queue definitions, configuration tables, and message queues. The queue manager also forwards messages from the sender queue to the remote recipient queues.</td>
</tr>
<tr>
<td>Triggers</td>
<td>is an MQSeries feature that enables an application to be started automatically when a message event, such as the arrival of a message, occurs. Triggers can be used to invoke programs or transactions. For example, a trigger could cause an Oracle application to call the gateway to retrieve an MQSeries message and process it.</td>
</tr>
</tbody>
</table>
The Oracle9i Procedural Gateway for IBM MQSeries allows Oracle applications to integrate with other MQSeries applications. Oracle applications can send messages to other MQSeries applications or receive messages from them. With the gateway, Oracle applications access IBM MQSeries message queues through remote procedure call (RPC) processing.

The gateway extends the remote procedure call (RPC) facilities that are available with the Oracle server and enables any client application to use PL/SQL to access messages in MQSeries queues. The gateway provides PL/SQL procedures that are translated by the gateway into MQI (message queue interface) calls. These procedures resemble the calls and types of MQI, but they are adapted to take full advantage of the transaction integration with the Oracle integrating server. For more information, refer to Appendix A, "The PGM8, PGM_UTL8, and PGM_SUP Packages".

Through MQSeries, the gateway communicates with any other MQSeries systems on various platforms, including mainframes, UNIX, Windows NT, and other desktop environments. The gateway does not require any Oracle software on the remote system. The gateway integrates with existing MQSeries applications without any changes to those applications and allows users to leverage their investment in these applications while providing them with the ability to exploit the benefits of message-oriented middleware.

The gateway also provides a way to integrate these existing MQSeries applications with new technology areas, such as network computing. Any Oracle application can invoke the PL/SQL procedures, including applications that use the Oracle application server.

**Developing Gateway Applications**

If you are developing applications that access MQSeries through the gateway, use the Oracle Visual Workbench for Oracle Procedural Gateways for IBM MQSeries. The Visual Workbench allows you to define an interface for accessing MQSeries and to define how to convert message data sent or retrieved from MQSeries queues.

The Visual Workbench generates PL/SQL code for the interface and data conversions. This generated code is called the message interface package (MIP). The MIP provides the underlying code to interact with the gateway, performs message data conversion, and provides an easy to use interface for Oracle applications to exchange messages with remote MQSeries applications. Refer to the

When necessary, the generated MIP code can be modified to use MQSeries functions that are not supported by Visual Workbench or can be modified to enhance the message data conversions. Refer to Appendix A, "The PGM8, PGM_U TL8, and PGM_SUP Packages" and Appendix B, "UTL_RAW Package" for more information.

Gateway Terms

Gateway initialization file is a file containing parameters that govern the operation of the gateway.

Gateway remote procedures are remote procedures implemented by the gateway. These procedures are used to invoke MQSeries operations.

Message interface package (MIP) is an Oracle PL/SQL package generated by the Visual Workbench that serves as an interface between an existing MQSeries application and an Oracle application. The MIP performs any necessary data conversion and invokes the gateway RPCs to perform appropriate MQSeries operations. Refer to the Oracle Visual Workbench for Oracle Procedural Gateways for IBM MQSeries Installation and User’s Guide for more information.

Oracle integrating server is any Oracle server that communicates with the gateway. Oracle applications do not communicate directly with the gateway. Instead they execute PL/SQL code at an Oracle integrating server to invoke the gateway procedures. The Oracle integrating server can be on the same computer as the gateway or on a different computer.

production Oracle server as used in this book, production Oracle server refers to any Oracle server that you are using for production, for actual business, not for testing.

PL/SQL stored procedure is a compiled PL/SQL procedure that is stored in the Oracle integrating server or is included with the gateway.
Remote procedure call is a programming call that invokes a program on a system in response to a request from another system.

Visual Workbench is an abbreviated term for the Oracle Visual Workbench for Oracle Procedural Gateways for IBM MQSeries.

Advantages of Using the Gateway

Using the gateway to access MQSeries provides the following advantages:

- Transactional support
  The gateway and the Oracle integrating server allow MQSeries operations and Oracle integrating server updates to be performed in a coordinated fashion. Oracle two-phase commit protection is extended to the MQSeries environment without any special programming.

- Fast remote procedures
  The remote procedures implemented by the gateway are optimized for efficient processing of MQSeries requests.
  The remote procedures to the gateway and MQSeries are an optimized PL/SQL package that is precompiled in the gateway. Because there are no additional software layers on the target system, overhead is minimized.

- Location transparency
  Client applications need not be specific to the operating system. For example, your Oracle application can send MQSeries messages to an application on IBM MVS. If the receiving application is moved to a different platform, then you do not need to change your Oracle application.

- Flexible interface
  Using the MIPs generated by the Visual Workbench, you can use the gateway to interface with existing procedural logic or to integrate new procedural logic into an Oracle integrating server environment.

- Oracle integrating server integration
  The integration of the Oracle integrating server with the gateway allows you to benefit from existing and future Oracle features.
- Wide selection of tools

The gateway supports any tool or application that supports PL/SQL. This includes applications built with traditional Oracle tools, such as Developer/2000, or applications built for intranet or Internet environments supported by the Oracle Application Server. The gateway also works with packaged Oracle applications such as Oracle Financials® and with many third party tools such as Visual Basic, PowerBuilder, and Lotus Notes.

- Security

The gateway is compatible with the MQSeries security authorization mechanism.

Gateway Architecture

**Figure 1–1 Components of the Gateway Architecture**
Component Descriptions

This section describes components of the gateway architecture.

Oracle Applications
Oracle applications connect to an Oracle integrating server. They send data to, and receive data from, MQSeries queues by invoking the gateway RPCs.

Oracle Integrating Server
Oracle applications do not connect directly to the gateway, but connect indirectly by connecting to an Oracle integrating server. The Oracle integrating server communicates with a gateway in the normal Oracle server-to-server manner using Oracle Net. The gateway is a single process and does not start background processes. On UNIX platforms, a gateway process is started for each user session.

Oracle Net
Oracle Net provides client-to-server and server-to-gateway communication. It enables an Oracle application to communicate with the Oracle integrating server, and it enables the Oracle integrating server to communicate with the gateway.

If the Oracle integrating server is not on the workstation where the gateway resides, then you must install the correct Oracle networking software on the platform where the Oracle integrating server resides.

Gateway
The Oracle application invokes the RPCs (remote procedure calls) which are implemented by the gateway with PL/SQL. The gateway procedures map these RPCs to MQSeries MQI calls to perform the appropriate MQSeries operation.

The gateway is accessed through the Oracle integrating server by using a database link name created by an Oracle CREATE DATABASE LINK statement. The database link is the same construct used to identify other Oracle integrating server databases.

MQSeries Queue Manager
The MQSeries server is where the MQSeries queue manager and message queue reside. The MQSeries server might, or might not, be on the same computer as the gateway.
**MQSeries Application**

MQSeries applications connect directly to the MQSeries queue manager using MQSeries MQI calls to perform the appropriate MQSeries operation.

**Gateway Structure**

The gateway has some of the same components as an Oracle integrating server. The following components are included:

- a directory where the gateway software is installed
- a system identifier (SID)
- an initialization file similar to the Oracle integrating server’s initialization file

The gateway does not have control, redo, or database files, nor does it have the full set of subdirectories and other files associated with an Oracle integrating server.

**Starting the Gateway**

The gateway is not started in the same way as the Oracle integrating server. It has no background processes and does not require a management utility such as the Oracle Enterprise Manager. Each Oracle integrating server user session that accesses a gateway creates an independent process on the host computer that runs the gateway.

**Communication**

All communication between the Oracle integrating server, gateway, and MQSeries queues is handled through RPC calls to the gateway. The PL/SQL code to do these calls is automatically generated by the Visual Workbench. For more information, refer to Appendix A, “The PGM8, PGM_UTL8, and PGM_SUP Packages” or the Oracle Visual Workbench for Oracle Procedural Gateways for IBM MQSeries Installation and User’s Guide.
This chapter contains information specific to Release 9.2.0.1.0 of the gateway:

- The Product Set on page 2-2
- Changes and Enhancements on page 2-2
- Known Problems for Release 9.2.0.1.0 on page 2-6
- Known Restrictions for Release 9.2.0.1.0 on page 2-6
The Product Set

The following table lists the release numbers of the components included with the Oracle9i server.

<table>
<thead>
<tr>
<th>Product</th>
<th>Release Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle9i Procedural Gateway for IBM MQSeries</td>
<td>9.2.0.1.0</td>
</tr>
<tr>
<td>Oracle Net</td>
<td>9.2.0.1.0</td>
</tr>
</tbody>
</table>

Changes and Enhancements

The following changes and enhancements are for all releases of the Oracle9i gateway products.

Changes and Enhancements for Release 9.2.0.1.0

For heterogeneous message propagation, refer to Chapter 9.

Oracle Server Dependencies

This release of the Oracle Procedural Gateway and Tools for IBM MQSeries requires the latest released patch set for Oracle9i server release 9.2 or release 9.0.1, or Oracle8i server release 8.1.7.

Password Encryption Utility

This release of the Oracle Procedural Gateway and Tools for IBM MQSeries includes a utility to support encryption of plain text passwords in the Gateway Initialization File. For more information, refer to "TRANSACTION_RECOVERY_PASSWORD" on page C-8.
Changes and Enhancements for Release 9.0.1.0.1

The following changes and enhancements are for the previous release of the Oracle9i gateway products.

**HS= Parameter in tnsnames.ora file**

When configuring the connect descriptor entry in tnsnames.ora for the gateway, the release 8.0.4 gateway used \((HS=(\text{ \text{ DRIVER=} }))\), and the release 9.0.1.0.1 gateway uses \((HS=\text{OK})\). Refer to "Configuring Oracle Net for the Oracle Integrating Server" on page 7-16 for details.

**PG4MQ Data Types**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>V401</th>
<th>V804</th>
<th>V817 and V901</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQOD</td>
<td>PGM.MQOD@dblink</td>
<td>PGM.MQOD</td>
<td>PGM8.MQOD</td>
</tr>
<tr>
<td>MQMD</td>
<td>PGM.MQMD@dblink</td>
<td>PGM.MQMD</td>
<td>PGM8.MQMD</td>
</tr>
<tr>
<td>MQPMO</td>
<td>PGM.MQPMO@dblink</td>
<td>PGM.MQPMO</td>
<td>PGM8.MQPMO</td>
</tr>
<tr>
<td>MQGMO</td>
<td>PGM.MQGMO@dblink</td>
<td>PGM.MQGMO</td>
<td>PGM8.MQGMO</td>
</tr>
<tr>
<td>MQODRAW</td>
<td>N.A.</td>
<td>PGM.MQODRAW</td>
<td>PGM8.MQODRAW</td>
</tr>
<tr>
<td>MQMDRAW</td>
<td>N.A.</td>
<td>PGM.MQMDRAW</td>
<td>PGM8.MQMDRA</td>
</tr>
<tr>
<td>MQPMORAW</td>
<td>N.A.</td>
<td>PGM.MQPMORAW</td>
<td>PGM8.MQPMORAW</td>
</tr>
<tr>
<td>MQGMORAW</td>
<td>N.A.</td>
<td>PGM.MQGMORAW</td>
<td>PGM8.MQGMORAW</td>
</tr>
</tbody>
</table>

**PGM_UTL Procedures**

<table>
<thead>
<tr>
<th>Procedure</th>
<th>V401</th>
<th>V804</th>
<th>V817 and V901</th>
</tr>
</thead>
<tbody>
<tr>
<td>TO_RAW</td>
<td>N.A.</td>
<td>PGM_UTL.TO_RAW</td>
<td>PGM_UTL8.TO_RAW</td>
</tr>
<tr>
<td>RAW_TO_MQMD</td>
<td>N.A.</td>
<td>PGM_UTL_RAW.TO_MQMD</td>
<td>PGM_UTL8_RAW.TO_MQMD</td>
</tr>
<tr>
<td>RAW_TO_MQPMO</td>
<td>N.A.</td>
<td>PGM_UTL_RAW.TO_MQPMO</td>
<td>PGM_UTL8_RAW.TO_MQPMO</td>
</tr>
<tr>
<td>RAW_TO_MQGMO</td>
<td>N.A.</td>
<td>PGM_UTL_RAW.TO_MQGMO</td>
<td>PGM_UTL8_RAW.TO_MQGMO</td>
</tr>
</tbody>
</table>
Changes and Enhancements

PG4MQ API prototype changes

<table>
<thead>
<tr>
<th>API</th>
<th>V401 arguments</th>
<th>V804 arguments</th>
<th>V817 &amp; V901 arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQOPEN</td>
<td>(MQOD,INT,INT)</td>
<td>(RAW,INT,INT)</td>
<td>(RAW,INT,INT)</td>
</tr>
<tr>
<td>MQPUT</td>
<td>(INT,MQMD,MQPMO,RAW)</td>
<td>(INT,RAW,RAW,RAW)</td>
<td>(INT,RAW,RAW,RAW)</td>
</tr>
<tr>
<td>MQGET</td>
<td>(INT,MQMD,MQGMO,RAW)</td>
<td>(INT,RAW,RAW,RAW)</td>
<td>(INT,RAW,RAW,RAW)</td>
</tr>
<tr>
<td>MQCLOSE</td>
<td>(INT,INT)</td>
<td>(INT,INT)</td>
<td>(INT,INT)</td>
</tr>
</tbody>
</table>

Refer to Appendix A for the details of APIs.

Heterogeneous Services Architecture

This release of the Oracle9i Procedural Gateway for IBM MQSeries utilizes the Oracle Heterogeneous Services component within the Oracle9i server.

For additional information, refer to Oracle9i Heterogeneous Connectivity Administrator’s Guide.

Performance Enhancements

Oracle9i Procedural Gateway for IBM MQSeries contains several internal performance enhancements. This product has shown major improvements in response time and CPU utilization for all relevant address spaces for a variety of workloads compared to version 4 gateways. The actual performance improvement at your site might vary, depending upon your installation type and workload.

New PG4MQ Packages

PGM8 and PGM_UTL8 packages are new in this release. These packages provide new features as well as ensure backward compatibility. Refer to "Migration Tips" on page A-5 in Appendix A for the details of upgrading your existing PL/SQL application programs to use Oracle9i PG4MQ features.
## Bugs Fixed

<table>
<thead>
<tr>
<th>Bug Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2329347</td>
<td>GATEWAY HANGS OR GIVES ORA-4062 ON MQGET CALL TO EMPTY QUEUE</td>
</tr>
<tr>
<td>2321389</td>
<td>PG4MQ 9.2.0.1.0 PG4MQ FAILS WHEN AUTHORIZATION_MODE=STRICT</td>
</tr>
<tr>
<td>2321387</td>
<td>TRANSACTION RECOVERY USER/PASSWORD IS SET AND CHECKED INCORRECTLY.</td>
</tr>
<tr>
<td>2182483</td>
<td>MQMD.USERIDENTIFIER CONTAINS WHITE SPACES AND CAUSED MQ-IMS BRIDGE FAILURE</td>
</tr>
<tr>
<td>2147453</td>
<td>BROWSE OPTION OF THE PGM.MQGET PACKAGE RETURNS ONLY 1 MESSAGE.</td>
</tr>
<tr>
<td>2115441</td>
<td>PUTDATE AND PUTTIME ARE NOT REPORTED TO THE QUEUE</td>
</tr>
<tr>
<td>2074241</td>
<td>MQPUT.MQMD.USERIDENTIFIER NOT POSTED CORRECTLY UNDER VERSION 8</td>
</tr>
<tr>
<td>2026117</td>
<td>ORA-4062 WHEN SENDING MESSAGES &gt; 4000 BYTES, WHILE SMALLER MESSAGES ARE OK</td>
</tr>
<tr>
<td>1925901</td>
<td>PG4MQ9I RETRUNS ORA-4062 WHEN TRANSACTION_MODE IS SET TO COMMIT_CONFIRM</td>
</tr>
<tr>
<td>1918699</td>
<td>LONG MESSAGES CAN NOT BE PASSED TO PGM.MQPUT/PGM.MQGET PROCEDURES</td>
</tr>
<tr>
<td>1917293</td>
<td>VERSION 8 OF THE GATEWAY DOES NOT CREATE THE COORRECT MSG HDR FOR MQSERIES MSG</td>
</tr>
<tr>
<td>1897823</td>
<td>ORA-4062 WHEN RUNNING A MIP</td>
</tr>
</tbody>
</table>
Known Problems for Release 9.2.0.1.0

The problems documented in this section are specific to the Oracle Procedural Gateway for IBM MQSeries and are known to exist in this release of the product. These problems will be fixed in a future gateway release. If you have any questions or concerns about these problems, contact Oracle Support Services.

A current list of problems is available online. Contact your local Oracle Corporation office for information about accessing this online information.

Known Restrictions for Release 9.2.0.1.0

The following restrictions are known to exist for the products in this release. Restrictions are not scheduled to change in future releases.

32K buffer size limitation
The buffer sizes of MQPUT and MQGET gateway function calls are limited to 32K (32767) bytes due to the limitation of PL/SQL RAW data type.

Encrypted Format Login
If the DBLINK_ENCRYPT_LOGIN parameter is set to TRUE in the initialization parameter file (initSID.ora) for the database that is used by the Oracle client, then you must change the setting to FALSE to allow Oracle to communicate with the gateway.
This chapter provides information about the hardware and software required for the installation of the Oracle9i Procedural Gateway for IBM MQSeries and the online documentation. The topics include:

- Hardware Requirements on page 3-2
- Software Requirements on page 3-3
- Oracle Integrating Server on page 3-4
- Recommended Documentation on page 3-4
# Hardware Requirements

The following table contains the hardware requirements for Oracle9i Procedural Gateway for IBM MQSeries.

<table>
<thead>
<tr>
<th>Hardware Items</th>
<th>Required for AIX-Based Systems</th>
<th>Required for HP 9000 Series HP-UX</th>
<th>Required for Sun Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD-ROM</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>CPU</td>
<td>IBM RS/6000 AIX-Based System Processor</td>
<td>HP 9000 Series 700 or 800 processor for HP-UX 11.0</td>
<td>Sun Solaris Operating System (SPARC) Processor</td>
</tr>
<tr>
<td>Disk Space</td>
<td>1.1 GB</td>
<td>550 MB</td>
<td>550 MB</td>
</tr>
<tr>
<td>Memory</td>
<td>256 MB</td>
<td>256 MB</td>
<td>256 MB</td>
</tr>
<tr>
<td>Swap Space</td>
<td>512 MB or twice the amount of RAM, whichever is greater</td>
<td>512 MB or twice the amount of RAM, whichever is greater</td>
<td>512 MB or twice the amount of RAM, whichever is greater</td>
</tr>
</tbody>
</table>
Software Requirements

The following table contains the software requirements for Oracle9i Procedural Gateway for IBM MQSeries.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Requirement</th>
<th>IBM Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX-Based Systems</td>
<td>64-bit OS only</td>
<td>When the gateway resides on the same computer as the MQSeries server software, then IBM MQSeries for AIX version 5 release 5.1 or later is required. When the gateway resides on a different computer than the MQSeries server software, then IBM MQSeries Client for AIX version 5 release 5.1 or later is required on the gateway computer.</td>
</tr>
<tr>
<td></td>
<td>AIX 4.3.3 or AIX 5L</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OS Patches:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Check with your software vendor for current maintenance requirements.</td>
<td></td>
</tr>
<tr>
<td>HP 9000 Series 700 or 800 (HP-UX)</td>
<td>64-bit OS only</td>
<td>When the gateway resides on the same computer as the MQSeries server software, then IBM MQSeries for HP 9000 version 5 release 5.1 or later is required. When the gateway resides on a different computer than the MQSeries server software, then IBM MQSeries Client for HP 9000 version 5 release 5.1 or later is required on the gateway computer.</td>
</tr>
<tr>
<td></td>
<td>HP-UX 11.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OS Patches:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Check with your software vendor for current maintenance requirements.</td>
<td></td>
</tr>
<tr>
<td>Sun Solaris Operating System (SPARC)</td>
<td>32-bit OS only (32-bit or 64-bit hardware)</td>
<td>When the gateway resides on the same computer as the MQSeries server software, then IBM MQSeries for Sun Solaris version 5 release 5.1 or later is required. When the gateway resides on a different computer than the MQSeries server software, then IBM MQSeries Client for Sun Solaris version 5 release 5.1 or later is required on the gateway computer.</td>
</tr>
<tr>
<td></td>
<td>Solaris 2.6 or later</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OS Patches:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Check with your software vendor for current maintenance requirements.</td>
<td></td>
</tr>
</tbody>
</table>

Set ulimit value for the maximum number of open files per process to 1024 or greater:

prompt> ulimit -n 1024

Note: All IBM software must be installed before the gateway software is installed. For example, if IBM MQSeries software is not installed before PG4MQ, then the product link will fail.
Oracle Integrating Server

The Oracle server that is to act as the Oracle integrating server requires the latest released patch set for Oracle9i server release 9.2 or release 9.0.1, or Oracle8i server release 8.1.7.

Recommended Documentation

In addition to the documentation that is sent with the gateway, the following Oracle publications are recommended:

- *Oracle Visual Workbench for Oracle Procedural Gateways for IBM MQSeries Installation and User’s Guide*
- *PL/SQL User’s Guide and Reference*
- *Oracle9i Net Services Administrator’s Guide*
- *Oracle9i Net Services Reference Guide*

Documentation can be found at the following web sites:

http://metalink.oracle.com
http://docs.oracle.com
This chapter guides you through the basic concepts and pre-installation steps for Oracle9i Procedural Gateway for IBM MQSeries. The following topics provide information about Oracle9i Procedural Gateway for IBM MQSeries, environment variables settings, and starting Oracle Universal Installer:

- Preinstallation Tasks on page 4-2
- About Oracle Universal Installer on page 4-7
Preinstallation Tasks

The preinstallation tasks for the Oracle9i Procedural Gateway for IBM MQSeries are divided into the following parts:

- MQSeries Software
- Setting Environment Variables

MQSeries Software

1. Determine where the MQSeries queue manager is running.
   - Local Machine
     - If the MQSeries queue manager is running on a local machine, then the queue manager is running on the same machine where you intend to install the gateway product set.
   - Remote Machine
     - If the MQSeries queue manager is running on a remote machine, then the queue manager is running on a different machine, not the machine where you intend to install the gateway product set.

2. Verify that the MQSeries software is already installed. If the MQSeries server software resides on a different machine than the gateway, then the MQSeries client software must be installed on the gateway machine.

3. Identify the name of the MQSeries queue manager.

4. Identify the MQSeries client channel definition.

   If the queue manager resides on a different machine than the gateway, then the MQSeries client software is used to access the remote queue manager. A channel definition is required for this configuration.
Setting Environment Variables

Before installing Oracle9i Procedural Gateway for IBM MQSeries on UNIX platforms, the following environment variables must be set:

- ORACLE_HOME
- ORACLE_SID
- DISPLAY
- TMP

**Note:** Verify that the values that you assign to the environment variables, which are listed in this section, are less than 42 characters long. Longer values might generate errors such as “Word too long” during installation.

**ORACLE_HOME**

*ORACLE_HOME* is the root directory in which Oracle software is installed.

Oracle9i Procedural Gateway for IBM MQSeries cannot share the same *ORACLE_HOME* directory with other Oracle products. If you have installed other Oracle products, then Oracle9i Procedural Gateway for IBM MQSeries must be installed in a different *ORACLE_HOME* directory. Refer to "Preventing Conflicts Between ORACLE_HOME Directories" on page 4-4.

**Note:** Be sure to avoid installing Oracle9i Procedural Gateway for IBM MQSeries in an *ORACLE_HOME* directory containing other Oracle products, including the database. Such an installation could overwrite shared components, causing the products to malfunction.
Preventing Conflicts Between ORACLE_HOME Directories

To prevent a conflict between the software in an existing ORACLE_HOME directory and Oracle9i Procedural Gateway for IBM MQSeries, you must remove all references to the existing ORACLE_HOME directory. The following steps describe removing these references.

1. Unset your existing ORACLE_HOME variable by using one of the following commands.
   - C shell:
     `prompt> unsetenv ORACLE_HOME`
   - Bourne/Korn shell:
     `prompt> export ORACLE_HOME=

2. Edit the following environment variables so that they do not use the existing ORACLE_HOME value:

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>PATH</td>
<td>AIX-based Systems, HP 9000 Series HP-UX, and Sun Solaris</td>
</tr>
<tr>
<td>CLASSPATH</td>
<td>AIX-based Systems, HP 9000 Series HP-UX, and Sun Solaris</td>
</tr>
<tr>
<td>LD_LIBRARY_PATH</td>
<td>Sun Solaris</td>
</tr>
<tr>
<td>LIBPATH</td>
<td>AIX-based Systems</td>
</tr>
<tr>
<td>SHLIB_PATH</td>
<td>HP 9000 Series HP-UX</td>
</tr>
</tbody>
</table>

   **Note:** Verify that the C compiler is in your PATH before you start the installation.
Preinstallation Tasks

**Setting ORACLE_HOME**

Set the `ORACLE_HOME` environment variable by using one of the following commands:

- C shell
  ```
  prompt> setenv ORACLE_HOME <full path>
  ```

- Bourne/Korn shell
  ```
  prompt> ORACLE_HOME=<full path>
prompt> export ORACLE_HOME
  ```

**ORACLE_SID**

`ORACLE_SID` is used for the SID of the gateway. If the IBM MQSeries manager runs on the same machine as the gateway, set the value of the `ORACLE_SID` environment variable to `pg4mqs92`. If the IBM MQSeries manager runs on a different machine than the gateway, set the value of the `ORACLE_SID` environment variable to `pg4mqc92`.

**Setting ORACLE_SID**

Set the `ORACLE_SID` environment variable by using one of the following commands:

- C shell
  ```
  prompt> setenv ORACLE_SID pg4mqs92
  ```
  or
  ```
  prompt> setenv ORACLE_SID pg4mqc92
  ```

- Bourne/Korn shell
  ```
  prompt> ORACLE_SID=pg4mqs92
  prompt> export ORACLE_SID
  ```
  or
  ```
  prompt> ORACLE_SID=pg4mqc92
  prompt> export ORACLE_SID
  ```
DISPLAY

Setting the DISPLAY environment variable enables you to run the Oracle Universal Installer remotely from a local work station. On the system where you run the Oracle Universal Installer, set DISPLAY to the system name or IP address of your local workstation.

If you get an Xlib error when starting the Oracle Universal Installer such as “Failed to connect to server”, “Connection refused by server”, or “Can’t open display”, then run the commands on your local workstations as follows:

On Server where the Installer is Running

- C shell
  
  `prompt> setenv DISPLAY hostname:0.0`

- Bourne or Korn shell
  
  `prompt> export DISPLAY=hostname:0.0`
  `prompt> export DISPLAY`

In Session on Your Workstation

- C shell
  
  `prompt> xhost +server_name`

- Bourne or Korn shell
  
  `prompt> xhost +server_name`
About Oracle Universal Installer

**TMP**

During installation, Oracle Universal Installer uses a temporary directory for swap space. This directory must meet the "Hardware Requirements" on page 3-2. The installation might fail if you do not have sufficient space. The Oracle Universal Installer checks for the TMP environment variable to locate the temporary directory. If this environment variable does not exist, then the installer uses the /tmp directory.

The following example demonstrates how to set the TMP environment variable.

- **C shell**
  ```
  prompt> setenv TMP <full path>
  ```

- **Bourne/Korn shell**
  ```
  prompt> TMP=<full path>
  prompt> export TMP
  ```

About Oracle Universal Installer

Oracle9i Procedural Gateway for IBM MQSeries uses Oracle Universal Installer to configure environment variables and to install components. The Oracle Universal Installer guides you through each step of the installation process, so you can choose configuration options for a customized product.

The Oracle Universal Installer includes features that perform the following tasks:

- Explore and provide installation options for products
- Detect pre-set environment variables and configuration settings
- Set environment variables and configuration during installation
- Deinstall products
**oraInventory Directory**

The Oracle Universal Installer creates the *oraInventory* directory the first time it is run on your computer. The *oraInventory* directory keeps an inventory of products that the Oracle Universal Installer installs on your computer as well as other installation information. If you have previously installed Oracle products, then you may already have an *oraInventory* directory.

- When a UNIX group name is specified, it grants that group the permission to write to the *oraInventory* directory. If another group attempts to run the Oracle Universal Installer, then they must have permission to write to the *oraInventory* directory. If they do not have permission the installation will fail.

- Be sure the user running the Oracle Universal Installer has permission to write to the *oraInventory* directory and all its files. This is required to run the installer.

- The location of *oraInventory* is defined in `/etc/oratab/oraInst.loc` for HP 9000 Series HP-UX and AIX-Based Systems.

- The location of *oraInventory* is defined in `/var/opt/oraInst.loc` for Sun Solaris.

- The latest log file is `oraInventory_location/logs/installActions.log`. Log file names of previous installation sessions take the form `installActions datetime.log`.

- Do not delete or manually alter the *oraInventory* directory or its contents. Doing so can prevent the Oracle Universal Installer from locating products that you have installed on your system.
Starting Oracle Universal Installer

Perform the following steps to launch Oracle Universal Installer, which installs Oracle9i Procedural Gateway for IBM MQSeries:

1. Stop all Oracle processes and services (for example, the Oracle database).
2. Mount the installation CD-ROM.

   The Oracle Product Installation CD-ROM is in RockRidge format. To begin installation, insert the CD.

   To manually mount or unmount the CD-ROM, you must have root privileges. Unmount the CD-ROM before removing it from the drive.

To manually mount the CD-ROM, perform the following tasks:

a. Insert the Oracle9i Procedural Gateway for IBM MQSeries CD-ROM into the CD-ROM drive.

b. Log in as the root user.

c. Create the CD-ROM mount point directory.

   prompt> mkdir <mount_point>


d. Mount the CD-ROM drive on the mount point directory and exit the root account:

   prompt> mount options <device_name> <mount_point>
   prompt> exit
The following examples show how to mount the CD-ROM manually on /cdrom:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Command Example</th>
</tr>
</thead>
</table>
| AIX-Based Systems | prompt> mkdir /cdrom  
prompt> mount -o ro -V cdrfs <device_name> /mnt/cdrom  
prompt> exit |
| HP 9000 Series | Note: Add the following line to the /etc/pfs_fstab file:  
/dev/dsk/c5t2d0./cdrom pfs-rrip xlat=unix 0 0  
where /dev/dsk/c5t2d0 is the CD-ROM device file and pfs-rripxlat=unix 0 0 indicates the CD-ROM is in Rockridge extension.  
prompt> nohup /usr/sbin/pfs_mountd &  
prompt> nohup /usr/sbin/pfsd & |
| HP-UX         | Note: Insert CD-ROM in tray.  
prompt> /usr/sbin/pfs_mount /cdrom  
prompt> exit |
| Sun Solaris   | prompt> mkdir /cdrom  
prompt> mount -r -F hsfs <device_name> /cdrom  
prompt> exit |

**Note:** For AIX-Based System users only.

Before installing the gateways from the CD-ROM, become root and run rootpre.sh.

1. Log in as root and change directory to the CD-ROM mount point.
   
prompt> su root  
prompt> cd <mount_point>

2. Run rootpre.sh and exit root account.
   
prompt> ./rootpre.sh  
prompt> exit
3. Run Oracle Universal Installer from the CD-ROM.

**Note:** Be sure you are **not** logged in as the root user when you start the Oracle Universal Installer. If you are, then only the root user will have permissions to manage Oracle9i Procedural Gateway for IBM MQSeries.

**a.** Log in as the Oracle9i Procedural Gateway for IBM MQSeries user.

**b.** Start the Oracle Universal Installer by entering:

```
prompt> <mount_point>/runInstaller
```

This launches Oracle Universal Installer through which you can install Oracle9i Procedural Gateway for IBM MQSeries.
This chapter guides you through the installation of the Enterprise Edition of Oracle9i Procedural Gateway for IBM MQSeries, including detailed installation steps. The topics include:

- Installation on page 5-2
- Running root.sh on page 5-13
Installation

The first screen that is presented by the Oracle Universal Installer is the Welcome screen. To continue with the installation, click Next.

The next screen to be presented is the File Locations screen, Figure 5–1.

In the File Locations screen (Figure 5–1), the Source section of the screen allows you to specify the source location that the Oracle Universal Installer will use to install Oracle9i PG4MQ. You should not need to edit the file specification in the Path field - the default setting for this field points to the Oracle Universal Installer file on your Oracle9i PG4MQ CD-ROM.
The Path field in the Destination section of the File Locations screen allows you to specify the destination for your installation. You should not need to edit the path specification in the Path field because the default setting for this field points to ORACLE_HOME. After you have set the fields in the File Locations screen, as necessary, click Next to continue. After loading the necessary information from the CD-ROM, the installer will display the Available Products screen, Figure 5–2.

Figure 5–2 Available Products Screen

In the Available Products screen (Figure 5–2), select Oracle9i Database 9.2.0.1.0 and click Next to continue. The Oracle Universal Installer will display the Installation Types screen, Figure 5–3.
In the Installation Types screen (Figure 5–3), select Custom and click Next to continue. The Oracle Universal Installer will display the Available Product Component screen (Figure 5–4).
Figure 5–4  Available Product Components Screen

In the Available Product Components screen (Figure 5–4), use the check boxes to indicate the available product components that you want to install. By default, all of the available components are selected for you. You need to de-select the components that you do not want by clicking on the check boxes. Click Next to continue, and the Oracle Universal Installer will display the Where is the MQSeries Queue Manager Installed? screen, Figure 5–5.
In the Where is the MQSeries Queue Manager Installed? screen (Figure 5–5), select Local if the MQM runs on the same computer as the gateway, or select Remote if the MQM runs on a different computer than the gateway. Click Next to continue.
If you chose Local for your MQM in the earlier Oracle Universal Installer screen, then use this Local MQSeries Queue Manger Name screen (Figure 5-6), which is displayed now, and type in the local MQSeries queue manager name in the Queue Manager field. Click Next to continue and to display the Summary screen.
If you chose Remote for your MQM in the earlier Oracle Universal Installer screen, then the Remote MQSeries Queue Manager Name screen is displayed now. Enter the name for the remote MQSeries queue manager in the Queue Manager field and also enter the MQSeries channel name in the Channel field.

For information about server connection channels, refer to the IBM publication about MQSeries Clients or ask your MQSeries system administrator for the channel definition of the queue manager to which you want the gateway to connect.
The definition syntax is:

```
CHANNEL_NAME/PROTOCOL/server_address[(port)]
```

where `CHANNEL_NAME` and `PROTOCOL` must be uppercase, where `server_address` is the TCP/IP host name of the server, and where `(port)` is optional and is the TCP/IP port number on which the server is listening. If you do not provide a port number, then MQSeries uses the port number that is specified in the QM.INI file. If no value is specified in the QM.INI file, then MQSeries uses the port number that is identified in the TCP/IP services file for the service name MQSeries. If this entry in the services file does not exist, then a default value of 1414 is used. It is important that the port number that is used by the client and the port number that is used by the server listener program are the same.

For example:  `CHANNEL1/TCP/Sales`

Click Next to continue and to display the Summary screen, Figure 5–8.
The Oracle Universal Installer Summary screen (Figure 5-8) allows you to review a tree list of options and components for this installation. Click Install to display the Installation Status screen, Figure 5-9.
The Installation Status screen (Figure 5–9) shows the status of the installation as it proceeds, as well as the location of the Oracle Universal Installer log file for this installation session.

Be patient as the Oracle Universal Installer processes the software installation. Depending upon the CPU, CD-ROM drive, and hard drive in your computer, the installation process might take quite some time to be completed.
The final screen of the Oracle Universal Installer is the End of Installation screen (Figure 5-10). Assuming that your installation was successful, you can click Exit to exit the installer.
Running root.sh

Running root.sh

After installation is completed, use the following steps to run the root.sh script.

1. Log on as the root user.
2. Go to the $ORACLE_HOME/pg4mqseries/admin directory for your MQ gateway.
   
   prompt> cd $ORACLE_HOME/pg4mqseries/admin
3. Run the root.sh script.
   
   prompt> ./root.sh

   This script will allow the MQ gateway to operate for the strict security model.
4. Exit root user.
Running root.sh
Deinstallation and Reinstallation

This chapter guides you through the deinstallation and reinstallation options for Oracle9i Procedural Gateway for IBM MQSeries. They are described in the following topics:

- Deinstallation on page 6-2
- Reinstallation on page 6-5
Deinstallation

The following steps guide you through the deinstallation process of Oracle9i Procedural Gateway for IBM MQSeries. This process is divided into two parts:

- Deinstalling Using Oracle Universal Installer
- Deinstalling Oracle9i Procedural Gateway for IBM MQSeries

Deinstalling Using Oracle Universal Installer

Perform the following steps to deinstall Oracle9i Procedural Gateway for IBM MQSeries.

Deinstalling Oracle9i Procedural Gateway for IBM MQSeries

1. Start the Oracle Universal Installer. For information about starting the installer, refer to Starting Oracle Universal Installer on page 4-9.

   After the Oracle Universal Installer is launched, the Welcome screen appears. Click on Deinstall Products.

   The Welcome screen provides information about Oracle Universal Installer.

   The Oracle Universal Installer provides you with two ways to deinstall products:

   - **Deinstall Products**: deinstall individual components or the entire product.
   - **Installed Products**: View currently installed products and deinstall individual components or the entire product.

2. Review all installed components and check the ones you wish to deinstall. Click Remove.
The Inventory screen appears when you click **Deinstall Products** on the Welcome screen, or **Installed Products** on any screen.

The Inventory screen displays all of the components that are installed in **ORACLE_HOME**.

The following buttons appear on the Inventory screen:

- **Help**: Access detailed information about the functionality of the Inventory screen.
- **Remove**: deinstall all checked components from **ORACLE_HOME**.
- **Save As**: Save the inventory as text. A file browser dialog pops up when you click **Save As**. Accept a file name and the complete inventory list as displayed by the inventory screen will be logged into this file as text.
- **Close**: Quit the Inventory screen.
- **Location**: View the full location path of the selected component.
Note: The "+" sign before a product name indicates that there are more components and files installed within that particular product. Click on it to view dependent components. If you choose to remove a product or component, then all of its dependent components and files are also deinstalled.

If you wish to deinstall Oracle9i Procedural Gateway for IBM MQSeries completely, check the box displayed before the product name. It is listed directly below the ORACLE_HOME name.

Note: If you deinstall a product or component, then all of its dependent components and files will also be deinstalled.

3. Verify the components selected for deinstallation, and click Yes.

Figure 6–2 Confirmation Screen

The Confirmation screen lists all the components selected for deinstallation in the previous step. Scroll down the screen to verify selected components.

The following buttons appear on the Confirmation screen:

- Help: Access detailed information about the functionality of the Confirmation screen.
- Yes: Start deinstallation of listed components.
- No: Return to the Inventory screen. Listed components are not removed from ORACLE_HOME.
Monitor the deinstallation process.

The Remove Progress Bar screen appears when you click **Remove**. The Oracle Universal Installer detects all components chosen for deinstallation from the Inventory screen and removes them from **ORACLE_HOME**.

- **Cancel**: To discontinue the deinstallation process.

---

**Note**: If you deinstall a product or component, then all of its dependent components and files will also be deinstalled.

---

You have successfully deinstalled Oracle9i Procedural Gateway for IBM MQSeries.

---

**Reinstallation**

To reinstall Oracle9i Procedural Gateway for IBM MQSeries over the same version, deinstall, and then reinstall the product. Also refer to "Deinstalling Oracle9i Procedural Gateway for IBM MQSeries" on page 6-2.
After installing the gateway, follow the instructions in this chapter to configure the gateway. Gateway software configuration tasks and other administrative topics include:

- Configuration Overview on page 7-2
- Configuration Overview on page 7-2
- Configuring Oracle Net for the Gateway on page 7-10
- Configuring Oracle Net for the Oracle Integrating Server on page 7-16
- Creating a Transaction Log Queue on page 7-19
- Administering Database Links on page 7-19
- Installing the Visual Workbench Repository on page 7-22
- Preparing the Production Oracle Server on page 7-28
Configuration Overview

The gateway works with several components and products to communicate between the Oracle® server and MQSeries queues:

**Figure 7–1 Gateway Communication**

- **Oracle Net**
  The gateway and Oracle integrating server communicate using Oracle Net in a server-to-server manner. Both the gateway and the Oracle integrating server must have Oracle Net communication enabled, by configuring the `tnsnames.ora` and `listener.ora` files.

- **Gateway initialization files and parameters**
  The gateway has initialization files and parameters that you must customize for your installation. For example, you choose your gateway system identifier (SID), and you also provide information such as the gateway log file destination.
Configuring the Gateway

The gateway is installed and pre-configured using default values for the gateway SID, directory names, filenames, and gateway parameter settings. The default SID values are:

- pg4mq92
  This is the default SID that is used when the gateway resides on the same computer as the MQSeries software.
- pg4mqc92
  This is the default SID that is used when the gateway resides on a different computer than MQSeries software. In this case, the gateway functions as a remote MQSeries client.

A basic gateway boot file and a gateway initialization file are also installed, and values are set in these files based on information entered during the installation phase.

Using the Gateway with the Default Values

If you are configuring one gateway instance, and if you have no need to change any of the default values, then much of the gateway configuration process is completed by the Oracle Universal Installer. In this case, perform the following actions:

- Skip all steps under "Changing Default Values" on page 7-4.
- Skip "Step 1: Configure the Oracle Net TNS listener for the gateway" on page 7-10.
- Begin with "Step 2: Stop and start the TNS listener for the gateway" on page 7-14, and continue to the end of the chapter.

Using the Gateway without the Default Values

If multiple instances of the gateway are being configured, or to modify the default values set during the installation phases, then begin with the steps under "Changing Default Values" and continue to the end of the chapter.
Changing Default Values

When changing default values, choose a gateway SID and customize the gateway boot file and the Gateway Initialization File.

**Step 1: Choose a system ID for the gateway**

The gateway SID is a string of 1 to 64 alphanumeric characters that identifies a gateway instance. The SID is used in the gateway boot file and as part of the file name for the gateway parameter file. Choose a SID different from the default SID and different from `pg4mqs92` and `pg4mqc92`.

You need one gateway instance, and therefore one gateway SID, for each queue manager you want to access. If you want to access two different queue managers, then you need two gateway SIDs, one for each instance of the gateway. Or, if you have one queue manager and want to access it sometimes with one set of gateway parameter settings and at other times with different gateway parameter settings, then you can do so by having multiple gateway SIDs for the one queue manager.

**Step 2: Customize the gateway boot file**

The gateway boot file initializes environment variables. The boot file is required, and a default boot file is installed by the Oracle Universal Installer:

<table>
<thead>
<tr>
<th>Gateway Running Mode</th>
<th>Boot File Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>As MQSeries server</td>
<td><code>ORACLE_HOME/pg4mqseries/admin/initsid.gtwboot</code></td>
</tr>
<tr>
<td>As MQSeries client</td>
<td><code>ORACLE_HOME/pg4mqseries/admin/initsid.gtwboot</code></td>
</tr>
</tbody>
</table>

where `sid` is the default SID of `pg4mqs92` or `pg4mqc92`. If you chose a SID other than the default, then rename this file using the SID you chose in Step 1. If you have multiple gateway instances, then copy the default boot file as necessary, and rename each gateway instance using the SID of each instance.
The default boot files contain the following required entries:

**Gateway Running in MQSeries Server Configuration**

```plaintext
GATEWAY_SID=gateway_sid
SERVER_PATH=gateway
LOG_DESTINATION=log_file
LD_LIBRARY_PATH=mqseries_directory
```

where:

- **gateway_sid** specifies the SID of the gateway and matches the SID specified in the connect descriptor for the gateway in the `tnsnames.ora` file. Refer to "Configuring Oracle Net for the Oracle Integrating Server" on page 7-16 for more information about `tnsnames.ora`.

- **gateway** specifies the full path name of the gateway executable file.

- **log_file** specifies the full path name of the gateway bootstrap program log file.

- **mqseries_directory** specifies the directory where the MQSeries libraries (provided by IBM) are installed. If the libraries were not installed under the default directory of `/usr/lib`, then use `mqseries_directory` to specify the correct directory name.
Gateway Running in MQSeries Client Configuration

GATEWAY_SID=gateway_sid
SERVER_PATH=gateway
LOG_DESTINATION=log_file
LD_LIBRARY_PATH=mqseries_directory
MQSERVER=channel
MQCCSID=character_set

where:

-gateway_sid- specifies the SID of the gateway and matches the SID specified in the connect descriptor for the gateway in the tnsnames.ora file. Refer to "Configuring Oracle Net for the Oracle Integrating Server" on page 7-16 for more information about tnsnames.ora.

-gateway- specifies the full path name of the gateway executable file.

-log_file- specifies the full path name of the gateway bootstrap program log file.

-mqseries_directory- specifies the directory where the MQSeries libraries (provided by IBM) are installed. If the libraries were not installed under the default directory of /usr/lib, then use mqseries_directory to specify the correct directory name.

-channel- specifies the location of the MQSeries server and the communication method to use. The channel format is: channel_name/connection_type/hostname [(port_number)].

For example:

MQSERVER=CHAN9/TCP/dolphin(1425)

Refer to IBM publications for more information about the MQSERVER environment variable.

Note: The channel name and connection type must entered in CAPITAL LETTERS.
Configuring the Gateway

Step 3: Customize the Gateway Initialization File

The Gateway Initialization File (init\sid.ora) supports all procedural gateway initialization parameters described in the Oracle Open Gateway Guide for SQL-Based and Procedural Gateways and in Appendix C, “Gateway Initialization Parameters”. The initialization file must be available when the gateway is started.

During installation, a default initialization file is created in $ORACLE_HOME/pg4mqseries/admin/init\sid.ora, where \sid is the default SID of pg4mq92 or pg4mqc92. If you chose a SID other than the default, then rename this file using the SID you chose in Step 1: Choose a system ID for the gateway. Customize the default initialization file as necessary.

The following entries might appear in an initialization file:

```
SET LOG_DESTINATION=log_file
SET HS_DB_NAME=database_name
SET HS_DB_DOMAIN=domain_name
SET QUEUE_MANAGER=manager_name
SET AUTHORIZATION_MODEL=auth_model
SET TRANSACTION_MODEL=tx_model
SET TRANSACTION_LOG_QUEUE=tx_queue_name
SET TRANSACTION_RECOVERY_USER=rec_user
SET TRANSACTION_RECOVERY_PASSWORD=rec_password
SET TRACE_LEVEL=0
```

A \character_set specifies the coded character set number used by the gateway when communicating with the MQSeries queue manager. This is an optional parameter.

This parameter is set only if the computer that is running the MQSeries queue manager uses a different encoding scheme than the computer that runs the gateway. When set, the value of \character_set is used by the MQSeries client software on the gateway’s computer to convert the data.

The value specified for \character_set affects communication only between the gateway and the MQSeries queue manager. The message data sent or received by your application is not affected.

Refer to IBM publications for more information.
Note: Refer to Oracle9i Net Services Administrator’s Guide and Oracle9i Net Services Reference Guide for additional information.

where:

`log_file` specifies the full path name of the gateway log file.

`database_name` is only used when the Oracle integrating server initialization parameter `GLOBAL_NAMES` is set to `TRUE`. In this case, the value of `database_name` is the name of the database link that was created in the Oracle integrating server to connect to the gateway. It is 1 to 8 characters long.

`domain_name` is only used when the Oracle integrating server initialization parameter `GLOBAL_NAMES` is set to `TRUE`. In this case, the value of `domain_name` is the domain name of the database link that was created in the Oracle integrating server to connect to the queue manager.

`manager_name` is the name of the MQSeries queue manager to access.

`auth_model` is the authorization model to use. The default is `RELAXED`. Refer to Chapter 8, "Gateway Running Environment" for more information about security models.

`tx_model` is the transaction model to use. The default is `SINGLE_SITE`. Refer to Chapter 8, "Gateway Running Environment" for more information about transaction models.

`tx_queue_name` is the name of the queue for logging transaction IDs for distributed transactions. This is only used when `tx_model` is set to `COMMIT_CONFIRM`. Refer to Chapter 8, "Gateway Running Environment" for more information about transaction models.

`rec_user` specifies the user name that the gateway uses to start recovery of a distributed transaction. This is only used when `auth_model` is set to `STRICT` and `tx_model` is set to `COMMIT_CONFIRM`. Refer to Chapter 8, "Gateway Running Environment" for more information about security models.
*rec_password* specifies the password of the user name that the gateway uses to start recovery of a distributed transaction. Refer to "TRANSACTION_RECOVERY_PASSWORD" on page C-8 for more password information and to Chapter 8, "Gateway Running Environment" for more information about security models.
Configuring Oracle Net for the Gateway

The gateway requires Oracle Net to provide transparent data access to and from the Oracle integrating server. Oracle Net uses the TNS listener to receive incoming connections from an Oracle Net client. In the case of the gateway, the TNS listener listens for incoming requests from the Oracle integrating server. For the TNS listener to listen for the gateway, information about the gateway must be added to the TNS listener configuration file (listener.ora). This file, by default, is located in the directory `ORACLE_HOME/network/admin`, where `ORACLE_HOME` is the directory under which the gateway is installed. The default values in this file are set for you during the installation process by the Oracle Universal Installer.

Using Oracle Net with Gateway Default Values

If you are configuring one gateway instance, and if you have no need to change any of the default values, then no further configuration is necessary for Oracle Net regarding the gateway. Perform only "Step 2: Stop and start the TNS listener for the gateway" on page 7-14.

Using Oracle Net When Changing the Gateway Default Values

If you intend to use the Oracle Net listener for multiple gateway instances, or if you need to modify some of the default values set during the installation phase, then perform Step 1 and Step 2 in this section.

In Step 1, you add gateway information or change default information in the `listener.ora` file in the gateway directory `ORACLE_HOME/network/admin`.

Step 1: Configure the Oracle Net TNS listener for the gateway

Two entries must be added to the `listener.ora` file:

- a list of Oracle Net addresses for the TNS listener to listen on
- the gateway process that the TNS listener should start in response to incoming connection requests
Entry of Oracle Net Addresses for the TNS Listener

If you are using Oracle Net and the TCP/IP protocol adapter, then the syntax of the entry in the listener.ora file is:

```
LISTENER=
  (ADDRESS_LIST=
    (ADDRESS=
      (PROTOCOL=TCP)
      (HOST=host_name)
      (PORT=port_number)
    )
  )
```

where:

- `host_name` is the name of the computer where the gateway is installed.
- `port_number` specifies the IP port number used by the TNS listener. If you have other listeners running on `host_name`, such as the listener of an Oracle integrating server on the same computer, then the value of `port_number` must be different from the other listeners’ port numbers.

If you are using Oracle Net and the interprocess socket call (IPC) protocol adapter, the syntax of the entry in listener.ora file is:

```
LISTENER=
  (ADDRESS_LIST=
    (ADDRESS=
      (PROTOCOL=IPC)
      (KEY=key_name)
    )
  )
```

**Note:** The TNS listener and the gateway must reside on the same node. If you already have a TNS listener up and running on the node, then you may want to make the following changes (in Steps 1 and 2) to your existing listener.ora and tnsnames.ora files.

After making the changes, you can reload the changes by running the "reload" subcommand in the "lsnrctl" utility without shutting down the TNS listener.
where:

IPC specifies that the protocol used for connections is IPC.

key_name is the unique user-defined service name.

**Entry for the Gateway**

To direct the TNS listener to listen for a gateway instance in response to incoming connection requests, add an entry to the listener.ora file using the following syntax:

```plaintext
SID_LIST_LISTENER=
 (SID_LIST=
  (SID_DESC=
   (SID_NAME=gateway_sid)
   (ORACLE_HOME=gateway_directory)
   (PROGRAM=driver)
  )
 )
```

where:

**gateway_sid** specifies the SID of the gateway and matches the gateway SID specified in the connect descriptor entry in the tnsnames.ora file. Refer to "Configuring Oracle Net for the Oracle Integrating Server" on page 7-16.

**gateway_directory** specifies the gateway directory where the gateway software resides.

**driver** this is the name of the executable file that initializes the gateway environment using the boot file and that starts the gateway. If the gateway uses a local MQSeries server, then the file name is pg4mq92drv; the file name is pg4mqc92drv if the gateway is run as an MQSeries client to access a remote MQSeries server.
When adding an entry for multiple gateway instances, add the entry to the existing `SID_LIST` syntax:

```
SID_LIST_LISTENER=
  (SID_LIST=
    (SID_DESC=.
      .
      .
    )
    (SID_DESC=.
      .
      .
    )
    (SID_DESC=
      (SID_NAME=gateway_sid)
      (ORACLE_HOME=gateway_directory)
      (PROGRAM=driver)
    )
  )
```

The following is an example of an entry made to the `listener.ora` file:

```
(SID_DESC =
  (SID_NAME=pg4mqs92)
  (ORACLE_HOME=/oracle/app/oracle/product/pg4mqs92)
  (PROGRAM=pg4mqs92drv)
)
```

Refer to Oracle9i Net Services Administrator’s Guide and Oracle9i Net Services Reference Guide for more information about changing `listener.ora`.
Step 2: Stop and start the TNS listener for the gateway

The TNS listener must be started or reloaded to initiate the new settings.

- Set the gateway directory name:
  If you are using the Bourne or Korn shell, then enter:

  ```sh
  $ ORACLE_HOME=gateway_directory;export ORACLE_HOME
  ```

  If you have the C shell, then enter:

  ```sh
  $ setenv ORACLE_HOME gateway_directory
  ```

  where `gateway_directory` specifies the directory where the gateway software is installed.

- If the listener is already running, then use the `lsnrctl` command to reload the listener with the new settings:

  ```sh
  $ cd $ORACLE_HOME/bin
  $ ./lsnrctl reload your_listener_name
  ```

  where `ORACLE_HOME` specifies the directory where the gateway software is installed.

- Check the status of the listener with the new settings:

  ```sh
  $ ./lsnrctl status your_listener_name
  ```

Note: If you already have a TNS listener up and running on the Oracle integrating server where the gateway is installed, then you may want to make changes to your existing `listener.ora` and `tnsnames.ora` files. After making the changes, you can reload the changes by running the "reload" subcommand in the "lsnrctl" utility without shutting down the TNS listener.

Refer to the Note on page 7-11 after "Step 1: Configure the Oracle Net TNS listener for the gateway".
The following is an example of output from a `lsnrctl` status check:

Connecting to (ADDRESS=(PROTOCOL=IPC)(KEY=ORAIPC))
STATUS of the LISTENER
------------------------
Alias            LISTENER
Version           TNSLSNR for Solaris: version 9.2.0.1.0 - Production
Start Date         21-AUG-99 18:16:10
Uptime           0 days 0 hr. 2 min. 19 sec
Trace Level       off
Security          OFF
SNMP              OFF
Listener Parameter File   /oracle/app/oracle/product/pg4mqs901/network/admin/listener.ora
Listener Log File      /oracle/app/oracle/product/pg4mqs901/network/log/listener.log
Services Summary...
  pg4mqs90       has 1 service handler(s)
The command completed successfully

In the example above, `pg4mqs90` is the default SID value that was assigned during installation. You can use any valid ID for the SID, or keep the default.

---

**Note:** You must use the same SID value in the `tnsnames.ora` file, the `listener.ora` file, and the `GATEWAY_SID` environment variable in the gateway boot file for each gateway instance being configured.
Configuring Oracle Net for the Oracle Integrating Server

Any Oracle application that has access to an Oracle integrating server can also access MQSeries through the gateway. Before you use the gateway to access MQSeries, you must configure the Oracle integrating server so that it can communicate with the gateway over Oracle Net. To configure the server, add connect descriptors to the `tnsnames.ora` file.

Any Oracle integrating server that accesses the gateway needs a service name entry or a connect descriptor name entry in its `tnsnames.ora` file to tell the Oracle integrating server how to make connections. This file, by default, is located in the directory `ORACLE_HOME/network/admin`, where `ORACLE_HOME` is the directory in which the Oracle integrating server is installed. The `tnsnames.ora` file is required by the Oracle integrating server that is accessing the gateway, not by the gateway itself. Refer to Oracle9i Net Services Administrator’s Guide and Oracle9i Net Services Reference Guide for more information about changing `tnsnames.ora`. Refer to "Configuration Overview" and Figure 7–1, "Gateway Communication" on page 7-2 and to "Configuring the Gateway" on page 7-3.

Using Gateway Default Values

The Oracle Universal Installer creates and pre-configures a `tnsnames.ora` file in the directory `ORACLE_HOME/network/admin`, where `ORACLE_HOME` is the directory where the gateway software is installed. If you use the default values configured for you, and if you do not need to configure for additional gateway instances, then the contents of this file can be appended to the `tnsnames.ora` file of each Oracle integrating server that accesses the gateway.

Changing Gateway Default Values

If you need to change some of the default settings, use the examples in this section to guide you.
TCP/IP Example

An Oracle integrating server accesses the gateway using Oracle Net and the TCP/IP protocol adapter. The syntax of the connect descriptor entry in tnsnames.ora is:

\[
\text{tns_name_entry} = \\
\text{(DESCRIPTION=}
\text{)}
\text{(ADDRESS=}
\text{)}
\text{(PROTOCOL=TCP)}
\text{(HOST=host_name)}
\text{(PORT=port_number)}
\text{)
\text{(CONNECT_DATA=}
\text{(SID=gateway_sid)}
\text{)
\text{(HS=OK)}
\text{)}
\]

where:

- \text{tns_name_entry} is the \text{tns_name_entry} of the CREATE DATABASE LINK statement. Refer to "Creating Database Links" on page 7-20 for more information.
- \text{TCP} specifies that the protocol used for connections is TCP/IP.
- \text{port_number} matches the port number used by the Oracle Net TNS listener that is listening for the gateway. The TNS listener’s port number can be found in the listener.ora file that is used by the TNS listener. Refer to "Entry of Oracle Net Addresses for the TNS Listener" on page 7-11.
- \text{host_name} specifies the computer on which the gateway is running. The TNS listener’s host name can be found in the listener.ora file used by the TNS listener that is listening for the gateway. Refer to "Entry of Oracle Net Addresses for the TNS Listener" on page 7-11.
- \text{gateway_sid} specifies the SID of the gateway and matches the SID specified in the listener.ora file of the TNS listener that is listening for the gateway.
IPC Example

An Oracle integrating server accesses the gateway using Oracle Net and the IPC protocol adapter. The syntax of the connect descriptor entry in `tnsnames.ora` is:

```plaintext
tns_name_entry =
  (DESCRIPTION =
    (ADDRESS =
      (PROTOCOL = IPC)
      (KEY = key_name))
    (CONNECT_DATA =
      (SID = gateway_sid))
    (HS = OK)
  )
```

where:

- `tns_name_entry` is the `tns_name_entry` of the `CREATE DATABASE LINK` statement. Refer to "Creating Database Links" on page 7-20 for more information.
- `IPC` specifies that the protocol used for connections is IPC.
- `key_name` is the service name.
- `gateway_sid` specifies the SID of the gateway and matches the SID specified in the `listener.ora` file of the TNS listener that is listening for the gateway.
Creating a Transaction Log Queue

When the TRANSACTION_MODEL parameter in the gateway initialization file is set to COMMIT_CONFIRM to allow for distributed transactions, then an additional configuration step is required to:

- create an MQSeries queue
- set the TRANSACTION_LOG_QUEUE, TRANSACTION_RECOVERY_USER and TRANSACTION_RECOVERY_PASSWORD parameters in the gateway initialization file

Refer to "Commit-Confirm" on page 8-8 for more information about the commit-confirm transaction model and Appendix C, "Gateway Initialization Parameters" for information about TRANSACTION_LOG_QUEUE, TRANSACTION_RECOVERY_USER, and TRANSACTION_RECOVERY_PASSWORD. Refer to IBM publications for information about creating and configuring a queue.

For the gateway to recover distributed transactions, a recovery account and queue must be set up in the queue manager by the MQSeries system administrator. The user name of the account must be a valid MQSeries user which has authorization to access the recovery queue. Refer to "Authorization for MQSeries Objects" on page 8-4 for more information about access privileges.

The gateway uses the recovery queue to check the status of failed transactions that were started at the queue manager by the gateway and were logged in this queue. The information in this queue is vital to the recovery process and must not be used, accessed, or updated except by the gateway.

Administering Database Links

A connection to the gateway is established through a database link when it is first used in an Oracle session. In this context, "connection" refers to the connection between the Oracle integrating server and the gateway. The connection remains established until the Oracle session ends. Another session or user can access the same database link and get a distinct connection to the gateway and the queue manager.

Database links are active for the duration of a gateway session. To close a database link during a session, use the ALTER SESSION statement. For more information about using database links, refer to the Oracle9i Database Administrator’s Guide.
Creating Database Links

To create a database link, use the CREATE DATABASE LINK statement. The USING clause points to a connect descriptor in the tnsnames.ora file.

The CONNECT TO clause specifies the MQSeries user ID and password when the security model is defined as STRICT with the AUTHORIZATION_MODEL parameter. If you do not include the CONNECT TO clause, then the current user ID and password are used.

When the AUTHORIZATION_MODEL parameter is set to RELAXED, you do not specify a user ID and password because the Oracle integrating server uses the user ID and password of the user who started the TNS listener for the gateway. If you attempt to specify a user ID and password with the CONNECT TO clause, then the Oracle integrating server and gateway ignore those values. Refer to "Security Models" on page 8-2 for more information.

The syntax of CREATE DATABASE LINK is:

```
CREATE [PUBLIC] DATABASE LINK dblink [CONNECT TO userid IDENTIFIED BY password] USING 'tns_name_entry';
```
where:

\( \text{dblink} \) is the database link name.

\( \text{userid} \) is the user ID used to establish a session at the queue manager. It is only used when AUTHORIZATION_MODEL is set to STRICT in the init\( \text{sid} \).ora file. The user ID must be authorized:

- to access all MQSeries objects referenced in the PL/SQL commands.
- to use any database object referenced in the PL/SQL commands.

The \( \text{userid} \) must be in the password file on the computer where MQSeries and the gateway are installed. Or, the \( \text{userid} \) must be published in the UNIX Network Information Service (NIS) when MQSeries and the gateway are installed on different computers. If \( \text{userid} \) contains lowercase letters or non-alphanumeric characters, then surround \( \text{userid} \) with double quote characters (".

\( \text{password} \) is the password used to establish a session at the queue manager. It is only used when AUTHORIZATION_MODEL is set to STRICT in the init\( \text{sid} \).ora file.

The \( \text{password} \) must be in the password file on the computer where MQSeries and the gateway are installed. Or, the \( \text{password} \) must be published in the UNIX Network Information Service (NIS) when MQSeries and the gateway are installed on different computers.

If \( \text{password} \) contains lowercase letters or non-alphanumeric characters, then surround \( \text{password} \) with double quote characters (".

\( \text{tns\_name\_entry} \) is the Oracle Net TNS connect descriptor name specified in the tnsnames.ora file.
Dropping Database Links

You can drop a database link with the DROP DATABASE LINK statement. For example, to drop the database link named `dblink`, enter:

```
DROP [PUBLIC] DATABASE LINK dblink;
```

A database link should not be dropped if it is still required to resolve an in-doubt distributed transaction. Refer to the Oracle9i Database Administrator’s Guide for more information about dropping database links.

Examining Available Database Links

The data dictionary of each database stores the definitions of all the database links in that database. The USER_DB_LINKS view shows the database links that are defined for a user. The ALL_DB_LINKS data dictionary views show all defined database links.

Limiting the Number of Active Database Links

You can limit the number of connections from a user process to remote databases with the parameter OPEN_LINKS. This parameter controls the number of remote connections that any single user process can use with a single user session. Refer to the Oracle9i Database Administrator’s Guide for more information about limiting the number of active database links.

Installing the Visual Workbench Repository

Install the Visual Workbench repository following the steps in this section.

You may skip installing the Visual Workbench Repository if you do not plan to use the Visual Workbench or if you are preparing your production Oracle server where you do not need a Visual Workbench Repository, but instead need a PG4MQ deployment. Refer to the "Preparing the Production Oracle Server" on page 7-28 for the details.

Preinstallation Tasks

Step 1: Choose a repository server

A repository server is an Oracle integrating server on which the Visual Workbench repository is installed.
Step 2: Locate the installation scripts

The Visual Workbench repository installation scripts are installed with the Visual Workbench. If the repository is to be installed on the same computer as Visual Workbench, then your repository server already has all the required installation scripts. Proceed to step 3.

1. Create a directory on the repository server that is to be the script directory. For example:
   
   ```bash
   $ mkdir $ORACLE_HOME/pg4mqseries/admin/repo
   $ chmod 777 $ORACLE_HOME/pg4mqseries/admin/repo
   ```

2. Use a file transfer program to transfer the repository zip file (`repos XXX.zip`, where `XXX` is the release number) or move all script files with the suffix `.sql` from the script file directory (`NT=%ORACLE_HOME\pg4mqvwb\server\admin`) on the Visual Workbench computer to the script file directory on the repository server computer.

Step 3: Ensure the UTL_RAW package is installed

All data mapping packages generated by the Visual Workbench use the UTL_RAW package, which provides routines for manipulating raw data.

From SQL*Plus, as user SYS, issue the following statement:

```
SQL> DESCRIBE UTL_RAW
```

If the DESCRIBE statement is successful, then your repository server already has UTL_RAW installed, and you can proceed to step 4.

If the DESCRIBE statement fails, then install UTL_RAW:

From SQL*Plus, as user SYS, run the `utlrw.sql` and `prvtrawb.plb` scripts that are in directory `ORACLE_HOME/rdbms/admin`. You must run the `utlrw.sql` script first.

```
SQL> @utlrw.sql
SQL> @prvtrawb.plb
```
Step 4: Ensure the DBMS_OUTPUT package is enabled

The sample programs and installation verification programs on the distribution CD-ROM use the standard DBMS_OUTPUT package.

From SQL*Plus, as user SYS, issue the following statement:

```
SQL> DESCRIBE DBMS_OUTPUT
```

If the DESCRIBE statement is successful, then your repository server has DBMS_OUTPUT installed, and you can proceed to Step 5.

If the DESCRIBE statement fails, then install DBMS_OUTPUT. Refer to your Oracle server DBA.

Step 5: Create a database link

Create a database link on your Oracle Production System Server to access the Oracle Procedural Gateway® for IBM MQSeries.

If you do not already have a database link, then refer to "Administering Database Links" on page 7-19 for more information on creating database links.

Visual Workbench Repository Installation Tasks

Use pgvwbrepos9.sql to install the Visual Workbench Repository on Oracle9i. To run pgvwbrepos9.sql, ensure you are currently in directory

```
ORACLE_HOME/pg4mqseries/admin/repo
```

and then enter:

```
sqlplus /nolog @pgvwbrepos9.sql
```

Note: If you are installing the Visual Workbench Repository on Oracle8i or prior, then you need to use pgvwbrepos8.sql. All of the examples in this section are provided with the assumption that you are installing on Oracle9i.

The script takes you through the following steps:
Step 1: Enter the database connection information
Use the default of LOCAL by pressing Return.

Next you are prompted to enter the passwords for the SYSTEM and SYS accounts of the Oracle integrating server. Press Return after entering each password.

The script stops if any of the information is incorrect. Verify the information before rerunning the script.

Step 2: Check for existing Workbench Repository
The script checks for an existing Visual Workbench repository and for the data dictionary. If neither one is found, the script proceeds to Step 3 below.

If the data dictionary exists, then the script stops. Choose another Oracle integrating server and rerun the script, starting at "Step 1: Choose a repository server" on page 7-22.

If a Visual Workbench repository exists, then the script gives you the following options:
A. Upgrade the existing private repository to public status and proceed to Step 3 below.
B. Replace the existing repository with the new private repository and proceed to Step 3 below.
C. Stop the script.

Step 3: Check for required PL/SQL packages
The script checks for the existence of UTL_RAW, DBMS_OUTPUT, and DBMS_PIPE in the Oracle integrating server. If this software exists, then the script proceeds to Step 4 below.

The script stops if this software does not exist. Refer to your Oracle integrating server DBA about the missing software. After the software is installed, rerun the script.

Step 4: Install the UTL_PG package
The script checks for the existence of the UTL_PG package. If it does not exist, then the UTL_PG package is installed. The script proceeds to Step 5 below.

If UTL_PG exists, then you are prompted to reinstall it. Press Return to reinstall UTL_PG.
Step 5: Create the admin user and all repository tables

This step creates the administrative user for the Visual Workbench repository as PGMADMIN with an initial password of PGMADMIN. This user owns all objects in the repository.

After this step, a private Visual Workbench repository, which includes the PGM_SUP, PGM_BQM, and PGM_UTL8 packages, is created in the Oracle integrating server, which only the user PGMADMIN can access.

Step 6: Create public synonyms and development roles

This is an optional step to change the private access privileges of the Visual Workbench repository. The private status allows only the PGMADMIN user to have access to the repository. If you enter N and press Return, then the repository retains its private status.

A public status allows the granting of access privileges to other users besides PGMADMIN. If you want to give the repository public status, then enter Y and press Return.

After the Repository is Created

After creating the Visual Workbench repository, there is one optional step:

Grant development privileges for the Visual Workbench repository to users.

To allow users other than PGMADMIN to perform development operations on the Visual Workbench repository, PGMADMIN must grant them the necessary privileges. To do this, perform the following:

- Ensure the repository has a public status. It has this status if you created it by using Steps 1 through 6 of the pgvwbrepos9.sql script. If you did not use Step 6, then rerun the script. When you get to Step 2 of the script, enter "A" to the prompt to upgrade the private repository to public status.

- Use SQL*Plus to connect to the repository as user PGMADMIN and grant the PGMDEV role to each user. For example:

  SQL> GRANT PGMDEV TO SCOTT;
Deinstall the Visual Workbench Repository

To deinstall a Visual Workbench repository on Oracle9i, use the repository script pgvwbremove9.sql. To run this script, ensure that you are currently under the Oracle integrating server directory ORACLE_HOME/pg4mqseries/admin/repo (where you copied the scripts), and then enter:
sqlplus /nolog @pgvwbremove9.sql

**Note:** If you are deinstalling the Visual Workbench Repository on Oracle8i or prior, then you need to use pgvwbremove8.sql. All of the examples in this section are provided with the assumption that you are installing on Oracle9i.

The script takes you through the following steps:

**Step 1: Enter the database connection information**
Use the default of LOCAL by pressing Return.

Next you are prompted to enter the passwords for the SYSTEM, SYS, and PGMADMIN accounts of the Oracle integrating server. Press Return after entering each password.

The script stops if any of the information is incorrect. Verify the information before rerunning the script.

**Step 2: Check for existing Workbench repository**
Enter "Y" and press Return for the prompt to remove public synonyms and development roles. This returns the repository to private status. You can exit the script now by entering "N" and pressing Return, or you can proceed to the next prompt under this step.

If you are certain you want to remove the private repository, then enter "Y" and press Return. The script removes all repository tables and related packages.
Preparing the Production Oracle Server

These preparations include preparing, installing, and removing PL/SQL packages.

Introduction

Before you can compile MIPs on a production Oracle server, the following PL/SQL packages must be present on the production Oracle server:

- DBMSPIPE, DBMS_OUTPUT, and UTL_RAW
  These packages are shipped with each Oracle server and are usually already installed.

- PGM8, PGM_BQM, PGM_SUP, and UTL_PG
  These packages are shipped with your Oracle Procedural Gateway for message queuing. They are installed during the creation process of the Visual Workbench repository. Do not execute deployment script on the Oracle server with an installed Visual Workbench repository. If the Oracle server used for the repository is different than the Oracle server used in the production environment, you must install these packages on the production Oracle server.

This section describes how to run:

- pgmdeploy9.sql, a deployment script to verify the existence of the required PL/SQL packages and install some of them if they do not exist on the production Oracle server

- pgmundeploy9.sql, a script to remove the PL/SQL packages from a production Oracle server

**Note:** If your production Oracle server is Oracle8i or prior, you need to use pgmdeploy8.sql to install Oracle9i PG4MQ deployment packages, and you need to use pgmundeploy8.sql to remove Oracle9i PG4MQ deployment packages.

All of the examples in this section are provided with the assumption that you are installing on Oracle9i.
Verifying and Installing PL/SQL Packages

1. Locate the necessary scripts:

   - pgm8.sql
   - pgmbqm8.sql
   - pgmdeploy8.sql
   - pgmdeploy9.sql
   - pgmsup8.sql
   - pgmundeploy8.sql
   - pgmundeploy9.sql
   - prvtpg.sql
   - utlpq.sql

   **Note:** pgm8.sql and other script names contain numbers such as 8 and 9. These numbers are not product version numbers.

These scripts are installed with the gateway, in the directory ORACLE_HOME/pg4mqseries/admin/deploy, where ORACLE_HOME is the gateway home directory.

2. If your production Oracle server is on a different computer than the gateway, you need to use a file transfer method, such as ftp, to transfer files in the directory ORACLE_HOME/pg4mqseries/admin/deploy, where ORACLE_HOME is the gateway home directory on your gateway computer. On your production Oracle server computer, change directory to the directory containing the deployment scripts you just transferred and skip to step 4.

3. If your production Oracle server is on the same computer as the gateway, then change directory to ORACLE_HOME/pg4mqseries/admin/deploy, where ORACLE_HOME is the gateway home directory.

4. Run the deployment script by entering:

   `$ sqlplus /nolog @pgmdeploy9.sql`

5. At the script prompt: Enter the connect string for the Oracle server... [LOCAL], press [Return] to use the default of LOCAL.
Preparing the Production Oracle Server

6. At the script prompt Enter the following required Oracle server password, enter the password of the SYS account.

After the script verifies the SYS account password, it connects to the production Oracle server. The script verifies and reports on which PL/SQL packages are installed there:

- If any of the Oracle server packages DBMS_OUTPUT, DBMS_PIPE or UTL_RAW are missing, the script stops. Have your DBA install the missing packages and re-run the deployment script.
- If any of the Oracle packages PGM8, PGM_BQM, PGM_SUP, and UTL_PG are missing, the script installs them on the production Oracle server.

Removing the PL/SQL Packages

You can remove the PL/SQL packages that were installed by the pgmdeploy9.sql script if, for example, none of your applications in the production environment uses a MIP. To remove these packages, perform the following steps:

1. On your production Oracle server computer, change directory to the directory containing the deployment scripts by entering the following command:

   $ cd $ORACLE_HOME/pg4mqseries/admin/deploy

2. Run the script by entering:

   $ sqlplus /nolog @pgmundeploy9.sql

3. At the script prompt: Enter the connect string for the Oracle server... [LOCAL], press [Return] to use the default of LOCAL.

4. At the script prompt, enter the required Oracle server passwords, enter the password of the SYS account.

After the script verifies the SYS account password, it connects to the production Oracle server and removes the packages installed by the pgmdeploy9.sql script. After the pgmundeploy9.sql script completes successfully, applications on the production Oracle server fail if they attempt to reference any of the MIP’s that are compiled there.
This chapter explains the following aspects of the gateway running environment:

- Security Models on page 8-2
- Transaction Support on page 8-5
- Troubleshooting on page 8-9
Security Models

MQSeries has its own authorization mechanism. Applications are allowed to perform certain operations on queues or queue managers only when their effective user ID has authorization for each operation. The effective user ID, typically the operating system user, depends on the MQSeries environment and the platform it runs on.

The effective user ID in an Oracle environment is not dependent on an operating system account or the platform. Because of this difference, the gateway provides two authorization models for Oracle applications to work with MQSeries:

- relaxed
- strict

Although Oracle and operating system user IDs can be longer than 12 characters, the length of user IDs used for either model cannot exceed 12 characters. Oracle user accounts do not have a minimum number of characters required for their passwords, but some platforms and operating systems do. Take their requirements into consideration when deciding on a password or user ID.

The authorization model is configured with the AUTHORIZATION_MODEL parameter in the gateway initialization file. Refer to Appendix C, "Gateway Initialization Parameters" for more information about AUTHORIZATION_MODEL.

Relaxed Model

This model discards the Oracle username and password. The authorizations granted to the effective user ID of the gateway by the queue manager are the only associations an Oracle application has. For example, if the gateway’s user ID is granted permission to open or read messages, or place messages on a queue, then all Oracle applications that access the gateway can request those operations.

The effective user ID is determined by how the gateway is running:

- If the gateway runs as an MQI client application, then the user ID is determined by the MQI channel definition (refer to IBM publications for more information about channel definitions)
- If the gateway runs as an MQI server application, then the effective user ID of the gateway is the user who started the Oracle Net listener and who must have authorization to all the MQSeries objects that the Oracle application wants to access. Refer to "Authorization for MQSeries Objects" on page 8-4 for more information
Oracle Corporation recommends against using the relaxed model unless your application has minimal security requirements.

**Strict Model**

This model uses the Oracle user ID and password provided in the Oracle CREATE DATABASE LINK statement when a database link is created, or the current Oracle user ID and password if none was provided with CREATE DATABASE LINK.

The Oracle user ID:

- should match a user account for the computer that runs the gateway and for the computer that runs the MQSeries queue manager
- must have authorization for all the accessed MQSeries objects. Refer to "Authorization for MQSeries Objects" on page 8-4 for more information

The authorization process to verify the Oracle user ID and password varies depending on how the gateway is running.

**The Authorization Process for an MQSeries Server Application**

If the gateway runs as an MQSeries server application, then the authorization process checks the user ID and password against the local or network password file. If they match, then the gateway performs a SET-UID for the user ID and continues to run under this user ID. Further MQSeries authorization checks happen for this user ID.

**The Authorization Process for an MQSeries Client Application**

If the gateway runs as an MQSeries client application, then the authorization process checks the user ID and password against the local or network password file. If they match, then the MQSeries environment variables MQ_USER_ID and MQ_PASSWORD are set to the values of the user ID and password. If the channel definition has specified the MQSeries environment variable MCAUSER as blank characters, then MQSeries authorization checks occur for the user ID.

If MCAUSER is set, not set, or security exits are defined for the MQI channel, then these override the gateway efforts. Refer to IBM publications for more information about MQSeries environment variables.
Authorization for MQSeries Objects

The effective user ID for the relaxed model and the Oracle user ID for the strict model require the following MQSeries authorizations:

Table 8–1  MQ Series Access Authorization

<table>
<thead>
<tr>
<th>Type of Access</th>
<th>MQSeries Authorization Keywords</th>
<th>Alternate MQSeries Authorization Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permission to access the MQSeries queue manager</td>
<td>all or allmqi</td>
<td>connect</td>
</tr>
<tr>
<td>Permission to send messages to an MQSeries queue</td>
<td>all or allmqi</td>
<td>passall</td>
</tr>
<tr>
<td>Permission to receive messages from an MQSeries queue</td>
<td>all or allmqi</td>
<td>passid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>put</td>
</tr>
<tr>
<td></td>
<td></td>
<td>setid</td>
</tr>
</tbody>
</table>

Refer to IBM publications for more information about MQSeries authorizations.
Transaction Support

For transactions that invoke MQSeries message queue operations and that are using the gateway, such transactions from an Oracle application are managed by the Oracle transaction coordinator at the Oracle server where the transaction originates.

Non-Oracle Data Sources and Distributed Transactions

This section describes generic transaction support features of Oracle Server. When an Oracle distributed database contains a gateway, the gateway must be properly configured to take part in a distributed transaction. The outcome of a distributed transaction involving a gateway should be that all participating sites roll back or commit their parts of the distributed transaction. All participating sites, including gateway sites, that are updated during a distributed transaction must be protected against failure and must be able to take part in the two-phase commit mechanism. A gateway that updates a target system as part of a distributed transaction must be able to take part in the automatic recovery mechanism, which might require that recovery information be recorded in transaction memory at the target system.

If a SQL-based gateway is involved in a distributed transaction, the distributed database must be in a consistent state after the distributed transaction is committed.

A procedural gateway or a SQL-based gateway with the procedural option translates remote procedure calls into target system calls. From the viewpoint of the Oracle transaction model, the gateway appears to be an Oracle server executing a PL/SQL block containing SQL statements used to access an Oracle database.

For a procedural gateway, it is unknown whether a target system call alters data. To ensure the consistency of a distributed database, it must be assumed that a procedural gateway updates the target system. Accordingly, all remote procedure calls sent to a procedural gateway take part in a distributed transaction and must be protected by the two-phase commit protocol. For example, you could issue the following SQL*Plus statements:

```
EXECUTE REMOTE_PROC@FACTORY;
INSERT INTO DEBIT@FINANCE
ROLLBACK;
```

In the example, REMOTE_PROC is a remote procedure call to access a procedural gateway, DEBIT is an Oracle table residing in an Oracle database, and FACTORY and FINANCE are database links used to access the remote sites.
Transaction Capability Types

This section describes generic transaction support features of Oracle Server.

When gateways are involved in a distributed transaction, the transaction capabilities of the non-Oracle data source determine whether the data source can participate in two-phase commit operations or distributed transactions.

Depending on the transaction capabilities of the non-Oracle data source, it can be mapped to one of the following transaction types:

- **Read-only**
  During a distributed transaction, the gateway provides readonly access to the data source, so the gateway can only be queried. Read-only is used for target systems that use the presumed-commit model or do not support rollback mechanisms.

- **Single-site**
  During a distributed transaction, the target system is readonly (other sites can be updated) or the only site updated (can participate in remote transactions). Single-site is used for target systems that support rollback, commit, and presumed-abort, but cannot prepare or commit-confirm as they have no distributed transaction memory, the ability to remember what happened during and after a distributed transaction identified by its transaction ID.

- **Commit-confirm**
  The gateway is a partial partner in the Oracle transaction mode. During a distributed transaction in which it is updated, the gateway must be the commit point site. Commit-confirm is used for target systems that support rollback, commit, presumed-abort, and commit-confirm, but do not support prepare. The commit-confirm capability requires distributed transaction memory.

- **Two-phase commit**
  The gateway is a partial partner in the Oracle transaction model. During a distributed transaction, the gateway cannot be the commit point site.

  Two-phase commit is used for target systems that support rollback, commit, presumed-abort, and prepare, but do not support commit-confirm, because they have no distributed transaction memory.
Transaction Support

Transaction Capability Types of Procedural Gateway for IBM MQSeries

This section describes transaction capabilities of the Oracle Procedural Gateway for IBM MQSeries.

Transactions from an Oracle application (that invoke MQSeries message queue operations and that are using the gateway) are managed by the Oracle transaction coordinator at the Oracle server where the transaction originates. The procedural gateway for IBM MQSeries provides the following transaction types:

- single-site
- commit-confirm

**Single-Site**

Single-site is supported for all MQSeries environments and platforms. Single-Site means that the gateway can participate in a transaction only when queues belonging to the same MQSeries queue manager are updated. An Oracle application can select, but not update, data at any Oracle database within the same transaction that sends to, or receives a message from, an MQSeries queue. To update objects in the Oracle database, the transaction involving the MQSeries queue should first be committed or rolled back.

This default mode of the gateway is implemented using MQSeries single-phase, where the queue manager acts as the synchpoint coordinator.
**Commit-Confirm**

This is an enhanced version of single-site and is supported for all MQSeries environments and platforms. Commit-confirm means that the gateway can participate in transactions when queues belonging to the same MQSeries queue manager are updated and, at the same time, any number of Oracle databases are updated. Only one gateway with the commit-confirm model can join the distributed transaction because the gateway operates as the focal point of the transaction. To apply changes to queues of more than one queue manager, updates applied to a queue manager need to be committed before a new transaction is started for the next queue manager.

As with single-site, commit-confirm is implemented using MQSeries single-phase, but it requires a dedicated recovery queue at the queue manager to log the transaction ID. At commit time, the gateway places a message in this queue with the message ID set to the Oracle transaction ID. After the gateway calls the queue manager to commit the transaction, the extra message on the transaction log queue becomes part of the overall transaction. This makes it possible to determine the outcome of the transaction in case of system failure, allowing the gateway to recover a failed transaction. When a transaction successfully completes, the gateway removes the associated message from the queue.

The MQSeries administrator must create a reserved queue at the queue manager. The name of this queue is specified in the gateway initialization file with the TRANSACTION_LOG_QUEUE parameter. All Oracle users that access MQSeries through the gateway should have full authorization for this queue. The transaction log queue is reserved for transaction logging only and must not be used, accessed, or updated other than by the gateway. When a system failure occurs, the Oracle recovery process checks the transaction log queue to determine the recovery strategy.

Two gateway initialization parameters, TRANSACTION_RECOVERY_USER and TRANSACTION_RECOVERY_PASSWORD, are set in the gateway initialization file to specify the user ID and password for recovery purposes. When set, the gateway uses this user ID and password combination for recovery work. The recovery user ID should have full authorization for the transaction log queue.

Refer to Appendix C, "Gateway Initialization Parameters" for more information about configuring the gateway for commit-confirm.
Troubleshooting

The troubleshooting topics include messages, error codes, gateway tracing, and gateway operations.

Message and Error Code Processing

The gateway architecture includes a number of separate components. Any of these components can detect and report an error condition while processing PL/SQL code. An error condition can be complex, involving error codes and supporting data from multiple components. In all cases, the Oracle application receives a single Oracle error code upon which to act.

Error conditions are represented in three ways:

1. Errors from the Oracle integrating server

   Messages from the Oracle integrating server are in the format ORA-xxxxx or PLS-xxxxx, where xxxx is a code number. ORA-xxxxx is followed by text explaining the error code. Refer to the Oracle9i Database Error Messages for explanations of these errors.

   Example:

   PLS-00306: wrong number or types of arguments in call to 'MQOPEN'
   ORA-06550: line7, column 3:
   PL/SQL: Statement ignored

2. Errors from the gateway

   Most gateway error conditions are reported to your application using one of the gateway error codes in the range of ORA-00700 through ORA-00799, ORA-08502 through ORA-08599, or ORA-09100 through ORA-09199. If an error is detected by the Oracle integrating server, then the gateway message lines do not occur. For example, if the gateway cannot be accessed because of an Oracle Net or gateway installation problem, then the gateway message line is not present in the generated error.

   The message format is explained under "Interpreting Gateway Messages" on page 8-10. Refer to the Oracle Open Gateway Guide for SQL-Based and Procedural Gateways for an explanation of messages in these number ranges.
Example:
ORA-09119: Initialization file contains error.
ORA-00718: Value "3" illegal for parameter "TRACE_LEVEL"
ORA-02063: preceding 2 lines from PG4MQ

3. MQSeries errors
When possible, an MQSeries error code is converted to an Oracle error code. If
that is not possible, then the Oracle error ORA-08500 with the corresponding
MQSeries error code is returned. Refer to “Common Error Codes” on page 8-11
for more information.

Note: Because the Oracle integrating server distinguishes only
between a successful or failed outcome of all user operations, MQI
calls that return a warning are reported as a successful operation.

Example:
ORA-08500: The IBM MQSeries MQI call "MQOPEN" fails with reason code 2085
ORA-06512: at line 8

Interpreting Gateway Messages
Error codes are generally accompanied by additional message text, beyond the text
associated with the Oracle message number. The additional text includes details
about the error.

Gateway messages have the following format:
ORA-nnnnn: error_message_text
gateway_message_line
ORA-02063: preceding n lines from dblink

where:

nnnn is an Oracle error number.
error_message_text is the text of the message associated with the error.
gateway_message_line is additional messages generated by the gateway.
n is the total number of gateway message lines.
dblink is the database link name.
Most gateway messages exceed the 70 character message area in the Oracle SQLCA. Use SQLGLM or OERHMS in the programmatic and Oracle Call Interfaces that you use with the gateway to view the entire message. Refer to the *Programmer’s Guide to the Oracle Precompilers* for additional information about SQLGLM, and the *Oracle Call Interface Programmer’s Guide* for additional information about OERHMS.

**Common Error Codes**

The error conditions that are described in this section are common error conditions that an application might receive while using the gateway, but do not cover all error situations.

**ORA-01017: invalid username/password; logon denied**

*Cause:* invalid username/password

*Action:* logon denied

**ORA-08500: "The IBM MQSeries MQI call "MQCONN" fails with reason code 2035"**

*Cause:* The MQCONN call failed. The gateway could not complete the current operation. (Note: Reason codes are listed in IBM documentation.)

*Action:* If the gateway is configured for the relaxed security model, then use the IBM MQSeries administrative command interface to grant connect privileges to the user who started the Oracle Net listener. The connect privileges allow access to the MQSeries queue manager.

If the gateway is configured for the strict security model, then use the IBM MQSeries administrative command interface to grant connect privileges to the username specified in the CREATE DATABASE LINK statement. If no username was specified in the CREATE DATABASE LINK statement, then the privileges are granted to the current Oracle user ID. The connect privileges allow access to the MQSeries queue manager.

Refer to "Authorization for MQSeries Objects" on page 8-4 and to IBM publications for information to verify and set access privileges.
ORA-02047: cannot join the distributed transaction in progress

Cause: The gateway TRANSACTION_MODEL parameter is set to SINGLE_SITE, but your application attempted a distributed transaction that involves an MQSeries operation and at least one Oracle database operation.

Action: Configure the gateway for distributed transactions by:

- setting the TRANSACTION_MODEL parameter to COMMIT_CONFIRM
- creating a transaction log queue in the MQSeries queue manager
- setting the TRANSACTION_LOG_QUEUE, TRANSACTION_RECOVERY_USER, and TRANSACTION_RECOVERY_PASSWORD parameters with the correct information

Refer to Chapter 7, "Configuration" and Appendix C, "Gateway Initialization Parameters" for more information.

ORA-02054: transaction n.n.nn in doubt

Cause: A problem occurred with the distributed transaction identified by n.n.nn.

Action: Verify the following:

- A valid queue name is specified by the TRANSACTION_LOG_QUEUE parameter.
- If the gateway is configured for the relaxed security model, then the user who started the Oracle Net listener must have sufficient MQSeries privileges to send or receive MQSeries messages for the transaction log queue.
- If the gateway is configured for the strict security model, then the username specified in the CREATE DATABASE LINK statement or the current Oracle user ID must have sufficient MQSeries privileges to send or receive MQSeries messages for the transaction log queue.

Refer to IBM publications for information to verify and set access privileges.
ORA-08500: "The IBM MQSeries MQI call "call_name" fails with reason code mqi_code"

Cause: An MQI call to an MQSeries queue manager failed. The gateway could not complete the current operation.

Action: If call_name is MQOPEN and mqi_code is 2035, then do the following:

- If the gateway is configured for the relaxed security model, then use the IBM MQSeries administrative command interface to grant sufficient message privileges to the user who started the Oracle Net listener. These privileges allow the user to send and receive messages for the specified MQSeries queue. Refer to IBM publications for more information.

- If the gateway is configured for the strict security model, use the IBM MQSeries administrative command interface to grant message privileges to the username specified in the CREATE DATABASE LINK statement. If no username was specified in the CREATE DATABASE LINK statement, the privileges are granted to the current Oracle user ID. These privileges allow the user to send and receive messages for the specified MQSeries queue. Refer to IBM publications for more information.

If call_name is MQOPEN, and if mqi_code is 2085, then verify that the queue that is specified in the MQSeries profile exists at the MQSeries queue manager that you are trying to access and that the queue name is correctly spelled and in the correct letter case.

For mqi_codes other than 2035 and 2085, refer to IBM publications.

ORA-08501: "No message available on the queue"

Cause: The MQGET procedure was executed, but there was no message on the queue satisfying the selection criteria that was specified by the type definition MQMD.

Action: If this is an anticipated condition, then add exception handling code to the calling PL/SQL block for a NO_MORE_MESSAGES exception, as defined in the gateway PGM_SUP package. If this is not an anticipated condition, then check whether the message was successfully placed on the queue. Refer to "PGM_SUP Package" on page A-29 for information about PGM_SUP.
ORA-08502: "Strict authorization fails for user "username""

**Cause:** The gateway did not find a matching entry in the network or local password file for **username**.

**Action:** Verify that the gateway executables `pg4mqs90` and `pg4mqs90drv` (MQSeries server configuration) or `pg4mc90` and `pg4mc90drv` (MQSeries client configuration) in the directory `ORACLE_HOME/bin` have root and SET-UID permission by using the `ls` command as follows:

```
$ ls -l pg4mq*
```

The result should look similar to the following:

```
-rwsr-xr-x 1 root root 3437292 Feb 28 15:18 pg4mqs92
-rwsr-xr-x 1 root root 13352 Feb 28 15:18 pg4mqs92drv
```

If you did not get this result, then run the `root.sh` script (refer to "Running root.sh" on page 5-13).

Ensure that the username and password that are specified with the CREATE DATABASE LINK statement (or the current Oracle user ID and password if no username was specified with CREATE DATABASE LINK) match the user account and password on the computer where the gateway is running. The characters and letter case must match.

Refer to "Creating Database Links" on page 7-20 and "Strict Model" on page 8-3 for more information.

ORA-08503: "The specified user ID "userid" or its password exceeds n characters."

**Cause:** The specified user ID or password of the database link exceeds the maximum length of characters supported by MQSeries.

**Action:** Create or use another user ID to access MQSeries.

ORA-08504: "Cannot initiate strict authorization for user ID "userid"

**Cause:** The gateway could not initiate MQSeries authorization for **userid**.

**Action:** Verify that the gateway has the necessary privileges.
Gateway Tracing

The gateway has a trace feature for testing or debugging purposes. The trace feature collects information about the gateway running environment, MQI calls, and parameter values of MQI calls. How much trace data to collect is based on the tracing level selected with the TRACE_LEVEL parameter. Do not enable tracing when your application is running in a production environment because it reduces gateway performance. Refer to Appendix C, "Gateway Initialization Parameters" for more information about enabling tracing.

The trace data is written to the directory and file specified by the LOG_DESTINATION parameter.

LOG_DESTINATION
This is a gateway initialization parameter.

Gateway: SQL-based and procedural

Default value. The default depends on your system. See your gateway installation and user’s guide (?) for the default value.

Range of values None

Syntax. LOG_DESTINATION = log_file

Parameter Description

LOG_DESTINATION = log_file

LOG_DESTINATION specifies the filename or directory where the gateway writes logging information. When log_file already exists, logging information is written to the end of file.

If you do not specify LOG_DESTINATION, then the default log file is overwritten each time the gateway starts up.
Example

The following example shows the trace data collected for MQI calls and their parameter values when gateway tracing was enabled by setting TRACE_LEVEL to 6:

Calling MQCONN()

    Name = mgr.nlsu41.1
    pHconn = 273248

Returning from MQCONN()

    Name = mgr.nlsu41.1
    Hconn = 2849264
    CompCode = 0
    Reason = 0

Calling MQOPEN()

    Hconn = 2849264
    pObjDesc = efffed58
    ObjectName = QUEUE2
    Options = 16
    pHobj = 274468

Returning from MQOPEN()

    Hconn = 2849264
    pObjDesc = efffed58
    Options = 16
    Hobj = 239888
    CompCode = 0
    Reason = 0

Calling MQCMIT()

    Hconn = 2849264

Returning from MQCMIT()

    Hconn = 2849264
    CompCode = 0
    Reason = 0

Calling MQDISC()

    Hconn = 2849264

Returning from MQDISC()

    Hconn = -1
    CompCode = 0
Verifying Gateway Operation

If your application cannot connect to the gateway, then rerun the application with the gateway trace feature enabled. If no trace information is written to the log file specified by LOG_DESTINATION, or if the log file is not created at all, then verify that:

- the Oracle Net configuration for the gateway and the Oracle integrating server is set up properly (refer to “Configuring Oracle Net for the Gateway” on page 7-10)
- a database link between the Oracle integrating server and the gateway was created (“Administering Database Links” on page 7-19)

If the Oracle Net configuration and database link are set up correctly, then check the gateway’s operation with the test.sql script:

1. Change directory to the gateway sample directory by entering:
   
   $ cd $ORACLE_HOME/pg4mqseries/sample

2. Using an editor, modify the test.sql script to:
   
   a. Specify the database link name that you created for the gateway. To do this, replace the characters @pg4mq with @dblink, where dblink is the name you chose when the database link was created.
   
   b. Replace the characters YOUR_QUEUE_NAME with a valid MQSeries queue name.

3. Using SQL*Plus, connect to your Oracle integrating server as a valid user.

4. Run test.sql, a script that sends and retrieves a message from an MQSeries queue. A successful completion displays the following output:

   SQL> @test.sql
   message put on queue = 00102030405060708090
   message read back = 00102030405060708090

   PL/SQL procedure successfully completed.
   An unsuccessful test displays the following output:

   SQL> @test.sql
   message put on queue = 00102030405060708090
   message read back = 00102030405060708090
   Error: Procedural Gateway for IBM MQSeries verification script failed.
   ORA-08500: The IBM MQSeries MQI call "MQOPEN" fails with reason code 2085
This chapter provides an overview of the Heterogeneous Message Propagation option. Heterogenous Services is a component of the Oracle9i server. You can also use the Oracle Messaging Gateway, which is included in the RDBMS bundle.

The following topics are included:

- **Heterogeneous Message Propagation Option** on page 9-2
- **Oracle Advanced Queuing (AQ)** on page 9-2
- **Operational and Administrative Interfaces** on page 9-2
- **Heterogeneous Outbound Propagation** on page 9-3
- **Heterogeneous Inbound Propagation** on page 9-5
Heterogeneous Message Propagation Option

Heterogeneous Message Propagation combines Oracle Advanced Queuing (AQ) with a gateway for message queuing systems in order to propagate messages between queues owned by Oracle Advanced Queuing (AQ) and queues owned by non-Oracle message queuing systems.

Oracle Advanced Queuing (AQ)

Oracle Advanced Queuing provides message queuing as an integral part of the Oracle® database server. With features that are tightly integrated with the database server, Oracle AQ enables application developers to use a messaging paradigm for integrating applications. The Oracle Open Gateways® for Message Queuing Systems provide access from the Oracle database server to non-Oracle message queuing systems such as IBM MQSeries. Using these gateways, Oracle applications can directly access (using PL/SQL) the messaging facilities of various message queuing systems while tightly integrating the non-Oracle messaging system with the Oracle database server transactional and typing model.

Heterogeneous Message Propagation will allow applications that are enabled with Oracle AQ to integrate with non-Oracle message queuing systems in an asynchronous, reliable, and efficient manner. Both sides will be able to deliver or consume the messages from their native message queues while the propagation will take care of distributing the messages between the two different message queuing systems.

Operational and Administrative Interfaces

Oracle Advanced Queuing (AQ) propagation is controlled through a set of well-defined operational and administrative interfaces. Heterogeneous Message Propagation will be integrated with these interfaces and will provide users with a unified Application Programming Interface (API) for both Oracle AQ and heterogeneous propagation.

Outbound Interfaces

On the Oracle integrating server side, Oracle Advanced Queuing (AQ) and heterogeneous outbound propagation share the same operational API. To the application programmer, using the operational API of Oracle AQ for enqueuing messages should be no different than using a non-Oracle queue. Enqueued
messages may be propagated to the queue of an Oracle (AQ) or non-Oracle message queuing system.

Similarly, the user should experience no difference in administering heterogeneous outbound message propagation in a non-Oracle message queuing system compared with doing so in an Oracle Advanced Queuing (AQ) environment.

Inbound Interfaces

On the Oracle integrating server side, both Oracle AQ and heterogeneous inbound propagation share the same operational API. To the application programmer, using the operational API of Oracle AQ for dequeueing messages should be no different whether the messages were propagated from an Oracle Advanced Queuing queue or a non-Oracle queue.

Heterogeneous Outbound Propagation

This section will define a model for heterogeneous outbound propagation and will describe how the various concepts that are described above are integrated with Oracle Advanced Queuing (AQ) or will describe where those concepts differ from an Oracle AQ user’s perspective.

Outbound Propagation in General

Heterogeneous outbound propagation of messages is controlled by the Oracle AQ user at three levels:

1. at the message level, by specifying (upon enqueuing the message) the recipient(s) to whom the message should be propagated.
2. at the queue level, by specifying one or more subscribers to whom the entered messages should be propagated.
3. at the administrative level, by scheduling propagation for an Oracle AQ source queue and a non-Oracle AQ destination.

A heterogeneous outbound propagation process analyses a scheduling table at regular intervals. This heterogeneous outbound propagation process determines which Oracle AQ queues (that are scheduled for outbound propagation) need to be serviced. The heterogeneous outbound propagation process also determines which of the messages must be propagated to a non-Oracle destination.
The propagation process removes any messages from the Oracle AQ source queue as specified in the schedule. The messages are retrieved using the regular Oracle AQ dequeue() interface call, and the messages are forwarded to the non-Oracle destination queue using a gateway for message queuing.

After being propagated, the messages in the destination queue are retrievable by non-Oracle applications using the native API of the non-Oracle message queuing system. The actual message propagation from an Oracle Advanced Queuing (AQ) source queue is transparent to these applications.

Refer to Figure 9–1, which shows all the components involved with heterogeneous outbound propagation. Each of the listed components will be described in more detail in the next sections.

**Figure 9–1  Heterogeneous Outbound Message Propagation**
Heterogeneous Inbound Propagation

Although heterogeneous inbound propagation has no Oracle AQ counterpart, the objective is to provide a model that (for Oracle AQ applications and administrators) looks similar to homogeneous propagation.

This section will define a model for heterogeneous inbound propagation and will describe how it integrates with Oracle AQ or where it differs from an Oracle AQ application’s perspective.

**Homogeneous Propagation versus Heterogeneous Propagation**

In homogeneous Oracle Advanced Queuing (AQ) propagation, inbound propagation does not exist. During homogeneous propagation, an Oracle integrating server only "pushes" messages to destinations at other Oracle servers. In this sense, an Oracle integrating server behaves in a manner that is very similar to heterogeneous outbound propagation. However, in homogeneous propagation, an Oracle server never "pulls" messages from source queues at other Oracle servers.

**Heterogeneous Propagation**

Heterogeneous outbound propagation is similar to homogeneous propagation in the sense that messages are "pushed" to destinations at other Oracle servers. With heterogeneous inbound propagation, however, messages are “pulled” from non-Oracle source queues when they are scheduled to be propagated to an Oracle AQ destination queue. The inbound propagation process on an Oracle integrating server acts as a propagation proxy for the non-Oracle message queuing system.
Inbound Propagation in General

Heterogeneous inbound propagation of messages is controlled from the Oracle side by the Oracle AQ administrator using an administrative package. The package allows the user to schedule and to unschedule inbound propagation for a non-Oracle source queue and an Oracle AQ destination. The scheduling information is maintained in an internal scheduling table.

A heterogeneous inbound propagation process analyses the scheduling table at regular intervals and determines which non-Oracle AQ queues that are scheduled for inbound propagation need to be serviced and need to have their message propagated to an Oracle AQ destination.

The propagation process removes messages from the pickup queue as specified in the schedule. The messages are retrieved (using a gateway for message queuing systems) and are forwarded to the target Oracle AQ queue on the local Oracle integrating server (using the regular Oracle AQ operational enqueue() interface call). Unlike heterogeneous outbound message propagation, with inbound propagation there is no "fan-out" to multiple Oracle AQ consumer queues. Each message from a non-Oracle source queue is sent to one, and only one, Oracle AQ destination queue.

After being propagated, the messages in the destination queue are retrievable by Oracle applications using the regular Oracle AQ dequeue() interface call. The actual message propagation from a non-Oracle source queue is transparent to these applications.

Refer to Figure 9–2, which shows all the components involved with heterogeneous inbound propagation. Each of the listed components is described in more detail in the next chapters.
Figure 9–2  Heterogeneous Inbound Message Propagation
Heterogeneous Inbound Propagation
10

Installing Heterogeneous Message Propagation

This chapter explains how to install and remove (deinstall) the Heterogeneous Message Propagation option. Heterogenous Services is a component of the Oracle9i server. You can also use the Oracle Messaging Gateway, which is included in the RDBMS bundle.

The following topics are included:

- Oracle Software Requirement on page 10-2
- Installing the Heterogeneous Message Propagation Option on page 10-2
- Verifying the Heterogeneous Message Propagation Option on page 10-4
- Verifying the Heterogeneous Message Propagation Option on page 10-4
Oracle Software Requirement

The following Oracle software is required to install the Heterogeneous Message Propagation option.

1. Make sure that your Oracle server is release 8.1.7 or higher.
2. Make sure that your Oracle Visual Workbench Repository for IBM MQSeries is installed. You need to install your Heterogeneous Message Propagation option on the Oracle integrating server where your Oracle Visual Workbench Repository for IBM MQSeries is installed.
3. If you have a release 4.0.1.1.0 (or a prior release) Visual Workbench Repository installed on your Oracle integrating server, then you need to back it up and upgrade it to release 8.1.7 or later.

Installing the Heterogeneous Message Propagation Option

Heterogeneous Message Propagation for IBM MQSeries comes as an option in the Oracle Procedural Gateway® for IBM MQSeries product. This feature can be enabled or disabled by running a PL/SQL script under SQL*Plus.

PL/SQL Installation Scripts

The heterogeneous message propagation option installation scripts are installed with the gateway software. The following scripts are available in the $ORACLE_HOME/pg4mqseries/admin directory.

<table>
<thead>
<tr>
<th>PL/SQL Scripts</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cathqprop.plb</td>
<td>install the Heterogeneous Message Propagation feature into the Oracle integrating server</td>
</tr>
<tr>
<td>catnohqprop.plb</td>
<td>remove the Heterogeneous Message Propagation feature from the Oracle integrating server</td>
</tr>
</tbody>
</table>

Installation

From SQL*Plus, as user SYS, run the cathqprop.plb script in the gateway admin directory, $ORACLE_HOME/pg4mqseries/admin.

SQL> @cathqprop.plb

Privileges and Access Control

Access to Heterogeneous Message Propagation operations is granted to users through roles. These roles provide execution privileges on the Heterogeneous Message Propagation procedures.

HQ_ADMINISTRATOR_ROLE:
This role grants execute privileges to procedures in the DBMS_HQADM and DBMS_HQ packages. These include all the administrative interfaces for Heterogeneous Message Propagation and operational interfaces for AQ. The user SYS must grant this role to the Heterogeneous Message Propagation administrator.

Calling DBMS_HQADM from a PL/SQL Function or Procedure:
If you wish to call DBMS_HQADM from a PL/SQL function or procedure, then you need to be explicitly granted the EXECUTE privilege by user SYS. You cannot inherit this right from the HQ_ADMINISTRATOR_ROLE.

GRANT EXECUTE ON DBMS_HQADM TO <user>;

AQ_ADMINISTRATOR_ROLE:
This role contains all the necessary GRANTS to allow the grantee EXECUTE privilege on DBMS_AQ and DBMS_AQADM. This role should be granted (by user SYS) to users who will need to administer Oracle AQ queues.

GRANT AQ_ADMINISTRATOR_ROLE TO <user>;
BEGIN DBMS_AQADM.GRANT_TYPE_ACCESS(’<user’);
END;
/
Verifying the Heterogeneous Message Propagation Option

1. Change directory to the gateway sample directory by entering:
   `$ cd $ORACLE_HOME/pg4mqseries/sample`

2. From SQL*Plus, as user SYS, run the grant.sql script and exit.
   ```
   SQL> @grant
   SQL> exit
   ```

3. Use an editor, modify aqmq.sql, outboundon.sql, outboundoff.sql, inboundon.sql, and inboundoff.sql scripts to:
   a. Specify the database link name that you created for the gateway. To do this, replace the characters `@pg4mq` with `@dblink`, where `dblink` is the name you chose when the database link was created.
   b. Replace the characters `YOUR_QUEUE_NAME` with a valid MQSeries queue name.
   c. Replace the characters `USERID` with `SCOTT`, which you use to run SQL*Plus.

4. Use SQL*Plus, connect to your Oracle integrating server as user SCOTT.

5. From SQL*Plus, run the `createhq.sql` script. This script creates sample AQ queue tables and sample queues for testing purposes.
   ```
   SQL> @createhq
   ```

6. From SQL*Plus, run the `aqmq.sql` script. This script puts messages into the sample queue "HMPDEMOQ1".
   ```
   SQL> @aqmq
   ```

7. From SQL*Plus, run the `outboundon.sql` script. This script turns on the Heterogeneous Message Outbound Propagation scheduler.
   ```
   SQL> @outboundon
   ```

To verify that the messages in sample queue "HMPDEMOQ1" are propagated to the target MQSeries queue, you may either check the sample queue table "HMPDEMOQTAB" or go to the MQSeries queue manager to check the queue depth. For more details, refer to your Oracle Advanced Queuing manual and IBM MQSeries manual.
8. From SQL*Plus, run the `outboundoff.sql` script. This script turns off the Heterogeneous Message Outbound Propagation scheduler.
   ```sql
   SQL> @outboundoff
   ```

9. From SQL*Plus, run the `inboundon.sql` script. This script turns on the Heterogeneous Message Inbound Propagation scheduler.
   ```sql
   SQL> @inboundon
   ``
   To verify that the messages in your MQSeries queue are propagated to the sample AQ queue "HMPDEMOQ2", you may either check the sample queue table "HMPDEMOQTAB" or go to the MQSeries queue manager to check the queue depth. For great details, refer to your Oracle Advanced Queuing manual and IBM MQSeries manual.

10. From SQL*Plus, run the `inboundoff.sql` script. This script turns off the Heterogeneous Message Inbound Propagation scheduler.
    ```sql
    SQL> @inboundoff
    ```

11. From SQL*Plus, run the `drophq.sql` script. This script removes all of the sample queue tables and sample queues.
    ```sql
    SQL> @drophq
    ```

---

**Deinstalling the Heterogeneous Message Propagation Option**

From SQL*Plus, as user SYS, run the `catnohqprop.plb` script in the gateway admin directory, `$ORACLE_HOME/pg4mqseries/admin`.

```sql
SQL> @catnohqprop.plb
```
Deinstalling the Heterogeneous Message Propagation Option
Configuring Heterogeneous Outbound Message Propagation

After installing the Heterogeneous Message Propagation option, follow the instructions in this chapter to configure and use the Heterogeneous Outbound Message Propagation option. Heterogenous Services is a component of the Oracle9i server. You can also use the Oracle Messaging Gateway, which is included in the RDBMS bundle.

Configuration tasks and other administrative topics include:

- Overview of Outbound Message Propagation on page 11-2
- Translation of Oracle AQ/IBM MQSeries Interfaces on page 11-2
- Configuring Outbound Message Propagation on page 11-8
- Operational Interface for Individual Messages on page 11-9
- Operational Interface for Queues on page 11-14
- Administrative Interfaces on page 11-17
- Security on page 11-23
- Exception Handling on page 11-24
- Schedule Status on page 11-25
Overview of Outbound Message Propagation

Heterogeneous Outbound Message Propagation to MQSeries is accomplished in three steps:

1. dequeuing the messages from the Oracle AQ source queue
2. translating the message service properties
3. enqueuing the messages in the MQSeries destination queue.

After the propagation schedule has started for a source/destination tuple, the propagation process will use the Oracle AQ operational Application Programming Interface (API) to dequeue those messages that have specified a list of recipients or subscribers (if the queue has defined a list of subscribers), or the propagation process will use the Oracle AQ operational API to dequeue all messages from the source queue.

For every message dequeued, the Oracle AQ service properties that are associated with the message are translated (when applicable) to MQSeries properties. Examples of translated Oracle AQ service properties are queue table, queue, message, and enqueue properties.

After the message service properties are translated, the message is enqueued at the MQSeries destination queue. The Oracle9i Procedural Gateway for IBM MQSeries provides PL/SQL™ access to MQSeries.

The outbound message propagation process will call the necessary gateway’s procedure calls to perform the following tasks:

- to establish a connection with the MQSeries Queue Manager
- to open the MQSeries destination queue
- to set the necessary MQSeries properties (as defined in the second step)
- to enqueue the message
- to close the queue again

Translation of Oracle AQ/IBM MQSeries Interfaces

IBM MQSeries provides a set of services that is comparable to the set of services that is provided by Oracle AQ. For outbound message propagation, Heterogeneous Message Propagation will provide dynamic, best-effort translations of Oracle AQ interfaces and message service properties into MQSeries properties. The next section lists the Oracle AQ interfaces, message service properties, and options that
applications can specify upon enqueuing a message. The next section then describes how these items are translated to MQI (the native API for MQSeries applications).

Oracle AQ messages are entered into Oracle AQ queues using the `enqueue()` procedure (refer to the Oracle AQ documentation for more details on the `dbms_aq` package).

### Translation for Outbound Propagation for MQSeries

#### DBMS_AQ.ENQUEUE

Oracle AQ queue properties and queue table properties are thought to have no impact on heterogeneous outbound message propagation. However, some queue properties and queue table properties are important for outbound message propagation and they are therefore listed here in Table 11–1 with the enqueue options and message properties.

<table>
<thead>
<tr>
<th>Oracle AQ Argument</th>
<th>Translation for IBM MQSeries (outbound propagation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>N.A. This is the Oracle AQ source queue.</td>
</tr>
<tr>
<td>enqueue_options</td>
<td>Refer to table on <code>enqueue_options_t</code> next.</td>
</tr>
<tr>
<td>message_properties</td>
<td>Refer to table on <code>message_properties_t</code> next.</td>
</tr>
<tr>
<td>payload</td>
<td>The message data is propagated to MQSeries. Outbound message propagation supports only RAW payloads. <strong>Note:</strong> The maximum length of payload for MQSeries V2, V5, and Oracle AQ is respectively 4M, 100M, and 32K. Therefore, payloads for outbound message propagation are restricted to 32K.</td>
</tr>
<tr>
<td>msgid</td>
<td>No translation</td>
</tr>
</tbody>
</table>

**Note:** Although IBM MQSeries allows applications to set `msgid`, Oracle AQ does not support this feature. Therefore, propagation of `msgid` values cannot be supported.
**DBMS_AQADM.CREATE_QUEUE_TABLE**

Oracle AQ Queue Table properties are specified upon using the `create_queue_table()` procedure (refer to the Oracle AQ documentation for more details on the `dbms_aqadm` package). Outbound message propagation for MQSeries will use the following translations:

<table>
<thead>
<tr>
<th>Oracle AQ Property</th>
<th>Translation for IBM MQSeries (outbound propagation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_table</td>
<td>N.A.</td>
</tr>
<tr>
<td>queue_payload_type</td>
<td>N.A.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Propagation supports only Oracle AQ source queues of type RAW.</td>
</tr>
<tr>
<td>storage_clause</td>
<td>N.A.</td>
</tr>
<tr>
<td>sort_list</td>
<td>No translation</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Propagation retains order of messages when moving them from source to destination queue.</td>
</tr>
<tr>
<td>multiple_consumers</td>
<td>No translation</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Although MQSeries V5 supports multiple consumers per message, outbound message propagation will not bundle messages when distributing them to MQSeries.</td>
</tr>
<tr>
<td>message_grouping</td>
<td>No translation</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Outbound message propagation will preserve grouping of messages. In other words, all messages belonging to a group are propagated to an MQSeries destination queue and are not visible to MQSeries consumers until the last message of the group has been propagated successfully.</td>
</tr>
<tr>
<td>comment</td>
<td>N.A.</td>
</tr>
<tr>
<td>auto_commit</td>
<td>N.A.</td>
</tr>
</tbody>
</table>
Oracle AQ Queue properties are specified upon using the `create_queue()` procedure (refer to the Oracle AQ documentation for more details on the `dbms_aqadm` package). Outbound message propagation for MQSeries will use the following translations:

### Table 11–3  DBMS_AQADM.CREATE_QUEUE

<table>
<thead>
<tr>
<th>Oracle AQ Property</th>
<th>Translation for IBM MQSeries (outbound propagation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>N.A.</td>
</tr>
<tr>
<td>queue_table</td>
<td>N.A.</td>
</tr>
<tr>
<td>queue_type</td>
<td>N.A.</td>
</tr>
<tr>
<td>max_retries</td>
<td>N.A.</td>
</tr>
<tr>
<td>retry_delay</td>
<td>N.A.</td>
</tr>
<tr>
<td>retention_time</td>
<td>N.A.</td>
</tr>
<tr>
<td>dependency_tracking</td>
<td>N.A.</td>
</tr>
<tr>
<td>comment</td>
<td>N.A.</td>
</tr>
<tr>
<td>auto_commit</td>
<td>N.A.</td>
</tr>
</tbody>
</table>
DBMS_AQ.ENQUEUE_OPTIONS_T
Upon entering a message in the Oracle AQ source queue using the enqueue() procedure (refer to the Oracle AQ documentation on dbms_aq package), the Oracle AQ application can specify enqueue options. Outbound message propagation for MQSeries will use the following translations:

<table>
<thead>
<tr>
<th>Oracle AQ Option</th>
<th>Translation for IBM MQSeries (outbound propagation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>visibility</td>
<td>N.A. It does not apply to a propagation context.</td>
</tr>
<tr>
<td>relativemsgid</td>
<td>No translation. Refer to sequence_deviation.</td>
</tr>
<tr>
<td>sequence_deviation</td>
<td>No translation. There is no equivalent service in MQSeries.</td>
</tr>
</tbody>
</table>

DBMS_AQ.MESSAGE_PROPERTIES_T
Upon entering a message in the Oracle AQ source queue using the enqueue() procedure (refer to the Oracle AQ documentation on dbms_aq package), the Oracle AQ application can specify message properties. Outbound message propagation for MQSeries will use the following translations:

<table>
<thead>
<tr>
<th>Oracle AQ Property</th>
<th>Translation for IBM MQSeries (outbound propagation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>priority</td>
<td>Translated to MQMD.Priority.</td>
</tr>
<tr>
<td></td>
<td>Values 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 are translated respectively to 9, 8, 7, 6, 5, 4, 3, 2, 1, 0,</td>
</tr>
<tr>
<td></td>
<td>Values &lt; 0 are translated as 9,</td>
</tr>
<tr>
<td></td>
<td>Values &gt;= 10 are translated to 0.</td>
</tr>
<tr>
<td>delay</td>
<td>No translation. Equivalent support not available in MQSeries</td>
</tr>
<tr>
<td></td>
<td>If Oracle AQ would support delay in the source queue when protocol is NOT equal to 0, then heterogeneous outbound propagation could provide delay support.</td>
</tr>
<tr>
<td>expiration</td>
<td>Translated to MQMD.Expiry</td>
</tr>
<tr>
<td></td>
<td>Value NEVER translated to MQEI_UNLIMITED.</td>
</tr>
<tr>
<td></td>
<td>Values in seconds -&gt; tenth of seconds.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: The expiration value in seconds is adjusted by subtracting the already expired time since the message was enqueued at enqueue_time in the Oracle AQ source queue.</td>
</tr>
</tbody>
</table>
correlation

Translated to MQMD.CorrelId. Values are handed over as binary data and values longer than 24 characters are truncated.

attempts

No translation.

recipient_list

No translation. Although MQSeries V5 supports multiple consumers per message, outbound message propagation will not bundle messages when distributing them to MQSeries.

Note: The outbound message propagation process for MQSeries internally uses the recipient_list information specified by the Oracle AQ application but does not need to forward it to MQSeries.

exception_queue

No translation

Note: The outbound message propagation process for MQSeries internally uses the exception_queue information specified by the AQ application but does not need to forward it to MQSeries.

enqueue_time

No translation

Note: Upon propagation the MQSeries Queue Manager will populate the MQMD.PutTime and MQMD.PutDate fields automatically.

state

No translation

Note: The AQ queue monitor internally maintains the state information, but the outbound message propagation process for MQSeries does not forward it to MQSeries.

<table>
<thead>
<tr>
<th>Oracle AQ Property</th>
<th>Translation for IBM MQSeries (outbound propagation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>correlation</td>
<td>Translated to MQMD.CorrelId. Values are handed over as binary data and values longer than 24 characters are truncated.</td>
</tr>
<tr>
<td>attempts</td>
<td>No translation.</td>
</tr>
<tr>
<td>recipient_list</td>
<td>No translation. Although MQSeries V5 supports multiple consumers per message, outbound message propagation will not bundle messages when distributing them to MQSeries. Note: The outbound message propagation process for MQSeries internally uses the recipient_list information specified by the Oracle AQ application but does not need to forward it to MQSeries.</td>
</tr>
<tr>
<td>exception_queue</td>
<td>No translation</td>
</tr>
<tr>
<td>enqueue_time</td>
<td>No translation. Note: Upon propagation the MQSeries Queue Manager will populate the MQMD.PutTime and MQMD.PutDate fields automatically.</td>
</tr>
<tr>
<td>state</td>
<td>No translation. Note: The AQ queue monitor internally maintains the state information, but the outbound message propagation process for MQSeries does not forward it to MQSeries.</td>
</tr>
</tbody>
</table>
Configuring Outbound Message Propagation

Configuration is needed on the Oracle® server, the Oracle Procedural Gateway®, and the IBM MQSeries to facilitate message propagation to an IBM MQSeries queue.

Oracle integrating server: A database link must be set up to point to the MQSeries gateway.

Gateway: The gateway must be set up to be the proxy of one IBM MQSeries queue manager (called ‘gateway queue manager’), and must be run under strict security mode.

MQSeries: If the target MQSeries queue is under the gateway queue manager, then no configuration is required. Otherwise, a remote queue must be created in the gateway queue manager to point to the target queue.

Figure 11–1 Heterogeneous Outbound Message Propagation
To propagate a message from Oracle Advanced Queue ‘aq12′ under Oracle server ‘ora1′ over to IBM MQSeries queue ‘ny11′ under queue manager ‘mgr.ny.1′, the following configuration is required:

Oracle server: database link pg4mq in ora1 to point to the MQSeries gateway

Gateway: configure the gateway with queue manager mgr.gtw and setup strict security mode.

MQSeries: create remote queue xmit1 under queue manager mgr.gtw for the remote queue ny11 under queue manager mgr.ny.1, and create transmission queue and channels for the remote queue manager mgr.ny.1 if they do not already exist.

**Operational Interface for Individual Messages**

Oracle AQ allows propagation to be defined for queues and individual messages. This section describes how heterogeneous outbound message propagation will work for individual messages; the section hereafter will outline how outbound message propagation is configured for a queue.

For heterogeneous outbound message propagation of individual messages, we will re-use the Oracle AQ operational interface. Oracle AQ application will be able to specify non-Oracle consumers for a message using the standard recipient list mechanism. Upon enqueuing a message, the message_properties argument of the enqueue() procedure allows the Oracle AQ application to specify a list of recipients. For heterogeneous outbound message propagation, the fields of the structure used to specify the recipient will contain the communication information to access the non-Oracle AQ destination queue.
Refer to the next figure with definitions of the Oracle AQ enqueue() procedure and Oracle AQ types involved in this model:

```sql
dbms_aq.enqueue (
    queue_name      IN     varchar2,
    enqueue_options    IN     enqueue_options_t,
    message_properties  IN     message_properties_t,
    payload        IN     raw,
    msgid         OUT    raw
)

TYPE dbms_aq.enqueue_options_t IS RECORD (
    visibility       binary_integer,
    relative_msgid     raw(16),
    sequence_deviation   binary_integer)

TYPE dbms_aq.message_properties_t IS RECORD (
    priority       binary_integer,
    delay         binary_integer,
    expiration      binary_integer,
    correlation      varchar2(128),
    attempts       binary_integer,
    recipient_list    aq$_recipient_list,
    exception_queue    varchar2(51),
    enqueue_time     date,
    state         binary_integer)

TYPE dbms_aq.aq$_recipient_list_t IS TABLE OF
    sys.aq$_agent INDEX BY binary_integer

TYPE dbms_aq.sys.aq$_agent IS OBJECT (
    name          varchar2(30),
    address         varchar2(1024),
    protocol        number)
```
Operational Interface for Individual Messages

Refer to the Oracle AQ documentation for detailed information on how to use the `enqueue()` procedure and its arguments. For heterogeneous outbound message propagation, the usage is generally compatible with Oracle AQ propagation. Some arguments have different behavior however, due to the limited capabilities of the non-Oracle message queuing system:

- The `payload` argument contains the application data. The `payload` type should be in concert with the type of the queue specified in the `queue_name` argument and can either be RAW or ADT. Heterogeneous outbound message propagation supports only RAW queues. Therefore, only RAW `payloads` are supported.

- The `msgid` argument is a message identifier assigned to a message by the Oracle AQ system. The `msgid` argument is globally unique within the Oracle domain. However, heterogeneous outbound message propagation cannot pass the `msgid` along with the message to the non-Oracle message queuing system. Although IBM MQSeries allows applications to set `msgid`, Oracle AQ does not support the IBM MQSeries method of setting `msgid`. For this reason, propagation of `msgid` values cannot be supported (because the `msgid` returned by the `enqueue()` procedure will not match the `msgid` of the propagated messages as assigned by the non-Oracle message queuing system).

---

**Note:** Oracle AQ applications can use the `correlation` field for message identification, and `correlation` values will be preserved when propagated to non-Oracle message queuing systems.

---

- The `sys.aq$_agent` structure that is used to specify the recipient list of the `message_properties` argument plays a key role in heterogeneous outbound message propagation. Its fields are used to specify the non-Oracle destination queue of the recipients.
### sys.aq$_agent usage

**Table 11–6  sys.aq$_agent.usage**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
</table>
| **name** | is a unique feature in Oracle AQ to identify the intended consumer of the message. The information is not forwarded to the non-Oracle message queuing system.  
**Note:** Administrators use the name to track the propagation status of individual messages. |
| **address** | is used to specify the non-Oracle destination queue and the communication path to access the non-Oracle queuing system. Using Oracle gateways, the same database link mechanism that is used to specify a remote Oracle AQ server in homogeneous Oracle AQ propagation is also used to specify a gateway for a non-Oracle message queuing system.  
**Syntax:**  
<non_oracle_queue>@<database_link>  
The syntax and maximum length of the non-Oracle queue name depend on the non-Oracle message queuing system and could differ from Oracle AQ. |
| **protocol** | specifies the type of outbound message propagation protocol.  
**Valid value:**  
DBMS_HQADM.MQSERIES_OUTBOUND for IBM MQSeries. |

The following example demonstrates how the Oracle AQ operational interface can be used for both Oracle AQ and heterogeneous outbound message propagation. The code excerpt shows part of an Oracle AQ application that specifies both Oracle AQ and non-Oracle AQ consumers as recipients for a message entered into an Oracle AQ queue:
Example:

```sql
DECLARE
    enqueue_options DBMS_AQ.ENQUEUE_OPTIONS_T;
    message_properties DBMS_AQ.MESSAGE_PROPERTIES_T;
    recipients DBMS_AQ.AQ$_RECIPIENT_LIST_T;
    message_handle RAW(16);
    message RAW(255) := 'something';
BEGIN
    /* message receiver 1: queue aq1 at remote Oracle Database */
    recipients(1) := SYS.AQ$_AGENT('Joe', 'aq1@remoteAq', 0);
    /* message receive 2: queue mq1 at MQSeries */
    recipients(2) := SYS.AQ$_AGENT('John',
        'mq1@mqseries',
        DBMS_HQADM.MQSERIES_OUTBOUND);
    message_properties.recipient_list := recipients;
    /* enqueue the message */
    DBMS_AQ.ENQUEUE(queue_name => 'msg_queue',
        enqueue_options => queue_options,
        message_properties => message_properties,
        payload => message,
       msgid => message_handle);
    COMMIT;
END;
/```
The example propagates the same message to two different recipients:

**Recipient 1: Joe at aq1@remoteAq**

As the protocol number 0 indicates, this is Oracle AQ propagation with the destination queue ‘aq1’ residing at a remote Oracle database referred to by database link ‘remoteAq’. The intended consumer of the message is consumer ‘Joe’.

**Recipient 2: mq1@mqseries**

Protocol `DBMS_HQADM.MQSERIES_OUTBOUND` indicates outbound message propagation from Oracle AQ to MQSeries. The destination is MQSeries queue ‘mq1’, which can be reached using the MQSeries gateway. The gateway is referred to by the database link ‘mqseries’. Because MQSeries does not have the concept of message consumer, ‘John’ will be ignored during propagation.

**Operational Interface for Queues**

Oracle AQ allows propagation to be defined for queues and individual messages. The previous section described how heterogeneous outbound message propagation works for individual messages. This section will define how to configure heterogeneous outbound message propagation for an Oracle AQ queue.

The Oracle AQ administrator’s application will be able to specify non-Oracle consumers for an Oracle AQ queue using the standard subscriber mechanism. After having created a queue, the Oracle AQ administrator can use the `add_subscriber()` procedure to add a non-Oracle AQ subscriber. Subscribers are defined using a `sys.aq$_agent` structure.

**Note:** Propagation that is specified for individual messages using a `recipient_list` overrides heterogeneous subscribers that are specified for queues.

The next figure shows the definitions of the Oracle AQ `add_subscriber()` procedure and Oracle AQ structures that are used for specifying heterogeneous outbound message propagation for Oracle AQ queues.
Oracle AQ Procedures and Structures

Oracle AQ procedures and structures used with outbound message propagation for Oracle AQ queues

DBMS_AQADM.ADD_SUBSCRIBER (
    queue_name IN   varchar2,
    subscriber     sys.aq$_agent)

TYPE sys.aq$_agent IS OBJECT (
    name       varchar2(30),
    address     archar2(1024),
    protocol     number)

Refer to the Oracle AQ documentation for detailed information on how to use the add_subscriber() procedure and its arguments. For heterogeneous outbound message propagation, the usage is generally compatible with Oracle AQ propagation. However, the sys.aq$_agent structure that is used to specify the subscriber plays a key role in heterogeneous outbound message propagation. Its fields are used to specify the non-Oracle destination queue of the subscriber.

Refer to Table 11-6, ”sys.aq$_agent.usage” for a description of the sys.aq$_agent field for heterogeneous outbound message propagation.

The following example demonstrates how Oracle AQ and heterogeneous propagation can be specified using the same Oracle AQ administrative interface. The code example shows part of an Oracle AQ application that adds both Oracle AQ and non-Oracle AQ subscribers for an Oracle AQ queue:
Example

DECLARE
    subscriber SYS.AQ$_AGENT;
BEGIN
    /* Add an AQ subscriber */
    subscriber := SYS.AQ$_AGENT('Joe',
        'aq1@remoteAq',
        0);

    DBMS_AQADM.ADD_SUBSCRIBER(queue_name => 'source_queue',
        subscriber => subscriber );

    /* Add a non-AQ subscriber */
    subscriber := SYS.AQ$_AGENT('John',
        'mq1@mqseries',
        DBMS_HQADM.MQSERIES_OUTBOUND);

    DBMS_AQADM.ADD_SUBSCRIBER(queue_name => 'source_queue',
        subscriber => subscriber );

END;
/

The above example specifies an Oracle AQ and a non-Oracle AQ subscriber for 'source_queue' such that any message entered into the source queue will automatically be propagated to the following subscribers:

**AQ Subscriber: Joe at aq1@remoteAq**

This will initiate Oracle AQ propagation, with the destination queue 'aq1' residing at a remote Oracle® database referred to by database link 'remoteAq', and with the intended consumer of the message set to 'Joe'.

**Non-AQ subscriber: mq1@pg4mqseries**

Protocol DBMS_HQADM.MQSERIES_OUTBOUND indicates outbound message propagation from Oracle AQ to MQSeries (refer to Table 11-6, "sys.aq$_agent.usage" for valid protocol numbers). The destination is an MQSeries queue 'mq1' that can be reached by the MQSeries gateway. The gateway is referred to by the database link 'mqseries'. Because MQSeries does not have the concept of message consumer, 'John' will be ignored during propagation.
Administrative Interfaces

For propagation in a homogeneous Oracle AQ environment, the administrator uses the Oracle AQ package `DBMS_AQADM` with procedures to schedule a propagation job. Specifying an Oracle AQ propagation job requires specifying three items:

1. an Oracle AQ source queue name
2. the destination
3. the scheduling times

The Oracle AQ administration package in Oracle 8.0.4 and Oracle 8.1 cannot yet accommodate heterogeneous protocols. For this reason, a separate scheduling package is used. This scheduling package has procedures for the scheduling and unscheduling of heterogeneous outbound message propagation. The procedures that are in this package resemble the Oracle AQ scheduling procedures.

The procedures maintain schedules in dictionary tables. These procedures are read by the propagation process to determine when a queue, and which queue, is due for another round of message propagation. The schedule information is available through a view named `DBA_OUTBOUND_SCHEDULES` (refer to Appendix D, "Heterogeneous Message Propagation DBA Views"). The package that contains the scheduling procedures (both inbound and outbound) is named `DBMS_HQADM` and contains two procedures for outbound message propagation:

- `SCHEDULE_PROPAGATION`
- `UNSCHEDULE_PROPAGATION`

The two outbound scheduling procedures that are in the package are each composed of a name, purpose, syntax, usage, and error messages. These two outbound scheduling procedures are described next.

Procedures for Outbound Message Propagation

Name: `SCHEDULE_PROPAGATION`

Purpose

Upon calling, the procedure schedules propagation of messages from an Oracle AQ source queue to a non-Oracle destination queue. If a message has multiple consumers at the same non-Oracle destination in the same or different queues, then the message will be propagated to all of them at the same time.
Syntax

```sql
dbms_hqadm.schedule_propagation(
    queue_name       in    varchar2,
    protocol        in    number,
    destination      in    varchar2,
    start_time       in    date default SYSDATE,
    duration        in    number default NULL,
    next_time       in    varchar2 default NULL,
    latency        in    number default 60 )
```

Usage:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| queue_name  | specifies the name of the source Oracle AQ queue whose messages are to be propagated. The name includes the schema name. If the schema name is not specified, then `queue_name` defaults to the schema name of the administrative user.  
  **Note:** Heterogeneous outbound message propagation supports queues of type RAW, and only type RAW. |
| protocol    | specifies the type of outbound message propagation protocol.  
  Valid values: `DBMS_HQADM.MQSERIES_OUTBOUND` |
| destination | defines the foreign queue and database link to the gateway for accessing the non-Oracle message queuing system. Messages in the source queue for recipients at this destination will be propagated. If the name is not fully qualified, then the default domain name is used.  
  **Note:** Although NULL is accepted as a valid destination value for Oracle AQ propagation, NULL is not accepted as a valid destination value for heterogeneous outbound message propagation. |
| start_time  | specifies the initial start time for the propagation window for messages from the Oracle AQ source queue to the non-Oracle destination queue. |
| duration    | specifies the duration of the propagation window in seconds. A NULL value means that the propagation window is forever, or until the propagation is unscheduled. |
| next_time   | specifies the initial start time for the propagation window for messages from the Oracle AQ source queue to the non-Oracle destination queue. |
Error Messages

ORA-20001 An invalid value was specified for QUEUE_ADDRESS. The value should be of the form queue@dblink[.domain]

*Cause:* The foreign queue specification for the source and/or destination parameter is either NULL or not in the right format.

*Action:* Check the parameter, and correct the mistake.

ORA-20002 Heterogeneous propagation for ADT queues is not yet supported.

*Cause:* The Oracle AQ queue that was specified for heterogeneous propagation is an ADT queue, which is not supported.

*Action:* Specify a RAW queue.

ORA-20003 This error occurred while creating a MIP for this schedule.

*Cause:* The propagation package has a problem creating the interface to the foreign queue system for the schedule.

*Action:* This is an internal error. Report this error to Oracle.

---

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>latency</td>
<td>is the maximum wait, in seconds, in the propagation window for a message to be propagated after it is enqueued. However, if the latency is (for example) 60 seconds, and if no messages are waiting to be propagated, then during the propagation window, messages from that Oracle AQ source queue for the non-Oracle destination queue will not be propagated for at least 60 more seconds. If the latency is 0, then the heterogeneous propagation process will be waiting for messages to be enqueued for the destination, and as soon as a message is enqueued, it will be propagated.</td>
</tr>
</tbody>
</table>
Name: UNSCHEDULE_PROPAGATION

Purpose
This procedure for outbound message propagation unschedules previously scheduled heterogeneous outbound message propagation from an Oracle AQ source queue to a non-Oracle destination. If no messages are left on the propagation-exception queue or the transmit queue then these queues will be deleted. If these queues still hold messages, then the procedure will fail unless the user specifies the force option as TRUE, causing the messages to be purged and the queues to be cleaned up. Refer to the section on 'Exception Handling' on page 11-24 for more information about the propagation-exception and transmit queues.

Syntax
```sql
dbms_hqadm.unschedule_propagation(
    queue_name   in    varchar2,
    protocol    in    number,
    destination   in    varchar2 default NULL,
    force      in    boolean default FALSE)
```

Usage

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>specifies the name of the source queue whose outbound message propagation schedule should be ended. The name includes the schema name. If the schema name is not specified, then queue_name defaults to the schema name of the administrative user.</td>
</tr>
<tr>
<td>protocol</td>
<td>specifies the type of outbound message propagation protocol. Valid values: DBMS_HQADM.MQSERIES_OUTBOUND</td>
</tr>
<tr>
<td>destination</td>
<td>defines the database link to the gateway for accessing the non-Oracle message queuing system. Messages in the source queue for recipients at this destination will be propagated. If the name is not fully qualified, then the default domain name is used. <strong>Note:</strong> Although NULL is accepted as a valid destination value for Oracle AQ propagation, for heterogeneous outbound message propagation, NULL is not accepted as a valid destination value.</td>
</tr>
</tbody>
</table>
Error Messages:

ORA-20001Invalid value for QUEUE_ADDRESS, should be of the form queue@dblink[.domain]

Cause: The foreign queue specification for the source/destination parameter is either NULL or is not in the right format.

Action: Check the parameter, and correct the mistake.

ORA-20004Propagation is still in progress.

Cause: Messages are left in the xmit queue or the exception queue.

Action: Use the ‘force’ option, or wait until the propagation is finished.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>force</td>
<td>specifies whether the procedure should succeed even if messages are left behind on the propagation-exception or transmit queues. Refer to the section on “Exception Handling” on page 11-24 for more information about the propagation-exception and transmit queues. Valid values: <strong>TRUE</strong>: Any message that is left on the exception or transmit queue will be purged, and the queues will be cleaned up. <strong>FALSE</strong>: The procedure will fail when messages are found on either the exception or transmit queue.</td>
</tr>
</tbody>
</table>
Example

Continuing on the earlier example of using the Oracle AQ operational interface for both Oracle AQ and heterogeneous outbound message propagation, the following code example demonstrates how Oracle AQ administrators would schedule propagation for both.

```
BEGIN
    /* schedule propagation for AQ */
    DBMS_AQADM.SCHEDULE_PROPAGATION(
        queue_name => 'aq1',
        destination => 'remoteAq');

    /* schedule outbound propagation for MQSeries */
    DBMS_HQADM.SCHEDULE_PROPAGATION(
        queue_name => 'mq1',
        protocol => DBMS_HQADM.MQSERIES_OUTBOUND,
        destination => 'QUEUE1@dblink');

END;
/
```
Heterogeneous Message Propagation does not impose new authorization requirements on either Oracle AQ or non-Oracle operational applications. For outbound message propagation, a sending Oracle AQ application enqueues the messages into the source queue but does not require authorization for the recipient and/or subscriber’s queue at the non-Oracle message queuing system. Non-Oracle consumers require authorization only for the non-Oracle AQ destination queue.

At the administration level, the user who schedules heterogeneous propagation for a source and/or destination tuple:

- requires authorization to access the Oracle AQ source queue.
- requires access to a database link for a gateway in order to access the destination queue at the non-Oracle message queuing system.

The user name and password specified for the database link require access privileges at the message queuing system of the destination queue. If the database link does not specify a user name and password, then it will implicitly use the ID of the current Oracle user.

Given these access privileges, the propagation process can dequeue the messages from the source queue and enqueue them through the gateway into the destination queue. This method of controlling access privileges ensures that the security issue for outbound message propagation remains transparent to both sender and consumer applications.
Exception Handling

If the outbound message propagation process cannot propagate messages from the Oracle AQ source queue to their destination, then the messages will be moved to either a propagation exception queue or to an inactive transmit queue, as explained below.

**Propagation Exception Queue**

For each schedule that is created by using the `dbms_hqadm.schedule_propagation()` procedure, an exception queue is created to stage messages that could not be successfully moved to the destination queue. Administrators can check the `exception_queue` field in the table `DBA_OUTBOUND_SCHEDULES` (refer to Appendix D, "Heterogeneous Message Propagation DBA Views") to obtain the name of this exception queue and to monitor it.

Upon unscheduling a propagation job by using the `dbms_hqadm.unschedule_propagation()` procedure, the exception queue is removed again (for full details, refer to the description of this procedure at "Name: UNSCHEDULE_PROPAGATION" on page 11-20).

**Inactive Transmit Queues**

Inactive transmit queues are created by the propagation process upon encountering messages in a source queue for which no propagation has been scheduled yet. Administrators can check the `transmit_queue` field in the table `DBA_OUTBOUND_SCHEDULES` (refer to Appendix D, "Heterogeneous Message Propagation DBA Views") to obtain the name of this queue (if any) and monitor it.

Upon unscheduling a propagation job using the `dbms_hqadm.unschedule_propagation()` procedure, the inactive transmit queue is removed again (for full details, also refer to the description of this procedure).

**Inactive transmit queue explanation:** The Oracle AQ API allows the propagation process to dequeue messages by protocol but not by destination or source. This means that messages with the same protocol but different destination or source are all dequeued at once. To avoid this, the propagation process must perform a 2-phase-commit when distributing messages to their respective destinations. The propagation process stages these messages into dedicated transmit queues. Each transmit queue is reserved for a particular schedule. If, however, no schedule was created for a message, and thus no transmit queue exists yet, then the propagation process creates an inactive transmit queue and moves the message in there.
Schedule Status

Status Information can be obtained both for an outbound message propagation schedule and for individual messages.

Status of an Outbound Message Propagation Schedule

The administrator can check the view, DBA_OUTBOUND_SCHEDULES, to determine if outbound message propagation has been scheduled for a particular combination of Oracle AQ source queue and non-Oracle destination queue. For each created schedule that the outbound message propagation process maintains in this view, the names of exception and transmit queues are included, as well as up to date status and statistics and other information.

For example, to find all schedules not running because of a failure, and to find the reason for the failure, the following query can be executed:

```sql
SELECT ERROR_TIME, ERROR, ERROR_MSG
FROM DBA_OUTBOUND_SCHEDULES
WHERE DISABLED = 'Y';
```
After installing the Heterogeneous Message Propagation option, follow the instructions in this chapter to configure and use the Heterogeneous Inbound Message Propagation option. Heterogenous Services is a component of the Oracle9i server. You can also use the Oracle Messaging Gateway, which is included in the RDBMS bundle.

Heterogeneous Inbound Message Propagation option configuration tasks and other administrative topics include:

- Overview of Inbound Message Propagation From MQSeries on page 12-2
- Translation of MQSeries/Oracle AQ Interfaces on page 12-2
- Configuring Inbound Message Propagation on page 12-6
- Operational Interface on page 12-8
- Administrative Interface on page 12-11
- Security on page 12-18
- Exception Handling on page 12-19
- Schedule Status on page 12-19
Overview of Inbound Message Propagation From MQSeries

Heterogeneous inbound propagation is accomplished in three steps:

1. by dequeuing the messages from the MQSeries source queue using the gateway
2. by translating the message service properties
3. by enqueuing the messages in the Oracle AQ destination queue.

After the propagation schedule has started for a source and/or destination tuple, the propagation process will make the necessary gateway procedure calls to establish a connection with the MQSeries Queue Manager, to open the source queue as described for the schedule, to dequeue the message, and to close the queue again.

For every message that is dequeued, the MQSeries service properties that are associated with the message are translated to Oracle AQ properties when applicable.

Translation of MQSeries/Oracle AQ Interfaces

IBM MQSeries provides a set of services that is comparable to Oracle AQ. For inbound message propagation, Heterogeneous Message Propagation will provide dynamic, best-effort translations of MQSeries interfaces and message service properties into Oracle AQ properties. This section lists the Oracle AQ interface and message service properties that Oracle AQ applications can retrieve upon dequeuing a message. The next section also describes how those interface and message service properties are translated from MQI (the native API for MQSeries applications).

Messages that are propagated from non-Oracle message queuing systems are available to Oracle AQ applications. These messages may be retrieved by dequeuing them from the Oracle AQ destination queues by using the `dequeue()` procedure (refer to the Oracle AQ documentation for more details on the `dbms_aq` package). Inbound propagation for MQSeries will use the following translations:

- `DBMS_AQ.DEQUEUE`
- `DBMS_AQ.DEQUEUE_OPTIONS_T`
- `DBMS_AQ.MESSAGE_PROPERTIES_T`
Translations for inbound propagation for MQSeries:

**DBMS_AQ.DEQUEUE:**

<table>
<thead>
<tr>
<th>Oracle AQ Argument</th>
<th>Translation for IBM MQSeries (inbound propagation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>N.A., this is the Oracle AQ destination queue.</td>
</tr>
<tr>
<td>dequeue_options</td>
<td>Refer to table on dequeue_options_t next.</td>
</tr>
<tr>
<td>message_properties</td>
<td>Refer to table on message_properties_t next.</td>
</tr>
<tr>
<td>payload</td>
<td>The message data is propagated from MQSeries. Inbound propagation supports only RAW payloads. <strong>Note:</strong> The maximum length of payload for MQSeries V2, V5, and Oracle AQ is (respectively) 4M, 100M, and 32K. As such, payloads for inbound propagation are restricted to 32K.</td>
</tr>
<tr>
<td>msgid</td>
<td>Not translated; <strong>Note:</strong> Although IBM MQSeries allows applications to set msgid, Oracle AQ does not support this. Therefore, propagation of msgid values cannot be supported.</td>
</tr>
</tbody>
</table>
DBMS_AQ.DEQUEUE_OPTIONS_T

Upon dequeuing a message from an Oracle AQ destination queue using the dequeue() procedure, the Oracle AQ application can specify dequeue options. Refer to Oracle8 Application Developer’s Guide - Advanced Queuing for documentation on dbms_aq package. Inbound propagation for MQSeries will use the following translations:

<table>
<thead>
<tr>
<th>Oracle AQ Option</th>
<th>Translation for IBM MQSeries (inbound propagation)</th>
</tr>
</thead>
</table>
| consumer_name      | For multiple-consumer queues, Oracle AQ applications can specify the consumer name that matches the consumer that is specified upon scheduling inbound propagation. If the queue is not set up for multiple consumers, then this field should be set to NULL.  
**Note:** Also refer to dbms.hqadm.schedule_inbound_propagation(). |
| dequeue_mode       | N.A. inbound propagation has no effect on this field. |
| navigation         | N.A. inbound propagation has no effect on this option. |
| visibility         | N.A. inbound propagation has no effect on this option. |
| wait               | N.A., inbound propagation has no effect on this option. |
| msgid              | Oracle AQ applications can specify the message ID that was assigned to the message by the Oracle AQ system.  
**Note:** Oracle AQ applications should not try to specify the message ID that was assigned to the message by the non-Oracle message queuing system. Propagation of msgid values is not supported. |
| correlation        | Oracle AQ applications can specify the correlation ID that was assigned to the message by the originating non-Oracle application. |
Upon retrieving a message from an Oracle AQ destination queue using the `dequeue()` procedure, the Oracle AQ application receives a set of message properties. Refer to *Oracle8 Application Developer’s Guide - Advanced Queuing* for documentation on `dbms_aq` package. The values for these properties are set by the inbound propagation process using the following translations:

**Table 12–3 DBMS_AQ.MESSAGE_PROPERTIES_T**

<table>
<thead>
<tr>
<th>Oracle AQ Property</th>
<th>Translation for IBM MQSeries (inbound propagation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>priority</td>
<td>Translated from <code>MQMD.Priority</code>. Values 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 are translated respectively to 9, 8, 7, 6, 5, 4, 3, 2, 1, 0.</td>
</tr>
<tr>
<td>delay</td>
<td>No translation.</td>
</tr>
</tbody>
</table>
| expiration         | Translated from `MQMD.Expiry`. Value `MQEI_UNLIMITED` translated to `NEVER`. Values in seconds -> tenth of seconds.  
  **Note:** The expiration value in seconds is adjusted by subtracting the already-expired time since the message was enqueued at `MQMD.PutDate` and `MQMD.PutTime` in the MQSeries source queue. |
| correlation        | Translated from `MQMD.CorrelId`. |
| attempts           | No translation. |
| recipient_list     | No translation. |
| exception_queue    | No translation. |
| enqueue_time       | No translation.  
  **Note:** Upon propagation, the Oracle AQ system populates the `enqueue_time` field automatically. |
| state              | No translation. |
Configuring Inbound Message Propagation

Configurations must be performed on the Oracle server, on the Oracle Procedural Gateway for IBM MQSeries, and on the IBM MQSeries to facilitate message propagation from an IBM MQSeries queue to an Oracle AQ queue.

**Oracle integrating server:** Database link must be set up to point to the MQSeries gateway
**Gateway:** The gateway must be set up to be the proxy of the IBM MQSeries queue manager that owns the pickup queues
**MQSeries:** If the source queue is under the gateway queue manager, then no configuration is required. Otherwise, the pickup queue must be created under the gateway queue manager, and a remote queue must be created in the source queue’s queue manager to move messages to the pickup queue.

*Figure 12–1  Inbound Scheduling*
The following configuration is required in order to propagate a message from IBM MQSeries queue ‘ny11’ under queue manager ‘mgr.ny.1’ over to Oracle Advanced Queue ‘aq12’ under Oracle server ‘ora1’:

Oracle integrating server: Database link pg4mq in ora1 to point to the MQSeries gateway

Gateway: Configure the gateway with queue manager mgr.gtw and set up strict security mode.

MQSeries: Create pickup queue pkup1 under queue manager mgr.gtw, create remote queue ny11 under queue manager mgr.ny.1 for remote queue pkup1, and create transmission queue under queue manager mgr.ny.1 to remote queue manager mgr.gtw, if it does not exist yet.
Unlike outbound message propagation, as described in Chapter 11, no operational interface exists for Oracle applications to control inbound message propagation. Control occurs at the administrative level only (refer to the next section), and the whole inbound message propagation process appears almost transparent to the receiving Oracle application. Messages propagated into an Oracle AQ destination queue are dequeued in the same manner as any other message using the regular Oracle AQ interfaces as defined in the package DBMS_AQ.

Refer to the following code example with definitions of the Oracle AQ dequeue() procedure and the Oracle AQ types involved:

**Oracle AQ procedures and types used in heterogeneous inbound message propagation:**

```sql
dbms_aq.dequeue(
    queue_name      IN   varchar2,
    dequeue_options    IN   enqueue_options_t,
    message_properties  OUT   message_properties_t,
    payload        OUT   raw,
    msgid         OUT   raw)
```

**TYPE dequeue_options_t IS RECORD {**
```sql
    consumer_name     varchar2(30),
    dequeue_mode      binary_integer,
    navigation       binary_integer,
    visibility       binary_integer,
    wait          binary_integer,
    msgid        raw(16),
    correlation     varchar2(128))
```

**TYPE message_properties_t IS RECORD {**
```sql
    priority        binary_integer,
    delay          binary_integer,
    expiration      binary_integer,
    correlation     varchar2(128),
    attempts      binary_integer,
    recipient_list     aq$_recipient_list,
    exception_queue     varchar2(51),
    enqueue_time     date,
    state          binary_integer)
```
Refer to the Oracle AQ documentation for detailed information on how to use the `dequeue()` procedure and its arguments. For heterogeneous inbound message propagation, the usage is for the most part compatible with Oracle AQ propagation. However, some arguments have different behavior due to the limited capabilities of non-Oracle message queuing systems.

The `payload` argument contains the application data. The `payload` type should be compatible with the type of the queue specified in the `queue_name` argument, thus the `payload` type can be either an RAW or an ADT. Heterogeneous inbound message propagation supports only RAW queues. Therefore, only RAW payloads are supported.

The `msgid` argument is a message identifier assigned to the message by the Oracle AQ system. The `msgid` argument is globally unique within the Oracle domain. Heterogeneous inbound message propagation cannot pass the original `msgid` along with the message to the Oracle AQ system. For that reason, the `msgid` that is returned by the `dequeue()` procedure will not match the `msgid` that was assigned to messages by the non-Oracle message queuing system. Similarly, the `msgid` that is specified in the `dequeue_options` argument should match the value that was assigned by the Oracle AQ system to the propagated message. The `msgid` does not need to match the original value assigned by the non-Oracle message queuing system.

Although IBM MQSeries allows applications to set `msgid`, Oracle AQ and Microsoft MSMQ do not support this, because such propagation of `msgid` values cannot be supported.

---

**Note:** Although IBM MQSeries allows applications to set `msgid`, Oracle AQ and Microsoft MSMQ do not support this, because such propagation of `msgid` values cannot be supported.

---

Oracle AQ applications can use the `correlation` field for point-to-point message identification. The values of `correlation` will be preserved when they are propagated from non-Oracle message queuing systems to Oracle AQ.

An example of an Oracle application that is using the `dequeue()` procedure for a propagated message from MQSeries would look like the following:
**Example**

```plsql
DECLARE
  dequeue_options DBMS_AQ.DEQUEUE_OPTIONS_T;
  message_properties DBMS_AQ.MESSAGE_PROPERTIES_T;
  message_handle RAW(16);
  message RAW(255);
BEGIN
  /* Specify correlation as assigned by the MQSeries appl. */
  dequeue_options.correlation := 'AMQ qmgr.phx1029';
  /* dequeue the message from the AQ destination queue. */
  DBMS_AQ.DEQUEUE(queue_name  => 'aq2',
                   dequeue_options => queue_options,
                   message_properties => message_properties,
                   payload => message,
                   msgid => message_handle);
  COMMIT;
END;
/
```

Assuming that the Oracle AQ queue ‘aq2’ was used as the destination of an inbound message propagation schedule, the above example shows an Oracle AQ application attempting to dequeue a specific message from the destination queue by using `correlation`. Because the messages in ‘aq2’ were sent by an MQSeries application, the specified `correlation` should match a value that was assigned to the messages by this application. Considering this method of assigning `correlation` values, the Oracle application needs to take into account that the length of `correlation` values that are assigned by non-Oracle applications may differ from the length of `correlation` values that are assigned by an Oracle AQ. Other dequeue options and message properties may have similar issues.
Administrative Interface

For propagation in a homogeneous Oracle AQ environment, the administrator uses the Oracle AQ package, DBMS_AQADM, with procedures to schedule a propagation job. Specifying an Oracle AQ propagation job requires specifying three items:

1. an Oracle AQ source queue name
2. the destination
3. the scheduling times

As noted previously, inbound message propagation does not exist in homogeneous Oracle AQ propagation. Therefore, the inbound message propagation mechanism has defined its own API with procedures to schedule/unschedule propagation. The heterogeneous inbound administrative interface resembles the interface of homogeneous Oracle AQ propagation.

The procedures maintain schedules in dictionary tables. These procedures are read by the inbound message propagation process to determine when a queue, and which queue, is due for another round of message propagation. The schedule information is available through a view named DBA_INBOUND_SCHEDULES (refer to Appendix D, "Heterogeneous Message Propagation DBA Views"). The package that contains the scheduling procedures (both outbound and inbound) is named DBMS_AQADM. The following two procedures for inbound message propagation scheduling are defined in this package:

- SCHEDULE_INBOUND_PROPAGATION
- UNSCHEDULE_INBOUND_PROPAGATION

The two inbound scheduling procedures that are in the package are each composed of: name, purpose, syntax, usage, and error messages. These two inbound scheduling procedures are described next.
Procedures for Inbound Message Propagation

Name: SCHEDULE_INBOUND_PROPAGATION

Purpose
Upon calling, the procedure schedules propagation of messages from an non-Oracle AQ source queue to an Oracle AQ destination queue.

Syntax
```
dbms_hqadm.schedule_inbound_propagation(
    queue_name   in    varchar2,
    consumer    in    sys.aq$_agent,
    protocol    in    number,
    source     in    varchar2 default NULL,
    start_time   in    date default SYSDATE,
    duration    in    number default NULL,
    next_time    in    varchar2 default NULL,
    latency     in    number default 60 )
```

Usage

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>specifies the Oracle AQ destination queue to which the messages will be propagated. The name includes the schema name. If the schema name is not specified, then queue_name defaults to the schema name of the administrative user. Note: Heterogeneous inbound message propagation supports Oracle AQ destination queues of type RAW, and only type RAW.</td>
</tr>
<tr>
<td>consumer</td>
<td>If a queue (named queue_name) is created as a multiple consumer queue, then the inbound message propagation process will set the consumer name of the propagated messages to the value specified with this argument. If queue_name is not created as a multiple consumer queue, then this field should be set to NULL. Note: Creating multiple inbound schedules with the same queue_name and source, but a different value for consumer, is not allowed.</td>
</tr>
</tbody>
</table>
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>protocol</td>
<td>specifies the type of inbound message propagation protocol. Valid values: DBMS_HQADM.MQSERIES_INBOUND for IBM MQSeries.</td>
</tr>
<tr>
<td>source</td>
<td>is used to specify the address of the designated “pickup” queue at the non-Oracle message queuing system. Using Oracle gateways, the same database link mechanism that is used to specify a remote Oracle AQ server for homogeneous propagation can be used to specify a gateway for a non-Oracle message queuing system. Syntax: <code>&lt;non_oracle_queue&gt;@&lt;database_link&gt;</code> The syntax and length of the non-Oracle queue name are regulated by the non-Oracle message queuing system. These queue-name properties may differ from those of Oracle AQ.</td>
</tr>
<tr>
<td>start_time</td>
<td>Specifies the initial start time for the propagation window for messages from the source queue to the destination.</td>
</tr>
<tr>
<td>duration</td>
<td>specifies the duration of the propagation window in seconds. A NULL value means that the propagation window is forever, or until the propagation is unscheduled.</td>
</tr>
<tr>
<td>next_time</td>
<td>specifies the date function to compute the start of the next propagation window from the end of the current window.</td>
</tr>
<tr>
<td>latency</td>
<td>is the maximum wait, in seconds, in the propagation window for a message to be propagated after it is enqueued. However, if the latency is (for example) 60 seconds, and if no messages are waiting to be propagated, then during the propagation window, messages from that Oracle AQ source queue for the non-Oracle destination queue will not be propagated for at least 60 more seconds. If the latency is 0, then the heterogeneous propagation process will be waiting for messages to be enqueued for the destination, and as soon as a message is enqueued, it will be propagated.</td>
</tr>
</tbody>
</table>
Error Messages

ORA-20001 An invalid value was specified for QUEUE_ADDRESS. The value should be of the form queue@dblink[.domain]

Cause: The foreign queue specification for the source and/or destination parameter is either NULL or is not in the right format.

Action: Check the parameter, and correct the mistake.

ORA-20002 Heterogeneous propagation for ADT queues is not yet supported.

Cause: The Oracle AQ queue that was specified for heterogeneous propagation is an ADT queue, which is not supported.

Action: Specify a RAW queue.

ORA-20003 This error occurred while creating a MIP for this schedule.

Cause: The propagation package has a problem creating the interface to the foreign queue system for the schedule.

Action: This is an internal error. Report this error to Oracle.

ORA-20006 A message consumer is not required for inbound message propagation into a single-consumer queue.

Cause: The consumer parameter has a non-NULL value, and the destination Oracle AQ queue is a single consumer queue.

Action: Give a NULL value to the consumer parameter.

ORA-20007 A Message Consumer is required for inbound message propagation into a multiple-consumer queue.

Cause: The consumer parameter has a NULL value, and the destination Oracle AQ queue is a multiple-consumer queue.

Action: Give a NULL value to the consumer parameter.

ORA-20008 An inbound schedule exists between the specified foreign source queue and the specified Oracle AQ queue.

Cause: The existing schedule must be removed first.

Action: Remove the old schedule before scheduling the new one.
Name: UNSCHEDULE_INBOUND_PROPAGATION

Purpose
The UNSCHEDULE_INBOUND_PROPAGATION procedure unschedules previously scheduled heterogeneous inbound message propagation from a non-Oracle source queue to an Oracle AQ destination. If no messages are left in the propagation-exception queue, then it will be deleted again. If the exception queue still holds messages, then the procedure will fail unless the user specifies the force option as TRUE (causing the messages on the propagation queue to be purged and causing the queue to be cleaned up).

Refer to "Exception Handling" on page 12-19 for more information.

Syntax

dbms_hqadm.unschedule_propagation(
    queue_name   in    varchar2,
    protocol    in    number,
    source     in    varchar2 default NULL,
    force     in    boolean default FALSE)

Usage

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue_name</td>
<td>specifies the name of the destination queue whose inbound message propagation schedule should be ended. The name includes the schema name. If the schema name is not specified, then queue_name defaults to the schema name of the administrative user.</td>
</tr>
<tr>
<td>protocol</td>
<td>specifies the type of inbound message propagation protocol. Valid values: DBMS_HQADM.MQSERIES_INBOUND for IBM MQSeries.</td>
</tr>
</tbody>
</table>
### Error Messages

**ORA-20001** An invalid value was specified for QUEUE_ADDRESS. The value should be of the form `queue@dblink[.domain]`.

- **Cause:** The foreign queue specification for the `source` or `destination` parameter (or both) is either NULL or is not in the right format.
- **Action:** Check the parameter, and correct the mistake.

**ORA-20009** No inbound schedule exists between the specified foreign source queue and the specified Oracle AQ queue.

- **Cause:** The specified schedule cannot be found in the schedule table.
- **Action:** Check the parameters.

### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>is used to specify the address of the designated “pickup” queue at the non-Oracle message queuing system whose inbound message propagation schedule should be ended. Using Oracle gateways, the same database link mechanism that is used to specify a remote Oracle AQ server for homogeneous propagation can be used to specify a gateway for a non-Oracle message queuing system. Syntax: <code>&lt;non_oracle_queue&gt;@&lt;database_link&gt;</code> The syntax and length of the non-Oracle queue name are regulated by the non-Oracle message queuing system. These queue-name properties may differ from those of Oracle AQ.</td>
</tr>
<tr>
<td>force</td>
<td>specifies whether the procedure should succeed even if messages are left behind on the propagation-exception or transmit queues. Refer to &quot;Exception Handling&quot; on page 12-19 for more information. Valid values: TRUE: Any message that is left on the exception queue will be purged, and the queues will be cleaned up. FALSE: The procedure will fail when messages are found on the exception queue.</td>
</tr>
</tbody>
</table>
Example

Continuing on the earlier example of scheduling propagation, the following code example demonstrates how Oracle AQ administrators would schedule homogeneous message propagation, heterogeneous outbound message propagation, and heterogeneous inbound message propagation.

BEGIN

/* schedule propagation for AQ */
DBMS_AQADM.SCHEDULE_PROPAGATION(queue_name => 'aq1',
                                 destination => 'remoteAq');

/* schedule outbound propagation for MQSeries */
DBMS_HQADM.SCHEDULE_PROPAGATION(
    queue_name => 'mq1',
    protocol => DBMS_HQADM.MQSERIES_OUTBOUND,
    destination =>'mq1@mqseries');

/* schedule inbound propagation from MQSeries */
DBMS_HQADM.SCHEDULE_INBOUND_PROPAGATION(
    queue_name => 'aq2',
    consumer => 'JOE',
    protocol => DBMS_HQADM.MQSERIES_INBOUND,
    source => 'mq2@mqseries');

END;
/

In the above example, inbound message propagation is scheduled for an MQSeries queue named 'mq2' and for an Oracle AQ destination queue named 'aq2'. The source queue 'mq2' is accessed using an MQSeries gateway that is defined by the database link 'mqseries'.


Configuring Heterogeneous Inbound Message Propagation 12-17
Security

Heterogeneous Message Propagation does not impose new authorization requirements on either Oracle AQ or non-Oracle operational applications. For inbound message propagation, a non-Oracle application enqueues the message to the designated source queue at the non-Oracle message queuing system, but does not require authorization for the recipient queue controlled by Oracle AQ. Alternatively, the Oracle consumer application requires authorization only for the Oracle AQ destination queue.

At the administration level, the user who schedules inbound message propagation for a src_address and/or destination tuple:

- requires authorization to access the Oracle AQ destination queue
- requires access to a database link for a gateway in order to access the source queue at the non-Oracle message queuing system

The user name and password that were specified for the database link require access privileges at the message queuing system of the source queue. If the database link does not specify a user name and password, then it will implicitly use the ID of the current Oracle user.

Given these privileges, the propagation process can dequeue the messages from the source queue using the gateway and can enqueue them into the Oracle AQ destination queue. This ensures that the security issue for inbound message propagation remains transparent to both sender and consumer applications.
Exception Handling

If the inbound message propagation process cannot propagate messages from the non-Oracle source queue to the Oracle AQ destination queue, the messages will be moved to a “propagation exception” queue.

propagation exception queue

For each schedule that is created by using the dbms_hqadm.schedule_inbound_propagation() procedure, an exception queue is created to stage messages that could not be successfully moved to the Oracle AQ destination queue. Administrators can check the exception_queue field in the table DBA_INBOUND_SCHEDULES (refer to Appendix D, "Heterogeneous Message Propagation DBA Views") to obtain the name of this exception queue and to monitor it.

Upon unscheduling a propagation job by using the dbms_hqadm.unschedule_inbound_propagation() procedure, the exception queue is removed again (for additional details, refer to the description of this procedure at "Name: SCHEDULE_INBOUND_PROPAGATION" on page 12-12).

Schedule Status

Status Information can be obtained for an inbound message propagation schedule and for individual messages.

Status of an Inbound Message Propagation Schedule

The administrator can check the view DBA_INBOUND_SCHEDULES to determine if inbound message propagation has been scheduled for a particular combination of Oracle AQ destination queue and non-Oracle source queue. For each created schedule, the inbound message propagation process maintains the names of the exception queue as well as up to date status and statistics in the DBA_INBOUND_SCHEDULES view, among other information.

For example, to find all schedules that are not running because of a failure, and to find the reason for the failure, the following query can be executed:

```
SELECT ERROR_TIME, ERROR, ERROR_MSG
FROM DBA_INBOUND_SCHEDULES
WHERE DISABLED = 'Y';
```
The PGM8, PGM_UTL8, and PGM_SUP Packages

Use the Visual Workbench when developing applications that access MQSeries through the gateway. The Visual Workbench defines an interface for accessing MQSeries and automatically generates the PL/SQL code (the MIP) for Oracle applications to interface with the gateway. Refer to the Oracle Visual Workbench for Oracle Procedural Gateways for IBM MQSeries Installation and User’s Guide for more information about Visual Workbench.

The MIP uses definitions of the PGM8, PGM_UTL8, and PGM_SUP packages. When necessary, you can alter the MIP to include MQSeries functions that are not supported by Visual Workbench. This is done with the definitions and procedures of the PGM8, PGM_UTL8, and PGM_SUP.

The PGM8, PGM_UTL8, and the PGM_SUP packages are installed when the Visual Workbench repository or the PG4MQ deployment environment is created. For more information, refer to "Installing the Visual Workbench Repository" on page 7-22 and "Preparing the Production Oracle Server" on page 7-28.

- PGM8 Package and PG4MQ Gateway Procedures on page A-2
- MQCLOSE Procedure on page A-8
- MQGET Procedure on page A-9
- MQOPEN Procedure on page A-20
- MQPUT Procedure on page A-24
- PGM_SUP Package on page A-29
PGM8 Package and PG4MQ Gateway Procedures

The gateway procedures and type definitions of the PGM8 package are modeled after the MQSeries MQI calls. For relevant calls and structures found in MQI, a counterpart exists in PGM8. The gateway procedures and PGM8 type definitions are named the same as their MQI counterparts. However, the data types of arguments or structure fields are changed into corresponding PL/SQL data types.

Using these procedures and type definitions in an Oracle application is very similar to writing an MQSeries application. The fields of all PGM8 type definitions are initialized. These initialization values are based on default values defined by MQI.

Use of gateway procedures and PGM8 type definitions requires extensive knowledge of MQI and MQSeries programming in general. These procedures and records follow the MQI flow-chart, semantics, and syntax rules.

The PGM8 package is installed when the Visual Workbench repository or the PG4MQ deployment environment is created and is granted public access. It has no schema because the gateway omits all schema names when describing or executing a procedure. No schema qualifiers need to be prefixed to the names of the procedures and type definitions.

For complete information about writing MQSeries applications and using MQI calls, refer to the IBM MQSeries Application Programming Reference.

Summary of Procedures and Type Definitions

The gateway procedures and PGM8 provide the following procedures and type definitions:

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Procedure Purpose</th>
<th>Type Definitions Used by the Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQOPEN</td>
<td>Opens a queue.</td>
<td>PGM8.MQOD and PGM8.MQODRAW</td>
</tr>
<tr>
<td>MQPUT</td>
<td>Sends a message to the queue that was opened by MQOPEN.</td>
<td>PGM8.MQMD and PGM8.MQMDRAW  PGM8.MQPMO and PGM8.MQPMORAW</td>
</tr>
<tr>
<td>MQGET</td>
<td>Retrieves or scans a message from the queue that was opened by MQOPEN.</td>
<td>PGM8.MQMD and PGM8.MQMDRAW  PGM8.MQGMO and PGM8.MQGMORAW</td>
</tr>
<tr>
<td>MQCLOSE</td>
<td>Closes the queue that was opened by MQOPEN.</td>
<td>Does not use a type definition.</td>
</tr>
</tbody>
</table>
Procedure Conventions

The gateway procedures are described in alphabetical order in this appendix. The type definitions are described with the procedures that use them. Only type definition fields that can be changed are described. Other fields equivalent to MQI fields are left out because they are reserved for MQSeries, are not supported by the gateway, or contain values that should not be changed.

A procedure’s definition is shown using the IBM argument names associated with the equivalent MQI call. For example:

```
MQGET(hobj, mqmd, mqgmo, msg)
```

You can use your own names for these arguments if you code the arguments in the order shown in the definition. For example,

```
MQGET(handle, descript, get_options, message);
```

where:

- `handle` is your name for the first argument specified in the definition as `hobj`.
- `descript` is your name for the second argument specified in the definition as `mqmd`.
- `get_options` is your name for the third argument specified in the definition as `mqgmo`.
- `message` is your name for the fourth argument specified in the definition as `msg`.

If you code the arguments in a different order than the definition, then you must use the IBM names to identify your argument names using the positional notation syntax of PL/SQL. For example:

```
MQGET(msg=>message, mqgmo=>opts, hobj=>handle,
       mqmd=>descript);
```

For more information about PL/SQL, refer to the PL/SQL User’s Guide and Reference.
MQI Calls Performed by the Gateway

The following MQI calls have no equivalent procedures in the gateway because the Oracle integrating server and the gateway automatically perform the functions of these MQI calls:

- MQBACK
  Transaction control is handled by the Oracle transaction coordinator. The Oracle application does not need to invoke a separate MQBACK call to undo the changes sent to MQSeries.

- MQCONN
  A connection to a queue manager is established by the Oracle integrating server and the gateway whenever an Oracle application refers to a gateway procedure. The database link name that is used when calling the gateway procedure determines which queue manager the gateway connects to.

- MQCMIT
  Transaction control is handled by the Oracle transaction coordinator. An Oracle application does not need to invoke a separate MQCMIT call to commit the changes sent to MQSeries.

- MQDISC
  Connections to a queue manager are closed by the Oracle integrating server and gateway. An Oracle application does not need to close the connection with the queue manager. Ending the current Oracle session or dropping the database link causes the queue manager connection to end.

Unsupported MQI Calls

The following MQI calls are not supported by the gateway:

- MQINQ
- MQPUT1
- MQSET
Migration Tips

This section offers help to users of PG4MQ release 8.0.4 and prior as they upgrade their existing customized PL/SQL application programs to use Oracle9i PG4MQ features.

PG4MQ data types and RPC API prototypes are changed to meet the requirements of the gateway infrastructure, which has been changed since PG4MQ release 8.0.4.1.0 was introduced.

When upgrading your version 4 PG4MQ to version 8 or version 9, Oracle Corporation recommends that you install the newer version of PG4MQ on a separate development Oracle system. After you have finished with system configuration and testing, transfer all of the COBOL copy books and regenerate and recompile MIPs by using the VWB (Visual Workbench). For customized codes, make necessary changes and recompile.

PG4MQ release 4.0.1 users must perform the following three steps to upgrade their customized PL/SQL application programs to use Oracle9i PG4MQ:

1. Remove dblink references from the following PG4MQ data types:
   - PGM.MQOD
   - PGM.MQMD
   - PGM.MQPMO
   - PGM.MQGMO
   and change "PGM" to "PGM8". For example, change "PGM.MQOD@dblink" to "PGM8.MQOD".

2. Include the following new PG4MQ data types in your PL/SQL programs:
   - PGM8.MQODRAW
   - PGM8.MQMDRAW
   - PGM8.MQPMORAW
   - PGM8.MQGMORAW
3. Make changes to the following PG4MQ procedures for the prototype changes:

MQOPEN()
MQPUT()
MQGET()

V4:

MQOPEN(mqod, options, hobj);

V9:

mqodRaw := PGM_UTL8.TO_RAW(mqod);
MQOPEN(mqodRaw, options, hobj);
mqod := PGM_UTL8.RAW_TO_MQOD(mqodRaw);

V4:

MQPUT(hobj, mqmd, mqpmo, putbuffer);

V9:

mqmdRaw := PGM_UTL8.TO_RAW(mqmd)
mqpmoRaw := PGM_UTL8.TO_RAW(mqpmo)
MQPUT(hobj, mqmdRaw, mqpmoRaw, putbuffer);
mqmd := PGM_UTL8.RAW_TO_MQMD(mqmdRaw);
mqpmo := PGM_UTL8.RAW_TO_MQPMO(mqpmoRaw);

V4:

MQGET(hobj, mqmd, mqgmo, getbuffer);

V9:

mqmdRaw := PGM_UTL8.TO_RAW(mqmd)
mqgmoRaw := PGM_UTL8.TO_RAW(mqgmo)
MQGET(hobj, mqmdRaw, mqgmoRaw, getbuffer);
mqmd := PGM_UTL8.RAW_TO_MQMD(mqmdRaw);
mqgmo := PGM_UTL8.RAW_TO_MQGMO(mqgmoRaw);
PG4MQ release 8.0.4 users must perform the following 2 steps to upgrade their customized PL/SQL application programs to use Oracle9i PG4MQ:

1. Change the package name "PGM" to "PGM8" for the following PG4MQ data types:
   - PGM.MQMD
   - PGM.MQPMO
   - PGM.MQGMO
   - PGM.MQODRAW
   - PGM.MQMDRAW
   - PGM.MQPMORAW
   - PGM.MQGMRRAW

   For example, change "PGM.MQOD" to "PGM8.MQOD".

2. Change all the references for package name "PGM_UTL" to use "PGM_UTL8" for the following procedures:
   - PGM_UTL.TO_RAW();
   - PGM_UTL.RAW_TO_MQMD();
   - PGM_UTL.RAW_TO_MQPMO();
   - PGM_UTL.RAW_TO_MQGMO();

   For example, change "PGM_UTL.TO_RAW" to "PGM_UTL8.TO_RAW"
MQCLOSE Procedure

MQCLOSE closes a queue. Upon return, the queue handle is invalid and your application must reopen the queue with another call to MQOPEN before issuing another MQPUT, MQGET, or MQCLOSE call to the queue.

MQCLOSE differs from MQI calls in the following ways:

■ The connection handle argument is omitted from MQCLOSE because the gateway automatically takes care of managing queue manager connections.

■ The MQI completion code is not included in the procedure’s argument list. When a gateway procedure fails because the corresponding MQI call failed, then an Oracle error message is returned to the caller.

■ The MQI reason code is not included in the procedure’s argument list. When the corresponding MQI call for a gateway procedure returns a reason code, then the reason code is included in the Oracle error message returned to the caller.

Definition

MQCLOSE(hobj, options)

where:

hobj contains the handle for the queue to close. The handle was returned by a previous call to MQOPEN. This input argument is PL/SQL data type BINARY_INTEGER.

options specifies the close action. Use PGM_SUP.MQCO_NONE or the other PGM_SUP constants for a close option. Refer to "MQCLOSE Values" on page A-36. This input argument is PL/SQL data type BINARY_INTEGER.

Examples

1. Using your own variable names when arguments are in the required order:

   MQCLOSE(handle, close_options);

2. Using your own variable names when arguments are in any order:

   MQCLOSE(options=>close_options, hobj=>handle);
MQGET Procedure

MQGET retrieves a message from a queue. The queue must already be open from a previous call to MQOPEN with the option PGM_SUP.MQOO_INPUT_AS_Q_DEF (or an equivalent option) set. Retrieved messages for this form of MQGET must be shorter than 32,767 bytes.

MQGET differs from MQI calls in the following ways:

1. The connection handle argument is omitted from MQGET because the gateway automatically takes care of managing queue manager connections.

2. The MQI completion code is not included in the procedure’s argument list. When a gateway procedure fails because the corresponding MQI call failed, then an Oracle error message is returned to the caller.

3. The MQI reason code is not included in the procedure’s argument list. When the corresponding MQI call for a gateway procedure returns a reason code, then the reason code is included in the Oracle error message that was returned to the caller.

4. The msg length argument is not included in the procedure’s argument list because the Oracle integrating server and the gateway automatically keep track of the message data length.

Definition

MQGET(hobj, mqmdRaw, mqgmoRaw, msg)

where:

hobj contains the handle for the queue to open. The handle is returned by a previous call to MQOPEN. This input argument is PL/SQL data type BINARY_INTEGER.
mqmdRaw is used on input to describe the attributes of the message being retrieved. Use the fields of the PGM8.MQMD type definition to describe these attributes, and use `PGM_UTL8.TO_RAW()` to convert the PGM8.MQMD data type to `mqmdRaw` before the `MQGET()`.

On output, `mqmdRaw` contains information about how the request was processed. The queue manager sets some of the PGM8.MQMD fields upon return. Use `PGM_UTL8.RAW_TO_MQMD()` to convert the `mqmdRaw` back to PGM8.MQMD data type after the `MQGET()` in order to process the PGM8.MQMD fields.

This input and output argument is PL/SQL RAW data type. For the details of PGM8.MQMD, refer to "PGM8.MQMD Type Definition" on page A-14.

mqgmoRaw is used on input to describe the option values that control the retrieve request. Use the fields of the PGM8.MQGMO type definition to describe these options, and use `PGM_UTL8.TO_RAW()` to convert the PGM8.MQGMO data type to `mqgmoRaw` before the `MQGET()`.

On output, the queue manager sets some of the PGM8.MQGMO fields upon return. Use `PGM_UTL8.RAW_TO_MQGMO()` to convert the `mqgmoRaw` back to PGM8.MQGMO data type after the `MQGET()` in order to process the PGM8.MQGMO fields.

This input and output argument is PL/SQL RAW data type. For the details of PGM8.MQGMO, refer to "PGM8.MQGMO Type Definition" on page A-19.

msg contains the retrieved message. This output argument is PL/SQL data type RAW.
Examples

1. Using your own variable names when arguments are in the required order:

   MQGET(handle, descrip, opts, message);

2. Using your own variable names when arguments are in any order:

   MQGET(msg=>message, mqgmo=>opts, hobj=>handle, 
   mqmd=>descript);

3. The following example, which is provided as a sample with the gateway 
   (ORACLE_HOME/pg4mqseries/sample/getsample.sql), reads all 
   messages from an MQSeries queue. For more information, refer to the IBM 
   publication referring to MQSeries Application Programming.
Example

-- Copyright Oracle Corporation, 2002 All Rights Reserved.
--
-- NAME
-- getsample.sql
--
-- DESCRIPTION
-- Specify the database link name you created for the gateway. To do this,
-- replace the characters @pg4mq with @dblink, where dblink is the name
-- you chose when the database link was created.
--
-- This script performs a test run for the MQSeries gateway. In this
-- script the queuename is 'YOUR_QUEUE_NAME', replace it with a valid
-- queue name at the queue manager the gateway is configured for.
--
-- NOTES
-- Run the script from the SQL*Plus command line.
--
-- Make sure the script dblink.sql is executed to create the
-- database link to the gateway.
--
-- Make the sure the user is granted 'EXECUTE' on package dbms_output
--
--
SET SERVEROUTPUT ON

DECLARE
    objdesc   PGM8.MQOD;
    msgDesc   PGM8.MQMD;
    getOptions PGM8.MQGMO;
    objectHandle binary_integer;
    message   raw(32767);
    mqodRaw   PGM8.MQODRAW;
    mqmdRaw   PGM8.MQMDRAW;
    mqpmoRaw  PGM8.MQPMORAW;
    mqgmoRaw  PGM8.MQGMORAW;
BEGIN
  -- Open the queue 'YOUR_QUEUE_NAME' for reading.
  objdesc.OBJECTNAME := 'YOUR_QUEUE_NAME';
  -- Convert PGM8.MQMD to RAW
  mqodRaw := PGM_UTL8.TO_RAW(objdesc);
  MQOPEN@pg4mq(mqodRaw, PGM_SUP.MQOO_INPUT_AS_Q_DEF, objectHandle);
  -- Convert RAW to PGM8.MQMD
  objdesc := PGM_UTL8.RAW_TO_MQOD(mqodRaw);
  -- Get all messages from the queue.
  WHILE TRUE LOOP
    -- Reset msgid and correlid to get the next message.
    msgDesc.MSGID := UTL_RAW.CAST_TO_VARCHAR2(PGM_SUP.MQMI_NONE);
    msgDesc.CORRELID := UTL_RAW.CAST_TO_VARCHAR2(PGM_SUP.MQCI_NONE);
    -- Convert PGM8.MQMD and PGM8.MQGMO to RAW
    mqmdRaw := PGM_UTL8.RAW_TO_MQMD(msgDesc);
    mqgmoRaw := PGM_UTL8.RAW_TO_MQGMO(getOptions);
    MQGET@pg4mq(objectHandle, mqmdRaw, mqgmoRaw, message);
    -- Convert RAW to PGM8.MQMD and PGM8.MQGMO
    msgDesc := PGM_UTL8.RAW_TO_MQMD(mqmdRaw);
    getOptions := PGM_UTL8.RAW_TO_MQGMO(mqgmoRaw);
    -- Process the message....
    DBMS_OUTPUT.PUT_LINE('message read back = ' || rawtohex(message));
    COMMIT;
  END LOOP;
EXCEPTION
  WHEN PGM_SUP.NO_MORE_MESSAGES THEN
    DBMS_OUTPUT.PUT_LINE('Warning: No more message found on the queue');
    -- Close the queue again.
    MQCLOSE@pg4mq(objectHandle, PGM_SUP.MQCO_NONE);
  WHEN OTHERS THEN
    -- Re-raise the error.
    DBMS_OUTPUT.PUT_LINE('Error: Procedural Gateway for IBM MQSeries verification script failed.');
    DBMS_OUTPUT.PUT_LINE(SQLERRM);
    raise;
END;
Notes:

- The PL/SQL block fails if the exception clause is left out. In that case, the error code PGM_SUP.NO_MORE_MESSAGES is raised.
- The MSGID and CORRELID fields that are used for MQGET are set after each call to MQGET. If they are not reset at each cycle, then MQGET checks for the next message that has the same identifiers as the last read operation, which usually do not exist. The PL/SQL block would only read one message.

**PGM8.MQMD Type Definition**

PGM8.MQMD specifies the control information that accompanies a message when it travels between the sending and receiving applications. It also contains information about how the message is handled by the queue manager or by the receiving application. PGM8.MQMD describes the attributes of the message being retrieved. Use PGM_UTL8.TO_RAW() to convert the PGM8.MQMD data type to PL/SQL RAW data type and pass it to MQPUT or MQGET procedure. An example of using PGM8.MQMD is on page A-267.

You can use the default values for the PGM8.MQMD fields or change the fields for your application’s requirements. For example, to change a field value:

```sql
mqmd.field_name := field_value;
mqmdRaw := PGM_UTL8.TO_RAW(mqmd);
```

where:

- `mqmd` is the PGM8.MQMD data type and it describe the attributes of the message being retrieved
- `field_name` is a field name of the PGM8.MQMD type definition. You can set as many fields as necessary. Refer to Table A-2 for field names and descriptions.
- `field_value` is the value to assign to `field_name`. You can specify a value or use a PGM_SUP constant to assign a value.
- `mqmdRaw` is an input output argument for the MQPUT or MQGET procedure
### Table A–2  PGM8.MQMD Fields

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
<th>PL/SQL Datatype</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPORT</td>
<td>allows the application that sends a message to specify which report message (or messages) should be created by the queue manager when an expected or unexpected event occurs. Use a PGM_SUP constant to assign a value. Refer to &quot;REPORT Field&quot; on page A-34.</td>
<td>RAW(4)</td>
<td>PGM_SUP.MQRO_NONE</td>
</tr>
<tr>
<td>MSGTYPE</td>
<td>specifies the message type: reply message, report message, or normal message (datagram). Use a PGM_SUP constant to assign a value. Refer to “MSGTYPE Field” on page A-32.</td>
<td>BINARY_INTEGER</td>
<td>PGM_SUP.MQMT_DATAGRAM</td>
</tr>
<tr>
<td>EXPIRY</td>
<td>specifies the amount of time that a message stays in a queue. The expiration period is in tenths of a second, and is set by the sending application. Use a PGM_SUP constant to assign a value. Refer to “EXPIRY Field” on page A-31.</td>
<td>BINARY_INTEGER</td>
<td>PGM_SUP.QMEI_UNLIMITED</td>
</tr>
<tr>
<td>FEEDBACK</td>
<td>is used with the REPORT field to indicate the kind of report. Use a PGM_SUP constant to assign a value. Refer to “FEEDBACK Field” on page A-31.</td>
<td>BINARY_INTEGER</td>
<td>PGM_SUP.MQFB_NONE</td>
</tr>
<tr>
<td>ENCODING</td>
<td>is used for numeric values in the message data. Use a PGM_SUP constant to assign a value. Refer to &quot;ENCODING Field&quot; on page A-30.</td>
<td>RAW(4)</td>
<td>PGM_SUP.MQENC_NATIVE</td>
</tr>
<tr>
<td>CODEDCHARSETID</td>
<td>specifies the coded character set identifier of the characters in the message. Use a PGM_SUP constant to assign a value. Refer to &quot;CODEDCHARSETID Field&quot; on page A-30.</td>
<td>BINARY_INTEGER</td>
<td>PGM_SUP.MQCCSI_DEFAULT</td>
</tr>
</tbody>
</table>
### MQGET Procedure

#### Table A–2  PGM8.MQMD Fields

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
<th>PL/SQL Datatype</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORMAT</td>
<td>is a free format name that is used to inform the receiver about the contents of the message. Specify a format or use a PGM_SUP constant. Refer to &quot;FORMAT Field&quot; on page A-32.</td>
<td>CHAR(8)</td>
<td>PGM_SUP.MQFMT_NONE</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>specifies a message's priority. Specify a value greater than or equal to zero (zero is the lowest priority), or use a PGM_SUP constant. Refer to &quot;PRIORITY Field&quot; on page A-32.</td>
<td>BINARY_INTEGER</td>
<td>PGM_SUP.MQPRI_PRIORITY_AS_Q_DEF</td>
</tr>
<tr>
<td>PERSISTENCE</td>
<td>is an input field for the sending application. Persistent messages survive when a queue manager is restarted. Non-persistent messages and messages in temporary queues are lost when a queue manager is restarted. Specify the desired persistence with a PGM_SUP constant. Refer to &quot;PERSISTENCE Field&quot; on page A-32.</td>
<td>BINARY_INTEGER</td>
<td>PGM_SUP.MQPER_PERSISTENCE_AS_Q_DEF</td>
</tr>
<tr>
<td>MSGID</td>
<td>specifies the message identifier of the message to be retrieve (when receiving a message). If no value is specified when a sending a message (PGM_SUP.MQMI_NONE), then the queue manager assigns a unique value.</td>
<td>CHAR(24)</td>
<td>UTL_RAW.CAST_TO VARCHAR2(PGM_SUP.MQMI_NONE)</td>
</tr>
<tr>
<td>CORRELID</td>
<td>specifies the correlation identifier for the message to retrieve when receiving a message (refer to the MSGID field). When sending a message, specify any value, or use PGM_SUP.MQCI_NONE if the message does not require a correlation ID.</td>
<td>CHAR(24)</td>
<td>UTL_RAW.CAST_TO VARCHAR2(PGM_SUP.MQMI_NONE)</td>
</tr>
<tr>
<td>BACKOUTCOUNT</td>
<td>is an output field for the MQGET procedure. It indicates the number of times a message was placed back on a queue because of a rollback operation.</td>
<td>BINARY_INTEGER</td>
<td>Zero</td>
</tr>
</tbody>
</table>
MQGET Procedure

**Table A–2  PGM8.MQMD Fields**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
<th>PL/SQL Datatype</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPLYTOQ</td>
<td>specifies the name of the reply-to queue. This is an input field for MQPUT and allows the sending application to indicate where reply messages should be sent. It is also an output field for MQGET and tells the receiving application where to send a reply.</td>
<td>CHAR(48)</td>
<td>NULL</td>
</tr>
<tr>
<td>REPLYTOQMGR</td>
<td>specifies the queue manager to which the reply message or report should be sent. This is an input field for MQPUT and an output field for MQGET.</td>
<td>CHAR(48)</td>
<td>NULL</td>
</tr>
<tr>
<td>USERIDENTIFIER</td>
<td>is an output field for receiving applications. It identifies the user that sent the message. Sending applications can specify a user on input if the CONTEXT field for the macro argument of MQPUT was set to PGM_SUP.MQPMO_SET_IDENTITY_CONTEXT or to PGM_SUP.MQPMO_SET_ALL_CONTEXT.</td>
<td>CHAR(12)</td>
<td>NULL</td>
</tr>
<tr>
<td>ACCOUNTINGTOKEN</td>
<td>is used to transfer accounting information between applications. Sending applications provide accounting information or use PGM_SUP.MQACT_NONE to specify that no accounting information is included.</td>
<td>CHAR(32)</td>
<td>PGM_SUP.MQACT_NONE</td>
</tr>
<tr>
<td>APPLIDENTITYDATA</td>
<td>specifies more information to send along with the message to help the receiving application provide more information about the message or its sender.</td>
<td>CHAR(32)</td>
<td>NULL</td>
</tr>
<tr>
<td>PUTAPPLTYPE</td>
<td>describes the kind of application that placed the message on the queue. Use a PGM_SUP constant to assign a value. Refer to &quot;PUTAPPLTYPE Field&quot; on page A-33.</td>
<td>BINARY_INTEGER</td>
<td>PGM8.MQAT_NO_CONTEXT</td>
</tr>
</tbody>
</table>
### Table A–2  PGM8.MQMD Fields

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
<th>PL/SQL Datatype</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUTAPPLNAME</td>
<td>specifies the name of the application that placed the message on the queue. Sending applications specify a name or let the queue manager fill in this field. This is an output field for receiving applications.</td>
<td>CHAR(28)</td>
<td>NULL</td>
</tr>
<tr>
<td>PUTDATE</td>
<td>specifies the date on which a message was placed on the queue. Sending applications can set a date or let the queue manager take care of it. The date format used by the queue manager is YYYYMMDD. This is an output field for receiving applications.</td>
<td>CHAR(8)</td>
<td>NULL</td>
</tr>
<tr>
<td>PUTTIME</td>
<td>specifies the time that a message was placed on the queue. Sending applications can set a time or let the queue manager take care of it. The time format that is used by the queue manager is HHMMSSTH. This is an output field for receiving applications.</td>
<td>CHAR(8)</td>
<td>NULL</td>
</tr>
<tr>
<td>APPLORIGINDATA</td>
<td>is used by the sending application to add information to the message about the message origin. This is an output field for receiving applications.</td>
<td>CHAR(4)</td>
<td>NULL</td>
</tr>
</tbody>
</table>
PGM8.MQGMO Type Definition

PGM8.MQGMO is used to specify option and control information about how the message is retrieved from a queue. We use `PGM_UTL8.TO_RAW()` to convert the PGM8.MQGMO data type to PL/SQL RAW data type and pass it to MQGET procedure. An example of using "PGM8.MQGMO Type Definition" on page A-19.

You can use the default values for the PGM8.MQGMO fields or change the fields for your application’s requirements. For example, to change a field value:

```plaintext
mqgmo.field_name := field_value
mqgmoRaw := PGM_UTL8.TO_RAW(mqgmo);
```

where:

- `mqgmo` is the PGM8.MQGMO data type, and it specifies option and control information about how the message is retrieved from a queue.
- `field_name` is a field name of the PGM8.MQGMO type definition. You can set as many fields as necessary. Refer to Table A–3 for names and field descriptions.
- `field_value` is the value to assign to `field_name`. You can specify a value or use a PGM_SUP constant to assign a value.
- `mqgmoRaw` is an input output argument for the MQGET procedure.
MQOPEN Procedure

MQOPEN Procedure

MQOPEN establishes access to a queue. Depending on the mode selected to open the queue, an application can issue subsequent MQPUT, MQGET, or MQCLOSE calls.

MQOPEN differs from MQI calls in the following ways:

1. The connection handle argument is omitted from MQOPEN because the gateway automatically takes care of managing queue manager connections.

2. The MQI completion code is not included in the procedure’s argument list. When a gateway procedure fails because the corresponding MQI call failed, then an Oracle error message is returned to the caller.

3. The MQI reason code is not included in the procedure’s argument list. When the corresponding MQI call for a gateway procedure returns a reason code, then the reason code is included in the Oracle error message that is returned to the caller.

Table A–3  PGM8.MQGMO Fields

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
<th>PL/SQL Datatype</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTIONS</td>
<td>specifies options to control the MQGET procedure. Add one or more PGM_SUP</td>
<td>BINARY_INTEGER</td>
<td>PGM8.MQGMO_SYNCPOINT</td>
</tr>
<tr>
<td></td>
<td>constants to set it. Refer to &quot;OPTIONS Field” on page A-29.</td>
<td></td>
<td>(Messages that are retrieved from</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>the queue are coordinated by the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Oracle transaction coordinator.)</td>
</tr>
<tr>
<td>WAITINTERVAL</td>
<td>specifies the maximum time in milliseconds that MQGET waits for a message</td>
<td>BINARY_INTEGER</td>
<td>Zero</td>
</tr>
<tr>
<td></td>
<td>to arrive in the queue. WAITINTERVAL should be equal to or greater than</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>zero, or set to the value of PGM_SUPMQWI_UNLIMITED (unlimited wait interval).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESOLVEDQNAME</td>
<td>contains the resolved name of the destination queue from which the message</td>
<td>CHAR(48)</td>
<td>NULL</td>
</tr>
<tr>
<td></td>
<td>was retrieved. This is an output field set by the queue manager upon return</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>from the call.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Definition**

MQOPEN(mqodRaw, options, hobj)

where:

- **mqodRaw** specifies the queue to open. Use the fields of PGM8_MQOD type definition to describe these attributes and use `PGM_UTL8.TO_RAW()` to convert the PGM8.MQOD data type to mqodRaw before the MQOPEN().

  On output, the queue manager sets some of the PGM8.MQOD fields upon return. Use `PGM_UTL8.RAW_TO_MQOD()` to convert the mqodRaw back to PGM8.MQOD data type after the MQOPEN() in order to process the PGM8.MQOD fields.

  This input and output argument is PL/SQL RAW data type. For the details of PGM8.MQOD, refer to "PGM8.MQOD Type Definition" on page A-22.

- **options** specifies the kind of open. Refer to "MQOPEN Values" on page A-37.

  This input argument is PL/SQL datatype BINARY_INTEGER.

- **hobj** contains the handle of the queue after the queue is opened and becomes an input argument for subsequent PGM8 calls. The queue handle remains valid until one of the following conditions occurs:
  - the queue is closed by a call to MQCLOSE
  - the current transaction is made permanent by a COMMIT or ROLLBACK command
  - the Oracle user session is ended by a DISCONNECT command

  This output argument is PL/SQL datatype BINARY_INTEGER.

**Examples**

1. Using your own variable names when arguments are in the required order:

   ```sql
   MQOPEN (descript, open_options, handle);
   ```

2. Using your own variable names when arguments are in any order:

   ```sql
   MQOPEN (options=>open_options, hobj=>handle),
       mqod=>descript);
   ```
PGM8.MQOD Type Definition

PGM8.MQOD is used to define the object to open and we need to use PGM_UTL8.TO_RAW() to convert it to PL/SQL RAW data type and to use it as an argument of MQOPEN.

You can use the default values for the PGM8.MQOD fields or change the fields for your application’s requirements. For example, to change a field value:

```sql
mqod.field_name := field_value
mqodRaw := PGM_UTL8.TO_RAW(mqod);
```

where:

- `mqod` is the PGM8.MQOD data type and specifies the object to open.
- `field_name` is a field name of the PGM8.MQOD type definition. You can set as many fields as necessary. Refer to Table A-4 for field names and descriptions.
- `field_value` is the value to assign to `field_name`. You can specify a value or use a PGM_SUP constant to assign a value.
- `mqodRaw` is an input output argument for the MQOPEN procedure.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
<th>PL/SQL Datatype</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECTTYPE</td>
<td>specifies the object to open. Use a PGM_SUP constant to assign a value. Refer to &quot;OBJECTTYPE Field&quot; on page A-35.</td>
<td>BINARY_INTEGER</td>
<td>PGM_SUP.MQOT_Q(queue)</td>
</tr>
<tr>
<td>OBJECTNAME</td>
<td>specifies the local name of the object as defined by the queue manager.</td>
<td>CHAR(48)</td>
<td>NULL</td>
</tr>
</tbody>
</table>
**Table A–4  PGM8.MQOD Fields**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
<th>PL/SQL Datatype</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECTQMGRNAME</td>
<td>specifies the name of the queue manager for the object defined by OBJECTNAME. Leave OBJECTQMGRNAME set to null values because the gateway only supports the opening of objects at the connected queue.</td>
<td>CHAR(48)</td>
<td>NULL</td>
</tr>
<tr>
<td>DYNAMICQNAME</td>
<td>is ignored unless the OBJECTNAME field specifies the name of a model queue. When a model queue is involved, then this field specifies the name of the dynamic queue to be created at the queue manager to which the gateway is connected.</td>
<td>CHAR(48)</td>
<td>AMQ.*</td>
</tr>
<tr>
<td>ALTERNATEUSERID</td>
<td>If the <code>options</code> argument of MQOPEN is set to the value of PGM_SUP.MQOO_ALTERNATE_USER_AUTHORITY, then this field specifies the alternate user ID which the queue manager uses to check the authorization for the queue being opened.</td>
<td>CHAR(12)</td>
<td>NULL</td>
</tr>
</tbody>
</table>
MQPUT Procedure

MQPUT sends a message to a queue. The queue must already be open by a previous call to MQOPEN with its options argument set to the value of PGM_SUP.MQOO_OUTPUT.

MQPUT differs from MQI calls in the following ways:

1. The connection handle argument is omitted from MQPUT because the gateway automatically takes care of managing queue manager connections.

2. The MQI completion code is not included in the procedure’s argument list. When a gateway procedure fails because the corresponding MQI call failed, then an Oracle error message is returned to the caller.

3. The MQI reason code is not included in the procedure’s argument list. When the corresponding MQI call for a gateway procedure returns a reason code, then the reason code is included in the Oracle error message returned to the caller.

4. The msg length argument is not included in the procedure’s argument list because the Oracle integrating server and the gateway automatically keep track of the message data length.

**Definition**

MQPUT(hobj, mqmdRaw, mqpmoRaw, msg)

where:

- **hobj** contains the handle for the queue to send the message to. The handle is returned by a previous call to MQOPEN. This input argument is PL/SQL data type BINARY_INTEGER.
Examples

1. Using your own variable names when arguments are in the required order:

   MQPUT(handle, descript, options, message);

2. Using your own variable names when arguments are in any order:

   MQPUT(mqpmo=>options, msg=>message,
       mqmd=>descript);
3. The following sample, which is provided as a sample with the gateway (ORACLE_HOME/pg4mqseries/sample/putsample.sql), sends a message shorter than 32,767 bytes:

Example

-- Copyright Oracle Corporation, 2001 All Rights Reserved.
-- NAME
-- putsample.sql
-- DESCRIPTION
-- Specify the database link name you created for the gateway. To do this,
-- replace the characters @pgmq with @dblink, where dblink is the name
-- you chose when the database link was created.
--
-- This script performs a test run for the MQSeries gateway. In this
-- script the queuename is 'YOUR_QUEUE_NAME', replace it with a valid
-- queue name at the queue manager the gateway is configured for.
--
-- NOTES
-- Run the script from the SQL*Plus command line.
--
-- Make sure the script dblink.sql is executed to create the
-- database link to the gateway.
--
-- Make the sure the user is granted 'EXECUTE' on package dbms_output
--

SET SERVEROUTPUT ON

DECLARE
  objdesc    PGM8.MQOD;
  msgDesc    PGM8.MQMD;
  putOptions   PGM8.MQPMO;
  objectHandle binary_integer;
  message    raw(255);
  mqodRaw    PGM8.MQODRAW;
  mqmdRaw    PGM8.MQMDRAW;
  mqpmoRaw    PGM8.MQPMORAW;
  mqgmoRaw    PGM8.MQGMORAW;
BEGIN
  -- Open the queue 'YOUR_QUEUE_NAME' for sending.
MQPUT Procedure

The PGM8, PGM_UTL8, and PGM_SUP Packages

- Convert PGM8.MQMD to RAW
  
  mqodRaw := PGM_UTL8.TO_RAW(objdesc);
  MQOPEN@pg4mq(mqodRaw, PGM_SUP.MQOO_OUTPUT, objectHandle);

- Convert RAW to PGM8.MQMD
  
  objdesc := PGM_UTL8.RAW_TO_MQOD(mqodRaw);

- Put the message buffer on the queue.
  
  message := '00010203040506070809';
  -- Convert PGM8.MQMD and PGM8.MQGMO to RAW
  mqmdRaw := PGM_UTL8.TO_RAW(msgDesc);

- Convert RAW to PGM8.MQMD and PGM8.MQPMO
  
  mqpmoRaw := PGM_UTL8.TO_RAW(putOptions);
  MQPUT@pg4mq(objectHandle, mqmdRaw, mqpmoRaw, message);

- Print the message we are putting on the queue
  
  dbms_output.put_line('message put on queue = ' || rawtohex(message));

- Close the queue again.
  COMMIT;

-- something else went wrong.. tell the user.
WHEN OTHERS THEN
  DBMS_OUTPUT.PUT_LINE('Error: Procedural Gateway for IBM MQSeries verification script failed.');
  DBMS_OUTPUT.PUT_LINE(SQLERRM);
  MQCLOSE@pg4mq(objectHandle, PGM_SUP.MQCO_NONE);
END;
/

PGM8.MQPMO Type Definition

PGM8.MQPMO is used to define the mqpmo argument of MQPUT. It specifies option and control information for processing a message. An example of using PGM8.MQPMO is on page A-279.

You can use the default values for the PGM8.MQPMO fields or change the fields for the your application’s requirements. For example, to change a field value:

mqpmo.field_name := field_value
mqpmoRaw := PGM_UTL8.TO_RAW(mqpmo);
MQPUT Procedure

**mqpmo** is the PGM8.MQPMO data type and specifies option and control information about how the message is processed and put into a queue.

**field_name** is a field name of the PGM8.MQPMO type definition. You can set as many fields as necessary. Refer to Table A-5 for field names and descriptions.

**field_value** is the value to assign to **field_name**. You can specify a value or use a PGM_SUP constant to assign a value.

**mqpmoRaw** is an input output argument for the MQPUT procedure.

### Table A-5 PGM8.MQPMO Fields

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
<th>PL/SQL Datatype</th>
<th>Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTIONS</td>
<td>specifies options to control the MQPUT procedure. The field is set by adding one or more of the PGM_SUP definitions. Refer to &quot;OPTIONS Field&quot; on page A-36.</td>
<td>BINARY_INTEGER</td>
<td>PGM8.MQPMO_SYNCPOINT (Messages placed on the queue are coordinated by the Oracle transaction coordinator.)</td>
</tr>
<tr>
<td>CONTEXT</td>
<td>specifies the object handle of the input queue. It is only used when the OPTIONS field has the bit PGM_SUP.MQPMO_PASS_IDENTITY_CONTEXT or the bit PGM_SUP.MQPMO_PASS_ALL_CONTEXT set.</td>
<td>BINARY_INTEGER</td>
<td>Zero</td>
</tr>
<tr>
<td>RESOLVEDQNAME</td>
<td>contains the resolved name of the destination queue. This is an output field set by the queue manager upon return.</td>
<td>CHAR(48)</td>
<td>NULL</td>
</tr>
<tr>
<td>RESOLVEDQMGNAME</td>
<td>contains the resolved name of the queue manager for the queue name returned in the RESOLVEDQNAME field. This is an output field set by the queue manager upon return.</td>
<td>CHAR(48)</td>
<td>NULL</td>
</tr>
</tbody>
</table>
PGM_SUP Package

PGM_SUP contains constant and exception definitions to use with the gateway procedures and PGM8 type definitions. Using these values requires extensive knowledge of MQI and MQSeries programming in general. These definitions follow the MQI definition rules. For complete information about writing MQSeries applications, refer to the IBM MQSeries Application Programming Reference.

PGM8.MQGMO Values

<table>
<thead>
<tr>
<th>OPTIONS Field</th>
<th>Constant Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQGMO_NO_WAIT</td>
<td>constant binary_integer := 0;</td>
</tr>
<tr>
<td>MQGMO_NONE</td>
<td>constant binary_integer := 0;</td>
</tr>
<tr>
<td>MQGMO_WAIT</td>
<td>constant binary_integer := 1;</td>
</tr>
<tr>
<td>MQGMO_SYNCPOINT</td>
<td>constant binary_integer := 2;</td>
</tr>
<tr>
<td>MQGMO_NO_SYNCPOINT</td>
<td>constant binary_integer := 4;</td>
</tr>
<tr>
<td>MQGMO_SET_SIGNAL</td>
<td>constant binary_integer := 8;</td>
</tr>
<tr>
<td>MQGMO_BROWSE_FIRST</td>
<td>constant binary_integer := 16;</td>
</tr>
<tr>
<td>MQGMO_BROWSE_NEXT</td>
<td>constant binary_integer := 32;</td>
</tr>
<tr>
<td>MQGMO_ACCEPT_TRUNCATED_MSG</td>
<td>constant binary_integer := 64;</td>
</tr>
<tr>
<td>MQGMO_MARK_SKIP_BACKOUT</td>
<td>constant binary_integer := 128;</td>
</tr>
<tr>
<td>MQGMO_MSG_UNDER_CURSOR</td>
<td>constant binary_integer := 256;</td>
</tr>
<tr>
<td>MQGMO_LOCK</td>
<td>constant binary_integer := 512;</td>
</tr>
<tr>
<td>MQGMO_UNLOCK</td>
<td>constant binary_integer := 1024;</td>
</tr>
<tr>
<td>MQGMO_BROWSE_MSG_UNDER_CURSOR</td>
<td>constant binary_integer := 2048;</td>
</tr>
<tr>
<td>MQGMO_SYNCPOINT_IF_PERSISTENT</td>
<td>constant binary_integer := 4096;</td>
</tr>
<tr>
<td>MQGMO_FAIL_IFQUIESSCING</td>
<td>constant binary_integer := 8192;</td>
</tr>
<tr>
<td>MQGMO_CONVERT</td>
<td>constant binary_integer := 16384;</td>
</tr>
<tr>
<td>MQGMO_LOGICAL_ORDER</td>
<td>constant binary_integer := 32768;</td>
</tr>
<tr>
<td>MQGMO_COMPLETE_MSG</td>
<td>constant binary_integer := 65536;</td>
</tr>
<tr>
<td>MQGMO_ALL_MSGS_AVAILABLE</td>
<td>constant binary_integer := 131072;</td>
</tr>
<tr>
<td>MQGMO_ALL_SEGMENTS_AVAILABLE</td>
<td>constant binary_integer := 262144;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VERSION Field</th>
<th>Constant Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQGMO_VERSION_1</td>
<td>constant binary_integer := 1;</td>
</tr>
<tr>
<td>MQGMO_CURRENT_VERSION</td>
<td>constant binary_integer := 1;</td>
</tr>
<tr>
<td>MQGMO_VERSION_2</td>
<td>constant binary_integer := 2;</td>
</tr>
<tr>
<td>MQGMO_VERSION_3</td>
<td>constant binary_integer := 3;</td>
</tr>
</tbody>
</table>
MATCHOPTIONS Field

MQMO_DEFAULT            constant binary_integer := 3;
MQMO_NONE               constant binary_integer := 0;
MQMO_MATCH_MSG_ID        constant binary_integer := 1;
MQMO_MATCH_CORREL_ID     constant binary_integer := 2;
MQMO_MATCH_GROUP_ID      constant binary_integer := 4;
MQMO_MATCH_MSG_SEQ_NUMBER constant binary_integer := 8;
MQMO_MATCH_OFFSET        constant binary_integer := 16;
MQMO_MATCH_MSG_TOKEN     constant binary_integer := 32;

WAITINTERVAL

PGM_SUP.MQWI_UNLIMITED CONSTANT BINARY_INTEGER := -1;
PGM_SUP.MQWI_UNITS CONSTANT BINARY_INTEGER := 1000;

PGM8.MQMD Values

CODEDCHARSETID Field

PGM_SUP.MQCCSI_DEFAULT CONSTANT BINARY_INTEGER := 0;
PGM_SUP.MQCCSI_Q_MGR CONSTANT BINARY_INTEGER := 0;
PGM_SUP.MQCCSI_EMBEDDED CONSTANT BINARY_INTEGER := -1;

ENCODING Field

PGM_SUP.
MQENC_NATIVE
    CONSTANT RAW(4) := '00000111';

ENCODING Field, Values for Binary Integers

PGM_SUP.MQENC_INTEGER_UNDEFINED CONSTANT RAW(4) := '00000000';
PGM_SUP.MQENC_INTEGER_NORMAL CONSTANT RAW(4) := '00000001';
PGM_SUP.MQENC_INTEGER_REVERSED CONSTANT RAW(4) := '00000002';

ENCODING Field, Values for Floating Point Numbers

PGM_SUP.MQENC_FLOAT_UNDEFINED CONSTANT RAW(4) := '00000000';
PGM_SUP.MQENC_FLOAT_IEEE_NORMAL CONSTANT RAW(4) := '00000001';
PGM_SUP.MQENC_FLOAT_IEEE_REVERSED CONSTANT RAW(4) := '00000002';
PGM_SUP.MQENC_FLOAT_S390 CONSTANT RAW(4) := '00000003';
ENCODING Field, Mask Values

- PGM_SUP.MQENC_INTEGER_MASK  CONSTANT RAW(4) := '0000000f';
- PGM_SUP.MQENC_DECIMAL_MASK  CONSTANT RAW(4) := '000000f0';
- PGM_SUP.MQENC_FLOAT_MASK   CONSTANT RAW(4) := '00000f00';
- PGM_SUP.MQENC_RESERVED_MASK CONSTANT RAW(4) := 'fffffff00';

ENCODING Field, Values for Packed Decimal Integers

- PGM_SUP.MQENC_DECIMAL_UNDEFINED CONSTANT RAW(4) := '00000000';
- PGM_SUP.MQENC_DECIMAL_NORMAL   CONSTANT RAW(4) := '00000010';
- PGM_SUP.MQENC_DECIMAL_REVERSED  CONSTANT RAW(4) := '00000020';

EXPIRY Field

- PGM_SUP.MQEI_UNLIMITED   CONSTANT BINARY_INTEGER := -1;
- PGM_SUP.MQEI_MIN_EXPIRY  CONSTANT BINARY_INTEGER := 0;
- PGM_SUP.MQEI_UNITS     CONSTANT BINARY_INTEGER := 10;

FEEDBACK Field

- PGM_SUP.MQFB_NONE           CONSTANT BINARY_INTEGER := 0;
- PGM_SUP.MQFB_SYSTEM_FIRST       CONSTANT BINARY_INTEGER := 1;
- PGM_SUP.MQFB_EXPIRATION        CONSTANT BINARY_INTEGER := 258;
- PGM_SUP.MQFB_COA           CONSTANT BINARY_INTEGER := 259;
- PGM_SUP.MQFB_COD           CONSTANT BINARY_INTEGER := 260;
- PGM_SUP.MQFB_QUIT           CONSTANT BINARY_INTEGER := 256;
- PGM_SUP.MQFB_CHANNEL_COMPLETED    CONSTANT BINARY_INTEGER := 262;
- PGM_SUP.MQFB_CHANNEL_FAIL_RETRY    CONSTANT BINARY_INTEGER := 263;
- PGM_SUP.MQFB_CHANNEL_FAIL   CONSTANT BINARY_INTEGER := 264;
- PGM_SUP.MQFB_APPL_CANNOT_BE_STARTED CONSTANT BINARY_INTEGER := 265;
- PGM_SUP.MQFB_TM_ERROR         CONSTANT BINARY_INTEGER := 266;
- PGM_SUP.MQFB_APPL_TYPE_ERROR     CONSTANT BINARY_INTEGER := 267;
- PGM_SUP.MQFB_STOPPED_BY_MSG_EXIT   CONSTANT BINARY_INTEGER := 268;
- PGM_SUP.MQFB_XMIT_Q_MSG_ERROR     CONSTANT BINARY_INTEGER := 271;
- PGM_SUP.MQFB_SYSTEM_LAST       CONSTANT BINARY_INTEGER := 65535;
- PGM_SUP.MQFB_APPL_FIRST        CONSTANT BINARY_INTEGER := 65536;
- PGM_SUP.MQFB_APPL_LAST        CONSTANT BINARY_INTEGER := 999999999;
FORMAT Field

MQFMT_NONE         constant char(8) := '     ';  
MQFMT_ADMIN        constant char(8) := 'MQADMIN '; 
MQFMT_CHANNEL_COMPLETED constant char(8) := 'MQCHCOM '; 
MQFMT_CICS         constant char(8) := 'MQCICS '; 
MQFMT_COMMAND_1      constant char(8) := 'MQCMD1 '; 
MQFMT_COMMAND_2      constant char(8) := 'MQCMD2 '; 
MQFMT_DEAD_LETTER_HEADER constant char(8) := 'MQDEAD '; 
MQFMT_DIST_HEADER     constant char(8) := 'MQHDIST '; 
MQFMT_EVENT        constant char(8) := 'MQEVENT '; 
MQFMT_IMS         constant char(8) := 'MQIMS '; 
MQFMT_IMS_VAR_STRING    constant char(8) := 'MQIMSVS ';  
MQFMT_MD_EXTENTION     constant char(8) := 'MQHMDE '; 
MQFMT_PCF         constant char(8) := 'MQPCF '; 
MQFMT_REF_MSG_HEADER    constant char(8) := 'MQREF '; 
MQFMT_STRING        constant char(8) := 'MQSTR '; 
MQFMT_TRIGGER       constant char(8) := 'MQTRIG '; 
MQFMT_WORK_INFO_HEADER   constant char(8) := 'MQTRIG '; 
MQFMT_XMIT_Q_HEADER    constant char(8) := 'MQXMIT '; 

MSGTYPE Field

PGM_SUP.MQMT_SYSTEM_FIRST CONSTANT BINARY_INTEGER := 1;  
PGM_SUP.MQMT_REQUEST    CONSTANT BINARY_INTEGER := 1; 
PGM_SUP.MQMT_REPLY     CONSTANT BINARY_INTEGER := 2; 
PGM_SUP.MQMT_DATAGRAM    CONSTANT BINARY_INTEGER := 8; 
PGM_SUP.MQMT_REPORT     CONSTANT BINARY_INTEGER := 4; 
PGM_SUP.MQMT_SYSTEM_LAST  CONSTANT BINARY_INTEGER := 65535; 
PGM_SUP.MQMT_APPL_FIRST   CONSTANT BINARY_INTEGER := 65536; 
PGM_SUP.MQMT_APPL_LAST   CONSTANT BINARY_INTEGER := 999999999;

PERSISTENCE Field

PGM_SUP.MQPER_PERSISTENT       CONSTANT BINARY_INTEGER := 1;  
PGM_SUP.MQPER_NOT_PERSISTENT     CONSTANT BINARY_INTEGER := 0; 
PGM_SUP.MQPER_PERSISTENCE_AS_Q_DEF CONSTANT BINARY_INTEGER := 2;

PRIORITY Field

PGM_SUP.MQPRI_PRIORITY_AS_Q_DEF CONSTANT BINARY_INTEGER := -1;  
PGM_SUP.MQPRI_MIN_PRIORITY CONSTANT BINARY_INTEGER := 0; 
PGM_SUP.MQPRI_MAX_PRIORITY CONSTANT BINARY_INTEGER := 9;
PUTAPPLTYPE Field

MQAT_UNKNOWN constant binary_integer := -1;
MQAT_NO_CONTEXT constant binary_integer := 0;
MQAT_CICS constant binary_integer := 1;
MQAT_MVS constant binary_integer := 2;
MQAT_OS390 constant binary_integer := 2;
MQAT_IMS constant binary_integer := 3;
MQAT_OS2 constant binary_integer := 4;
MQAT_DOS constant binary_integer := 5;
MQAT_AIX constant binary_integer := 6;
MQAT_UNIX constant binary_integer := 6;
MQAT_QMGR constant binary_integer := 7;
MQAT_OS400 constant binary_integer := 8;
MQAT_WINDOWS constant binary_integer := 9;
MQAT_CICS_VSE constant binary_integer := 10;
MQAT_WINDOWS_NT constant binary_integer := 11;
MQAT_VMS constant binary_integer := 12;
MQAT_GUARDIAN constant binary_integer := 13;
MQAT_NSK constant binary_integer := 13;
MQAT_VOS constant binary_integer := 14;
MQAT_IMS_BRIDGE constant binary_integer := 19;
MQAT_XCF constant binary_integer := 20;
MQAT_CICS_BRIDGE constant binary_integer := 21;
MQAT_NOTES_AGENT constant binary_integer := 22;
MQAT_USER_FIRST constant binary_integer := 65536;
MQAT_USER_LAST constant binary_integer := 999999999;
MQAT_DEFAULT constant binary_integer := 6;
REPORT Field

MQRO_NEW_MSG_ID constant raw(4) := '00000000';
MQRO_COPY_MSG_ID_TO_CORREL_ID constant raw(4) := '00000000';
MQRO_DEAD_LETTER_Q constant raw(4) := '00000000';
MQRO_NONE constant raw(4) := '00000000';
MQRO_PAN constant raw(4) := '00000001';
MQRO_NAN constant raw(4) := '00000002';
MQRO_PASS_CORREL_ID constant raw(4) := '00000040';
MQRO_PASS_MSG_ID constant raw(4) := '00000080';
MQRO_COA constant raw(4) := '00000100';
MQRO_COA_WITH_DATA constant raw(4) := '00000300';
MQRO_COA_WITH_FULL_DATA constant raw(4) := '00000700';
MQRO_COD constant raw(4) := '00000800';
MQRO_COD_WITH_DATA constant raw(4) := '00001800';
MQRO_COD_WITH_FULL_DATA constant raw(4) := '00003800';
MQRO_EXPIRATION constant raw(4) := '00200000';
MQRO_EXPIRATION_WITH_DATA constant raw(4) := '00600000';
MQRO_EXPIRATION_WITH_FULL_DATA constant raw(4) := '00E00000';
MQRO_EXCEPTION constant raw(4) := '01000000';
MQRO_EXCEPTION_WITH_DATA constant raw(4) := '03000000';
MQRO_EXCEPTION_WITH_FULL_DATA constant raw(4) := '07000000';
MQRO_DISCARD_MSG constant raw(4) := '08000000';

VERSION Field

MQMD_VERSION_1 constant binary_integer := 1;
MQMD_VERSION_2 constant binary_integer := 2;
MQMD_CURRENT_VERSION constant binary_integer := 2;

Report Field, Mask Values

PGM_SUP.MQRO_REJECT_UNSUP_MASK CONSTANT RAW(4) := '101c0000';
PGM_SUP.MQRO_ACCEPT_UNSUP_MASK CONSTANT RAW(4) := 'efe000ff';
PGM_SUP.MQRO_ACCEPT_UNSUP_IF_XMIT_MASK CONSTANT RAW(4) := '0003ff00';
PGM8.MQOD Values

**OBJECTTYPE Field**

- `PGM_SUP.MQOT_Q` CONSTANT BINARY_INTEGER := 1;
- `PGM_SUP.MQOT_PROCESS` CONSTANT BINARY_INTEGER := 3;
- `PGM_SUP.MQOT_Q_MGR` CONSTANT BINARY_INTEGER := 5;
- `PGM_SUP.MQOT_CHANNEL` CONSTANT BINARY_INTEGER := 6;

**OBJECTTYPE Field, Extended Values**

- `MQOT_ALL` constant binary_integer := 1001;
- `MQOT_ALIAS_Q` constant binary_integer := 1002;
- `MQOT_MODEL_Q` constant binary_integer := 1003;
- `MQOT_LOCAL_Q` constant binary_integer := 1004;
- `MQOT_REMOTE_Q` constant binary_integer := 1005;
- `MQOT_SENDER_CHANNEL` constant binary_integer := 1007;
- `MQOT_SERVER_CHANNEL` constant binary_integer := 1008;
- `MQOT_REQUESTER_CHANNEL` constant binary_integer := 1009;
- `MQOT_RECEIVER_CHANNEL` constant binary_integer := 1010;
- `MQOT_CURRENT_CHANNEL` constant binary_integer := 1011;
- `MQOT_SAVED_CHANNEL` constant binary_integer := 1012;
- `MQOT_SVRCONN_CHANNEL` constant binary_integer := 1013;
- `MQOT_CLNTCONN_CHANNEL` constant binary_integer := 1014;

**VERSION Field**

- `MQOD_VERSION_1` constant binary_integer := 1;
- `MQOD_VERSION_2` constant binary_integer := 2;
- `MQOD_CURRENT_VERSION` constant binary_integer := 2;
PGM8.MQPMO Values

OPTIONS Field
MQPMO_NONE constant binary_integer := 0;
MQPMO_SYNCPOINT constant binary_integer := 2;
MQPMO_NO_SYNCPOINT constant binary_integer := 4;
MQPMO_DEFAULT_CONTEXT constant binary_integer := 32;
MQPMO_NEW_MSG_ID constant binary_integer := 64;
MQPMO_NEW_CORREL_ID constant binary_integer := 128;
MQPMO_PASS_IDENTITY_CONTEXT constant binary_integer := 256;
MQPMO_PASS_ALL_CONTEXT constant binary_integer := 512;
MQPMO_SET_IDENTITY_CONTEXT constant binary_integer := 1024;
MQPMO_SET_ALL_CONTEXT constant binary_integer := 2048;
MQPMO_ALTERNATE_USER_AUTHORITY constant binary_integer := 4096;
MQPMO_FAIL_IF_QUIESCING constant binary_integer := 8192;
MQPMO_NO_CONTEXT constant binary_integer := 16384;
MQPMO_LOGICAL_ORDER constant binary_integer := 32768;

VERSION Field
MQPMO_VERSION_1 constant binary_integer := 1;
MQPMO_VERSION_2 constant binary_integer := 2;
MQPMO_CURRENT_VERSION constant binary_integer := 2;

MQCLOSE Values

hobj Argument
PGM_SUP.MQHO_UNUSABLE_HOBJ CONSTANT BINARY_INTEGER := -1;

options Argument
PGM_SUP.MQCO_NONE CONSTANT BINARY_INTEGER := 0;
PGM_SUP.MQCO_DELETE CONSTANT BINARY_INTEGER := 1;
PGM_SUP.MQCO_DELETE_PURGE CONSTANT BINARY_INTEGER := 2;
MQOPEN Values

options Argument

- MQOO_BIND_AS_Q_DEF constant binary_integer := 0;
- MQOO_INPUT_AS_Q_DEF constant binary_integer := 1;
- MQOO_INPUT_SHARED constant binary_integer := 2;
- MQOO_INPUT_EXCLUSIVE constant binary_integer := 4;
- MQOO_BROWSE constant binary_integer := 8;
- MQOO_OUTPUT constant binary_integer := 16;
- MQOO_INQUIRE constant binary_integer := 32;
- MQOO_SET constant binary_integer := 64;
- MQOO_SAVE_ALL_CONTEXT constant binary_integer := 128;
- MQOO_PASS_ALL_CONTEXT constant binary_integer := 256;
- MQOO_SET_ALL_CONTEXT constant binary_integer := 512;
- MQOO_ALTERNATE_USER_AUTHORITY constant binary_integer := 1024;
- MQOO_FAIL_IF_QUIESCING constant binary_integer := 2048;
- MQOO_BIND_ON_OPEN constant binary_integer := 4096;
- MQOO_BIND_NOT_FIXED constant binary_integer := 8192;
- MQOO_PASS_IDENTITY_CONTEXT constant binary_integer := 16384;
- MQOO_PASS_ALL_CONTEXT constant binary_integer := 32768;
- MQOO_SET_IDENTITY_CONTEXT constant binary_integer := 65536;
- MQOO_SET_ALL_CONTEXT constant binary_integer := 131072;

Maximum Lengths for Fields of PGM8 Type Definitions

These constants contain the maximum lengths allowed for fields used by the PGM8 Type Definitions. For example, the constant PGM_SUP.MQ_ACCOUNTING_TOKEN_LENGTH specifies that the maximum length for PGM8.MQMD.ACCOUNTINGTOKEN is 32 characters.

- MQ_ABEND_CODE_LENGTH constant binary_integer := 4;
- MQ_ACCOUNTING_TOKEN_LENGTH constant binary_integer := 32;
- MQ_APPL_IDENTITY_DATA_LENGTH constant binary_integer := 32;
- MQ_APPL_ORIGIN_DATA_LENGTH constant binary_integer := 4;
- MQ_ATTENTION_ID_LENGTH constant binary_integer := 4;
- MQ_AUTHENTICATOR_LENGTH constant binary_integer := 8;
- MQ_CANCEL_CODE_LENGTH constant binary_integer := 4;
- MQ_CLUSTER_NAME_LENGTH constant binary_integer := 48;
- MQ_CORREL_ID_LENGTH constant binary_integer := 24;
- MQ_CREATION_DATE_LENGTH constant binary_integer := 12;
- MQ_CREATION_TIME_LENGTH constant binary_integer := 8;
- MQ_DATE_LENGTH constant binary_integer := 12;
- MQ_EXIT_NAME_LENGTH constant binary_integer := 128;
MQ_FACILITY_LENGTH  constant binary_integer := 8;
MQ_FACILITY_LIKE_LENGTH constant binary_integer := 4;
MQ_FORMAT_LENGTH     constant binary_integer := 8;
MQ_FUNCTION_LENGTH    constant binary_integer := 4;
MQ_GROUP_ID_LENGTH    constant binary_integer := 24;
MQ_LTERM_OVERRIDE_LENGTH constant binary_integer := 8;
MQ_MFS_MAP_NAME_LENGTH constant binary_integer := 8;
MQ_MSG_HEADER_LENGTH   constant binary_integer := 4000;
MQ_MSG_ID_LENGTH      constant binary_integer := 24;
MQ_MSG_TOKEN_LENGTH    constant binary_integer := 16;
MQ_NAMELIST_DESC_LENGTH constant binary_integer := 64;
MQ_NAMELIST_NAME_LENGTH constant binary_integer := 48;
MQ_OBJECT_INSTANCE_ID_LENGTH constant binary_integer := 24;
MQ_NAME_LENGTH        constant binary_integer := 48;
MQ_PROCESS_APPL_ID_LENGTH constant binary_integer := 256;
MQ_PROCESS_DESC_LENGTH constant binary_integer := 64;
MQ_PROCESS_ENV_DATA_LENGTH constant binary_integer := 128;
MQ_PROCESS_NAME_LENGTH constant binary_integer := 48;
MQ_PROCESS_USER_DATA_LENGTH constant binary_integer := 128;
MQ_PUT_APPL_NAME_LENGTH constant binary_integer := 28;
MQ_PUT_DATE_LENGTH    constant binary_integer := 8;
MQ_PUT_TIME_LENGTH    constant binary_integer := 8;
MQ_Q_DESC_LENGTH       constant binary_integer := 64;
MQ_Q_MGR_DESC_LENGTH   constant binary_integer := 64;
MQ_Q_MGR_IDENTIFIER_LENGTH constant binary_integer := 48;
MQ_Q_MGR_NAME_LENGTH   constant binary_integer := 48;
MQ_Q_NAME_LENGTH       constant binary_integer := 48;
MQ_REMOTE_SYS_ID_LENGTH constant binary_integer := 4;
MQ_SERVICE_NAME_LENGTH constant binary_integer := 32;
MQ_SERVICE_STEP_LENGTH constant binary_integer := 8;
MQ_START_CODE_LENGTH   constant binary_integer := 4;
MQ_STORAGE_CLASS_LENGTH constant binary_integer := 8;
MQ_TIME_LENGTH         constant binary_integer := 8;
MQ_TRAN_INSTANCE_ID_LENGTH constant binary_integer := 16;
MQ_TRANSACTION_ID_LENGTH constant binary_integer := 4;
MQ_TP_NAME_LENGTH      constant binary_integer := 64;
MQ_TRIGGER_DATA_LENGTH constant binary_integer := 64;
MQ_USER_ID_LENGTH      constant binary_integer := 12;
Error Code Definitions

Error Codes -8500 and -5801

Error Code -8500
The PGM8 package returns error code -8500 when an underlying MQI call fails. The PGM_SUP package defines an exception for this error code that a PL/SQL application can use:

MQI_CALL_FAILED EXCEPTION;
PRAGMA EXCEPTION_INIT(MQI_CALL_FAILED, -8500);

Error Code -8501
MQGET returns error code -8501 when no more messages are available in the queue. The PGM_SUP package defines an exception for this error code that a PL/SQL application can use:

NO_MORE_MESSAGES EXCEPTION;
PRAGMA EXCEPTION_INIT(NO_MORE_MESSAGES, -8501);

The following example shows a PL/SQL application using MQI_CALL_FAILED and NO_MORE_MESSAGES to catch and handle the -8500 and -8501 errors. For NO_MORE_MESSAGES, the application considers this error to be a warning and closes the opened queue. For the MQI_CALL_FAILED exceptions, the application prints an error message.
Example

-- Copyright Oracle Corporation, 1999, 2002 All Rights Reserved.
--
-- NAME
-- test.sql
--
-- DESCRIPTION
-- Specify the database link name you created for the gateway. To do this,
-- replace the characters @pg4mq with @dblink, where dblink is the name
-- you chose when the database link was created.
--
-- This script performs a test run for the MQSeries gateway. In this
-- script the queuename is 'YOUR_QUEUE_NAME', replace queuename with a valid
-- queue name at the queue manager the gateway is configured for.
--
-- First the script puts a raw message of 10 bytes on the specified queue.
--
-- When successfully completed the put operation, the script does a
-- get on the same queue to read the message back.
--
-- The contents of both messages put and retrieved from the queue are
-- printed to standard out for verification by the user.
--
-- NOTES
-- Run the script from the SQL*Plus command line.
--
-- Make sure the script dblink.sql is executed to create the
-- database link to the gateway.
--
-- Make the sure the user is granted 'EXECUTE' on package dbms_output
--

set serveroutput on

declare

  objdesc   PGM8.MQOD;
  options   binary_integer;
  hobj     binary_integer;
  msgdesc   PGM8.MQMD;
  putmsgopts PGM8.MQPMO;
  getmsgopts PGM8.MQGMO;
  putbuffer  raw(10) := '00102030405060708090';
  getbuffer  raw(10);
mqodRaw   PGM8.MQODRAW;
mqmdRaw   PGM8.MQMDRAW;
mqpmoRaw   PGM8.MQPMORAW;
mqmMsgoRaw   PGM8.MQGMORAW;
correlid   char(10) := '0123456789';

queueName   constant varchar2(48) := 'YOUR_QUEUE_NAME';

begin

-- Print the message we are putting on the queue
--

dbms_output.put_line('message put on queue = ' || rawtohex(putbuffer));

--
-- Specify queue name (replace with proper name).
--

objdesc.objectname := queueName;

--
-- Specify a put operation.
--

options := pgm_sup.MQOO_OUTPUT;

--
-- Open the queue.
--

mqodRaw := PGM_UTL8.TO_RAW(objdesc);
MQOPEN@pg4mq(mqodRaw, options, hobj);
objdesc := PGM_UTL8.RAW_TO_MQOD(mqodRaw);
dbms_output.put_line('MQOD.OBJECTNAME = ' || objdesc.OBJECTNAME);

--
-- Put the message buffer on the queue.
--

msgdesc.CORRELID := correlid;
mqmdRaw := PGM_UTL8.TO_RAW(msgdesc);
mqpmoRaw := PGM_UTL8.TO_RAW(putmsgopts);
MQPUT@pg4mq(hobj, mqmdRaw, mqpmoRaw, putbuffer);
msgdesc := PGM_UTL8.RAW_TO_MQMD(mqmdRaw);
putmsgopts := PGM_UTL8.RAW_TO_MQPMO(mqpmoRaw);

The PGM8, PGM_UTL8, and PGM_SUP Packages   A-41
dbms_output.put_line('MQMD.USERIDENTIFIER = ' || msgdesc.useridentifier);

-- Define close options.
--
-- Close queue.
--
-- MQCLOSE@pg4mq(hobj, options);
--
-- Specify a get operation.
--
-- options := pgm_sup.MQCO_NONE;
--
-- Open queue.
--
-- MQOPEN@pg4mq(mqodRaw, options, hobj);
objdesc := PGM_UTL8.RAW_TO_MQOD(mqodRaw);
--
-- Get message from the queue.
--
-- Test correlation ID --
--msgdesc.CORRELID := '1234567890';
--mqmdRaw := PGM_UTL8.UNSAFE_RAW(mqmdRaw);
getmsgopts.msglength := 10;
getmsgopts.matchoptions := PGM_SUP.MQMO_MATCH_CORREL_ID;
getmsgopts.matchoptions := PGM_SUP.MQMO_MATCH_MSG_ID;
mqgмоRaw := PGM_UTL8.UNSAFE_RAW(getmsgopts);
--mqGET@pg4mq(hobj, mqodRaw, mqgмоRaw, getbuffer);
msgdesc := PGM_UTL8.UNSAFE_RAW(mqmdRaw);
getmsgopts := PGM_UTL8.UNSAFE_RAW(mqgмоRaw);

dbms_output.put_line('MQGMO.MSGLENGTH = ' || getmsgopts.MSGLENGTH);

dbms_output.put_line('MQMD.USERIDENTIFIER = ' || msgdesc.USERIDENTIFIER);
--
-- Define close options.
--
-- options := pgm_sup.MQCO_NONE;
--
-- Close the queue again.
--
-- MQCLOSE@pg4mq(hobj, options);
--
-- Print the result
--
dbms_output.put_line('message read back = ' || rawtohex(getbuffer));

exception

-- When no more messages... tell the user and close the queue.
--
when pgm_sup.NO_MORE_MESSAGES then
  dbms_output.put_line('Warning: No message found on the queue');
  options := pgm_sup.MQCO_NONE;
  MQCLOSE@pg4mq(hobj, options);
--
  -- something else went wrong.. tell the user.
--
when others then
  dbms_output.put_line('Error: Procedural Gateway for IBM MQSeries verification script
    failed.');
  dbms_output.put_line(substr(1,100,SQLERRM));
end;
/

The PGM8, PGM_UTL8, and PGM_SUP Packages   A-43
Use the Visual Workbench when developing applications that access MQSeries through the gateway. The Visual Workbench defines an interface for accessing MQSeries and automatically generates the PL/SQL code (the MIP) for Oracle applications to interface with the gateway. Refer to the Oracle Visual Workbench for Oracle Procedural Gateways for IBM MQSeries Installation and User’s Guide for more information about Visual Workbench.

- Message Data Types on page B-2
- UTL_RAW Functions on page B-2
Message Data Types

Messages sent to an MQSeries queue or retrieved from an MQSeries queue are transferred as untyped data by the MIP procedures. When data profiles are defined in the MIP, the MIP converts message data from Oracle data types to target data types that the receiving application understands. The message data is packed into a buffer of data type RAW before being sent to the MQSeries queue. The same conversion process applies when receiving a message. The MIP unpacks the message from the buffer and converts it to specified Oracle data types.

The MIP uses the functions of the UTL_RAW package to perform the message data conversions. The UTL_RAW package is a PL/SQL package that contains procedures for converting and packing message data that is sent back and forth through the MQSeries queues using the RAW data type and PL/SQL data types.

When necessary, you can enhance the message data conversions in the generated MIP with the UTL_RAW functions. When no data profiles are defined in the MIP, you can create your own data conversion procedures with UTL_RAW functions, calling these functions before sending a message and immediately after receiving a message.

The UTL_RAW package is not included with the gateway. It is shipped with each Oracle server. Refer to your Oracle DBA about installing the UTL_RAW package.

UTL_RAW Functions

The UTL_RAW functions are called with the following syntax:

UTL_RAW.function(arg1, arg2, ...)

The function name, arguments, their Oracle data types, and the return value data type are provided with each function description in this appendix. For ease of description, the functions are described with PL/SQL syntax that shows the resulting function value placed in a variable:

result := UTL_RAW.function(arg1, arg2, ...);

However, the function can be used as a component in a PL/SQL expression as illustrated in the following example. This example takes two characters strings, "Hello " and "world!", converts them to raw message data with UTL_RAW.CAST_TO_RAW, concatenates them with UTL_RAW.CONCAT, and uses the gateway to send them to an MQSeries queue. The same message is retrieved from the queue, converted to a character data type with UTL_RAW.CAST_TOVARCHAR2, and printed.
Example

DECLARE

    objdesc   PGM8.MQOD;
    options   BINARY_INTEGER;
    hobj      BINARY_INTEGER;
    msgdesc   PGM8.MQMD;
    putmsgopts PGM8.MQPMO;
    getmsgopts PGM8.MQGMO;
    message   RAW(32767);
    mqodRaw   PGM8.MQODRAW;
    mqmdRaw   PGM8.MQMDRAW;
    mqpmoRaw  PGM8.MQPMORAW;
    mqgmoRaw  PGM8.MQGمورaw;

BEGIN

    -- Open QUEUE2.
    objdesc.OBJECTNAME := 'QUEUE2';
    options := PGM_SUP.MQOO_OUTPUT;
    mqodRaw := PGM_UTL.TO_RAW(objdesc);
    PGM8.MQOPEN@pg4mq(mqodRaw, options, hobj);

    -- Convert first part of message to raw data.
    message := UTL_RAW.CAST_TO_RAW('Hello ');

    -- Concatenate second part of message to first part
    message := UTL_RAW.CONCAT(message, UTL_RAW.CAST_TO_RAW('World !'));

    -- Send the message to QUEUE2.
    mqmdRaw := PGM_UTL.TO_RAW(msgdesc);
    mqpmoRaw := PGM_UTL.TO_RAW(putmsgopts);
    PGM8.MQPUT@pg4mq(hobj, mqmdRaw, mqpmoRaw, message);

    -- Close QUEUE2.
    Options := PGM_SUP.MQCO_NONE;
    PGM8.MQCLOSE@pg4mq(hobj, options);

    -- Reopen QUEUE2 for retrieving message.

    -- Open QUEUE2.
    objdesc.OBJECTNAME := 'QUEUE2';
    options := PGM_SUP.MQOO_OUTPUT;
    mqodRaw := PGM_UTL.TO_RAW(objdesc);
    PGM8.MQOPEN@pg4mq(mqodRaw, options, hobj);

    -- Convert first part of message to raw data.
    message := UTL_RAW.CAST_TO_RAW('Hello ');

    -- Concatenate second part of message to first part
    message := UTL_RAW.CONCAT(message, UTL_RAW.CAST_TO_RAW('World !'));

    -- Send the message to QUEUE2.
    mqmdRaw := PGM_UTL.TO_RAW(msgdesc);
    mqpmoRaw := PGM_UTL.TO_RAW(putmsgopts);
    PGM8.MQPUT@pg4mq(hobj, mqmdRaw, mqpmoRaw, message);

    -- Close QUEUE2.
    Options := PGM_SUP.MQCO_NONE;
    PGM8.MQCLOSE@pg4mq(hobj, options);

    -- Reopen QUEUE2 for retrieving message.
options := PGM_SUP.MQOO_INPUT_AS_Q_DEF;
PGM8.MQOPEN@pg4mq(mqodRaw, options, hobj);
-- Retrieve the message from QUEUE2.
mqgmoRaw := PGM_UTL.TO_RAW(getmsgopts);
PGM8.MQGET@pg4mq(hobj, mqmdRaw, mqgmoRaw, message);
-- Close QUEUE2.
Options := PGM_SUP.MQCO_NONE;
PGM8.MQCLOSE@pg4mq(hobj, options);

DBMS_OUTPUT.PUT_LINE('message = ' || UTL_RAW.CAST_TO_VARCHAR2(message));
END;
/

UTL_RAW.TO_RAW

PGM_UTL.TO_RAW converts values of data types PGM8.MQOD, PGM8.MQMD, PGM8.MQPMO and PGM8.MQGMO to into raw values.

Syntax

result := PGM_UTL.TO_RAW(input);

where:

result is the output value of the function. It is data type RAW.
input is the input value of data type PGM8.MQOD or PGM8.MQMD or PGM8.MQPMO or PGM8.MQGMO to convert to raw data.
UTL_RAW.BIT_AND

UTL_RAW.BIT_AND performs a bitwise logical AND operation on two raw values. If the values have different lengths, then the AND operation is terminated after the last byte of the shorter of the two values. The unprocessed portion of the longer value is appended to the partial result to produce the final result. The length of the resulting value equals the longer of the two input values.

Syntax

\[
\text{result} := \text{UTL_RAW.BIT_AND}(\text{input1}, \text{input2});
\]

where:

\begin{align*}
\text{result} & \quad \text{is the output value of the function. It is data type RAW. The value is null if input1 or input2 is null.} \\
\text{input1} & \quad \text{is an input value of datatype RAW to AND with input2.} \\
\text{input2} & \quad \text{is an input value of datatype RAW to AND with input1.}
\end{align*}

UTL_RAW.BIT_COMPLEMENT

UTL_RAW.BIT_COMPLEMENT performs a bitwise logical COMPLEMENT operation of a raw value. The length of the resulting value equals the length of the input value.

Syntax

\[
\text{result} := \text{UTL_RAW.BIT_COMPLEMENT}(\text{input});
\]

where:

\begin{align*}
\text{result} & \quad \text{is the output value of the function. It is data type RAW. The value is null if input is null.} \\
\text{input} & \quad \text{is an input value of datatype RAW on which to perform the COMPLEMENT operation.}
\end{align*}
UTL_RAW Functions

UTL_RAW.BIT_OR
UTL_RAW.BIT_OR performs a bitwise logical OR operation of two raw values. If the values have different lengths, then the OR operation is terminated after the last byte of the shorter of the two values. The unprocessed portion of the longer value is appended to the partial result to produce the final result. The length of the resulting value equals the longer of the two input values.

Syntax
result := UTL_RAW.BIT_OR(input1, input2);

where:

result is the output value of the function. It is data type RAW. The value is null if input1 or input2 is null.

input1 is an input value of datatype RAW to OR with input2.

input2 is an input value of datatype RAW to OR with input1.

UTL_RAW.BIT_XOR
UTL_RAW.BIT_XOR performs a bitwise logical EXCLUSIVE OR operation of two raw values. If the values have different lengths, then the EXCLUSIVE OR operation is terminated after the last byte of the shorter of the two values. The unprocessed portion of the longer value is appended to the partial result to produce the final result. The length of the resulting value equals the longer of the two input values.

Syntax
result := UTL_RAW.BIT_XOR(input1, input2);

where:

result is the output value of the function. It is data type RAW. The value is null if input1 or input2 is null.

input1 is an input value of datatype RAW to EXCLUSIVE OR with input2.

input2 is an input value of datatype RAW to EXCLUSIVE OR with input1.
UTL_RAW.CAST_TO_RAW

UTL_RAW.CAST_TO_RAW converts a value of data type VARCHAR2 into a raw value with the same number of bytes. The input value is treated as if it were composed of single 8-bit bytes, not characters. Multibyte character boundaries are ignored. The data is not modified in any way, it is only changed to data type RAW.

Syntax

\[ \text{result} := \text{UTL_RAW.CAST_TO_RAW}(\text{input}); \]

where:

result is the output value of the function. It is data type RAW. The value is null if input is null.

input is the input value of datatype VARCHAR2 to convert to raw data.

UTL_RAW.CAST_TO_VARCHAR2

UTL_RAW.CAST_TO_VARCHAR2 converts a raw value into a value of data type VARCHAR2 with the same number of data bytes. The result is treated as if it were composed of single 8-bit bytes, not characters. Multibyte character boundaries are ignored. The data is not modified in any way, it is only changed to data type VARCHAR2.

Syntax

\[ \text{result} := \text{UTL_RAW.CAST_TO_VARCHAR2}(\text{input}); \]

where:

result is the output value of the function. It is data type VARCHAR2. The value is null if input is null.

input is the input value of datatype RAW to convert to datatype VARCHAR2.
UTL_RAW Functions

UTL_RAW.COMpare

UTL_RAW.COMpare compares one raw value to another raw value. If they are identical, then UTL_RAW.COMpare returns zero. If they are not identical, then COMPARE returns the position of the first byte that does not match. If the input values have different lengths, then the shorter input value is padded on the right by a value that you specify.

Syntax

result := UTL_RAW.COMpare(input1, input2[, pad]);

where:

result is the output value of the function. It is data type NUMBER. A value of zero is returned if the values of input1 and input2 are null or identical or the position, numbered from 1, of the first mismatched byte.

input1 is the first input value of datatype RAW to compare.

input2 is the second input value of datatype RAW to compare.

pad is a one-byte value used to pad the shorter input value. The default is X'00'.

UTL_RAW.ConvCat

UTL_RAW.CONCAT concatenates a set of up to 12 raw values into a single raw value. The values are appended together, left to right, in the order that they appear in the parameter list. Null input values are skipped, and the concatenation continues with the next non-null value.

If the sum of the lengths of the input values exceeds 32,767 bytes, then a VALUE_ERROR exception is raised.

Syntax

result := UTL_RAW.CONCAT(input1, ... input12);

where:

result is the output value of the function. It is data type RAW.

input1 ... input12 are the input values of datatype RAW to concatenate.
UTL_RAW Functions

UTL_RAW.CONVERT

UTL_RAW.CONVERT converts a raw value to a different character set. A VALUE_ERROR exception is raised for any of the following conditions:

- the input value is null or zero in length
- one or both of the specified character sets is missing, null, or zero in length
- the character set names are invalid or unsupported by the Oracle server

Syntax

\[
\text{result} := \text{UTL_RAW.CONVERT}(\text{input}, \text{new charset}, \text{old charset});
\]

where:

- \text{result} is the output value of the function. It is data type RAW.
- \text{input} is the input value of datatype RAW to convert.
- \text{new charset} is the national language support (NLS) character set to convert \text{input} to.
- \text{old charset} is the NLS character set that \text{input} is currently using.

UTL_RAW.COPIES

UTL_RAW.COPIES returns one or more copies of a value. The values are concatenated together. A VALUE_ERROR exception is raised for any of the following conditions:

- the input value is null or has a length of zero
- a negative number of copies is specified
- the length of the result exceeds 32 767 bytes

Syntax

\[
\text{result} := \text{UTL_RAW.COPIES}(\text{input}, \text{number});
\]

where:

- \text{result} is the output value of the function. It is data type RAW.
- \text{input} is a value of datatype RAW to copy.
- \text{number} is the number of times to copy \text{input}. It must be a positive value.
**UTL_RAW.LENGTH**

UTL_RAW.LENGTH returns the length, in bytes, of a raw value.

**Syntax**

\[
\text{result} \ := \ \text{UTL_RAW.LENGTH}(\text{input});
\]

where:

- \text{result}  is the output value of the function. It is data type NUMBER.
- \text{input}    is the input value of datatype RAW to evaluate.

**UTL_RAW.OVERLAY**

UTL_RAW.OVERLAY replaces a portion of a raw value with a new string of raw data. If the new data is shorter than the length of the overlay area, then the new data is padded to make it long enough. If the new data is longer than the overlay area, then the extra bytes are ignored. If you specify an overlay area that exceeds the length of the input value, then the input value is extended according to the length specified. If you specify a starting position for the overlay area that exceeds the length of the input value, then the input value is padded to the position specified, and then the input value is further extended with the new data.

A VALUE_ERROR exception is raised for any of the following conditions:

- the new data used to overlay the input value is null or has a length of zero
- the portion of the input value to overlay is not defined
- the length of the portion to overlay exceeds 32 767 bytes
- the number of bytes to overlay is defined as less than zero
- the position within the input value to begin the overlay operation is defined as less than 1
**Syntax**

\[ \text{result} \ := \text{UTL_RAW.OVERLAY}(\text{new\_bytes}, \text{input}, \text{position}, \text{length}, \text{pad}) ; \]

where:

- \text{result} is the output value of the function. It is data type RAW.
- \text{new\_bytes} is the new value, a byte string of data type RAW, to overlay input with. Bytes are selected from \text{new\_bytes} beginning with the leftmost byte.
- \text{input} is the input value of data type RAW to overlay.
- \text{position} is the position within input, numbered from 1, at which to begin overlaying. This value must be greater than zero. The default is 1.
- \text{length} is the number of bytes to overlay. This must be greater than, or equal to, zero. The default is the length of \text{new\_bytes}.
- \text{pad} is a one-byte value used to pad when length exceeds the overlay length or when position exceeds the length of \text{input}. The default is X’00’.

**UTL_RAW.REVERSE**

UTL_RAW.REVERSE inverts the byte sequence of a raw value from end-to-end. For example, this function reverses X’0102F3’ to X’F30201’ or xyz to zyx. The length of the resulting value is the same length as the input value. A VALUE_ERROR exception is raised if the input value is null or has a length of zero.

**Syntax**

\[ \text{result} \ := \text{UTL_RAW.REVERSE}(\text{input}) ; \]

where:

- \text{result} is the output value of the function. It is data type RAW.
- \text{input} is the input value of data type RAW to be reversed.
UTL_RAW Functions

**UTL_RAW.SUBSTR**

UTL_RAW.SUBSTR removes bytes from a raw value. If you specify a positive number as the starting point for the bytes to remove, then SUBSTR counts from the beginning of the input value to find the first byte. If you specify a negative number, then UTL_RAW.SUBSTR counts backwards from the end of the input value to find the first byte.

A VALUE_ERROR exception is raised for any of the following conditions:

- the position to begin the removal is specified as zero
- the number of bytes to remove is specified as less than zero

**Syntax**

```
result := UTL_RAW.SUBSTR(input, position[,length]);
```

where:

- **result** is the output value of the function. It is data type RAW. The value is the specified byte or bytes from *input*, or the value is a null value if *input* is null.
- **input** is the input value of datatype RAW from which to extract a portion of its bytes.
- **position** is the byte position from which to start extraction. This value cannot be zero. If *position* is negative, then SUBSTR counts backwards from the end of input.
- **length** is the number of bytes after *position* to extract from *input*. This value must be greater than zero. When not specified, all bytes to the end of input are returned.

**UTL_RAW.TRANSLATE**

UTL_RAW.TRANSLATE changes the value of some of the bytes in a raw value according to a scheme that you specify. Bytes in the input value are compared to a matching string, and (when found to match) the byte at the same position in the replacement string is copied to the result, or it is omitted from the result if the offset exceeds the length of the replacement string. Bytes in the input value that do not appear in the matching string are copied to the resulting value. Only the leftmost occurrence of a byte in the matching string is used, and subsequent duplicate occurrences are ignored.
If the matching string contains more bytes than the replacement string, then the extra bytes at the end of the matching string have no corresponding bytes in the replacing string. Any bytes in the input value that match such bytes are omitted from the resulting value.

A VALUE_ERROR exception is raised for any of the following conditions:
- the input value is null or has a length of zero
- the matching string is null or has a length of zero
- the replacement string is null or has a length of zero

UTL_RAW.TRANSLATE differs from the UTL_RAW.TRANSLITERATE function in the following ways:
- the raw values used for the matching and replacement strings have no default values
- bytes in the input value that are undefined in the replacement string are omitted in the resulting value
- the resulting value can be shorter than the input value

**Syntax**

```
result := UTL_RAW.TRANSLATE(input, match, replace_bytes);
```

where:

- `result` is the output value of the function. It is data type RAW.
- `input` is the input value of datatype RAW to change.
- `match` specifies the byte-codes to search for in `input` and to change to `replace_bytes`. It is data type RAW.
- `replace_bytes` specifies the byte-codes that replace the codes specified by `match`. It is data type RAW.
UTL_RAW.TRANSLITERATE

UTL_RAW.TRANSLITERATE replaces all occurrences of any bytes in a matching string with the corresponding bytes of a replacement string. Bytes in the input value are compared to the matching string, and if they are not found, then they are copied unaltered to the resulting value. If they are found, then they are replaced in the resulting value by the byte at the same offset in the replacement string, or with the pad byte that you specify when the offset exceeds the length of the replacement string. Only the leftmost occurrence of a byte in the matching string is used. Subsequent duplicate occurrences are ignored. The result value of UTL_RAW.TRANSLITERATE is always the same length as the input value.

If the replacement string is shorter than the matching string, then the pad byte is placed in the resulting value when a selected matching string byte has no corresponding byte in the replacement string. A VALUE_ERROR exception is raised when the input value is null or has a length of zero.

UTL_RAW.TRANSLITERATE differs from UTL_RAW.TRANSLATE in the following ways:

■ bytes in the input value that are undefined in the replacement string are padded with a value that you specify

■ the resulting value is always the same length as the input value
**UTL_RAW Functions**

**Syntax**

\[
\text{result} := \text{UTL_RAW.TRANSLITERATE (input, replace\_bytes, match, pad)};
\]

where:

- **result** is the output value of the function. It is data type RAW.
- **input** is the input value of datatype RAW to change.
- **replace\_bytes** specifies the byte-codes to which corresponding bytes of match are changed. This value can be any length that is valid for the RAW data type. The default is a null value and effectively extends with pad to the length of match as necessary.
- **match** specifies the byte-codes to match in input. The value can be any length that is valid for the RAW data type. The default is X'00' through X'FF'.
- **pad** is a one-byte value that is used to extend the length of replace\_bytes when replace\_bytes is shorter than match. The default is X'00'.

**UTL_RAW.XRANGE**

UTL_RAW.XRANGE returns a raw value containing all valid one-byte codes within a range that you specify. If the starting byte value is greater than the ending byte value, then the succession of resulting bytes begin with the starting byte, wrapping from X'FF' to X'00', and end at the ending byte.

When specified, the values for the starting and ending bytes must be single-byte raw values.

**Syntax**

\[
\text{result} := \text{UTL_RAW.XRANGE(start, end)};
\]

where:

- **result** is the output value of the function. It is data type RAW.
- **start** is the one-byte code to start with. The default is X'00'.
- **end** is the one-byte code to end with. The default is X'FF'.
The gateway has its own initialization parameters (described in this appendix) and supports the initialization parameters for procedural gateways.

The parameters in this appendix are:

- **HS_DB_DOMAIN** on page C-2
- **HS_DB_NAME** on page C-2
- **LOG_DESTINATION** on page C-3
- **AUTHORIZATION_MODEL** on page C-3
- **QUEUE_MANAGER** on page C-4
- **TRACE_LEVEL** on page C-5
- **TRANSACTION_LOG_QUEUE** on page C-6
- **TRANSACTION_MODEL** on page C-7
- **TRANSACTION_RECOVERY_PASSWORD** on page C-8
- **TRANSACTION_RECOVERY_USER** on page C-10
Gateway Initialization File

The Gateway Initialization File is named `init{sid}.ora`. A default initialization file is created in the directory `ORACLE_HOME/pg4mqseries/admin` during the installation of the gateway software.

Gateway Parameters

Each gateway parameter listing includes the default value, range of values, and syntax for usage.

**HS_DB_DOMAIN**

Default value: None  
Range of values: None  
Syntax: `SET HS_DB_DOMAIN = domain_name`

`HS_DB_DOMAIN` is used only when the Oracle integrating server initialization parameter `GLOBAL_NAMES` is set to `TRUE`. In this case, the value of `domain_name` is the domain name of the database link created in the Oracle integrating server to connect to the queue manager.

**HS_DB_NAME**

Default value: None  
Range of values: None  
Syntax: `SET HS_DB_NAME = database_name`

`HS_DB_NAME` is used only when the Oracle integrating server initialization parameter `GLOBAL_NAMES` is set to `TRUE`. In this case, the value of `database_name` is the database name of the database link created in the Oracle integrating server to connect to the queue manager.
**LOG_DESTINATION**

Default value: \( \text{pg4mq} \cdot \text{PID} \cdot \text{log}\) (PID is the process ID of the gateway)

Range of values: None

Syntax: \( \text{SET LOG_DESTINATION} = \text{log\_file} \)

LOG_DESTINATION specifies the full path name of the gateway log file.

**AUTHORIZATION_MODEL**

Default value: RELAXED

Range of values: RELAXED or STRICT

Syntax: \( \text{SET AUTHORIZATION\_MODEL} = \{\text{RELAXED}|\text{STRICT}\} \)

AUTHORIZATION_MODEL defines the authorization model for the gateway user. Specify AUTHORIZATION_MODEL as:

- RELAXED specifies that authorizations that are granted to the effective user ID of the gateway by the queue manager are the only associations that an Oracle application has.

- STRICT specifies that the Oracle user ID and password (that are provided when a database link is created), or the current user ID and password (when the Oracle user ID and password are not provided), is checked against the local or network password file.

Refer to "Security Models" on page 8-2 for more information about effective user IDs.
**QUEUE_MANAGER**

Default value: None
Range of values: None
Syntax: \texttt{SET \textit{QUEUE_MANAGER} = \textit{manager\_name}}

QUEUE_MANAGER, a required parameter, specifies the name of the queue manager that the gateway connects to at logon time. The effective user ID of the gateway should have the correct user privileges and/or should be authorized to connect to this queue manager. Specify \textit{manager\_name} using the following rules:

- 1 to 48 alphanumeric characters in length
- no leading or embedded blank characters
- trailing blank characters are permitted

Refer to "Security Models" on page 8-2 for more information about effective user IDs.
TRACE_LEVEL

Default value: 0
Range of values: 0 to 7
Syntax: SET TRACE_LEVEL = level

TRACE_LEVEL controls whether tracing information is collected as the gateway runs. When set to collect information, the trace data is written to the log file that is specified by the LOG_DESTINATION parameter. Specify level as an integer from 0 to 7, which is the sum of the desired trace values:

0 specifies that no tracing is done.
1 specifies that general tracing is done. This includes the user ID that is used to logon to the MQSeries queue manager, the name of the queue manager, the gateway transaction mode, security mode, and so on.
2 specifies that tracing is done for all MQI calls that are issued by the gateway.
3 specifies that tracing is a combination of level 1 and level 2.
4 specifies that tracing is done for all parameter values that are passed to, or received from, the MQI calls that were issued by the gateway.
5 specifies that tracing is a combination of level 1 and level 4.
6 specifies that tracing is a combination of trace level 2 and level 4.
7 specifies that tracing is a combination of trace level 1, level 2, and level 4.

For more information about MQI calls, refer to IBM publications.
TRANSACTION_LOG_QUEUE

Default value: None
Range of values: None
Syntax: \texttt{SET TRANSACTION\_LOG\_QUEUE = tx\_queue\_name}

TRANSACTION\_LOG\_QUEUE specifies the name of the queue for logging transaction IDs. Specify \texttt{tx\_queue\_name} using the following rules:

- 1 to 48 alphanumeric characters in length
- no leading or embedded blank characters
- trailing blank characters are permitted

Refer to "Creating a Transaction Log Queue" on page 7-19 for more information.
**TRANSACTION_MODEL**

Default value: SINGLE_SITE  
Range of values: COMMIT_CONFIRM or SINGLE_SITE  
Syntax:  
\[
\text{SET \text{TRANSACTION\_MODEL} =}
\begin{cases}
\text{COMMIT\_CONFIRM} & | \\
\text{SINGLE\_SITE} & |
\end{cases}
\]

**TRANSACTION_MODEL** defines the transaction mode of the gateway. Specify **TRANSACTION\_MODEL** as:

**COMMIT\_CONFIRM** specifies that the gateway can participate in transactions when queues belonging to the same MQSeries queue manager are updated. At the same time, any number of Oracle databases are updated. Only one gateway with the commit-confirm model can join the distributed transaction, because the gateway operates as the focal point of the transaction.

When this value is specified, you must also set the RECOVERY\_USER, RECOVERY\_PASSWORD, and TRANSACTION\_LOG\_QUEUE parameters.

**SINGLE\_SITE** specifies that the gateway can participate in a transaction only when queues belonging to the same MQSeries queue manager are updated. An Oracle application can select, but not update, data at any Oracle database within the same transaction that accesses MQSeries.
TRANSACTION_RECOVERY_PASSWORD

Default value:  * 

Range of values:  An asterisk (*), which indicates that the parameter must be encrypted, or any valid password

Syntax:  SET TRANSACTION_RECOVERY_PASSWORD = rec_password 

or

SET TRANSACTION_RECOVERY_PASSWORD = *

TRANSACTION_RECOVERY_PASSWORD specifies the password of the user that the gateway uses to start recovery of a transaction. The default value is set to an asterisk (*), and this asterisk designates that the value of this parameter is stored in an encrypted form in a separate password file. To specify or change a valid password for encrypted gateway parameters, you need to use the gateway utility pg4mqpwd to do the work. For more information, refer to "Using the pg4mqpwd Utility" in the following text.

This TRANSACTION_RECOVERY_PASSWORD parameter is required only when TRANSACTION_MODEL is set to COMMIT_CONFIRM. Refer to "Creating a Transaction Log Queue" on page 7-19 for more information.

Passwords in the Gateway Initialization File

The gateway uses userids and passwords to access the information in the remote database on the IBM MQSeries server. Some userids and passwords must be defined in the Gateway Initialization File to handle functions such as resource recovery. In a security conscious environment, plain-text passwords are deemed insecure when they are accessible in the Initialization File. A new encryption feature has been added to the gateway to help make such passwords more secure. The pg4mqpwd utility can be used to encrypt passwords that would normally be stored in the Gateway Initialization File. Using this feature is optional, but highly recommended by Oracle Corporation. With this feature, passwords are no longer stored in the Initialization File but are stored instead in a password file in an encrypted form, thus making the information more secure. Read the next section to learn how to use this feature.
Using the pg4mqpwd Utility

The pg4mqpwd utility is used to encrypt passwords that would normally be stored in the Gateway Initialization File. The utility works by reading the Initialization File and looking for parameters with a special value. That value is the asterisk (*). This asterisk designates that the value of the parameter is stored in an encrypted form in another file. Following is a sample section of the Initialization File with this value:

TRANSACTION_RECOVERY_PASSWORD=*  

The Initialization File is first edited to set the value of the parameter to "*". Then the pg4mqpwd utility is run, specifying the gateway SID on the command line. The utility will read the Initialization File and prompt the user to enter the values to be encrypted.

The syntax of the command is:  

```
pg4mqpwd [gateway_sid]
```

Where [gateway_sid] is the SID of the gateway.

Following is an example run, assuming a gateway SID of pg4mq92:

```
% pg4mqpwd pg4mq92
ORACLE Gateway Password Utility (pg4mqseries) Constructing password file for Gateway SID pg4mq92
Enter the value for TRANSACTION_RECOVERY_PASSWORD
welcome
%
```

In this example, the parameter TRANSACTION_RECOVERY_PASSWORD has been identified as requiring encryption. The user enters the value (for example, welcome) and presses enter. If more parameters require encryption, then you will be prompted for their values. The encrypted data is stored in the pg4mqseries/admin directory.

**Note:** It is important that the ORACLE_HOME environmental variable be pointing to the correct gateway home in order to ensure that the correct Gateway Initialization File is read.
 TRANSACTION_RECOVERY_USER

Default value: None
Range of values: Any valid operating system user ID that is authorized by MQSeries Manager (Man)
Syntax: SET TRANSACTION_RECOVERY_USER = rec_user

TRANSACTION_RECOVERY_USER specifies the username that the gateway uses to start recovery of a transaction. This parameter is required only when AUTHORIZATION_MODEL is set to STRICT, and TRANSACTION_MODEL is set to COMMIT_CONFIRM. Refer to "Creating a Transaction Log Queue" on page 7-19 for more information.
The following tables describe the views for heterogeneous outbound and inbound propagation schedules.

The DBA views described in the appendix are:

- **DBA_OUTBOUND_SCHEDULES**: on page D-2
- **DBA_INBOUND_SCHEDULES**: on page D-4
### DBA_OUTBOUND_SCHEDULES:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Null?</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA</td>
<td>NOT NULL</td>
<td>VARCHAR2(30)</td>
<td>Schema for AQ source queue</td>
</tr>
<tr>
<td>QNAME</td>
<td>NOT NULL</td>
<td>VARCHAR2(30)</td>
<td>AQ source queue name</td>
</tr>
<tr>
<td>DESTINATION</td>
<td>NOT NULL</td>
<td>VARCHAR2(128)</td>
<td>Database link to a gateway for access to the destination queues at the non-Oracle message queuing system.</td>
</tr>
<tr>
<td>PROTOCOL</td>
<td></td>
<td>NUMBER</td>
<td>Propagation protocol number</td>
</tr>
<tr>
<td>TRANSMIT_QUEUE</td>
<td>NOT NULL</td>
<td>VARCHAR2(30)</td>
<td>Transmit-queue for remote propagation</td>
</tr>
<tr>
<td>EXCEPTION_QUEUE</td>
<td>NOT NULL</td>
<td>VARCHAR2(30)</td>
<td>Propagation-exception queue</td>
</tr>
<tr>
<td>START_DATE</td>
<td></td>
<td>DATE</td>
<td>Date to start propagation</td>
</tr>
<tr>
<td>START_TIME</td>
<td></td>
<td>VARCHAR2(8)</td>
<td>Time to start propagation</td>
</tr>
<tr>
<td>PROPAGATION_WINDOW</td>
<td></td>
<td>NUMBER</td>
<td>Duration in seconds for the propagation windows</td>
</tr>
<tr>
<td>NEXT_TIME</td>
<td></td>
<td>VARCHAR2(128)</td>
<td>Function to compute the start of next propagation window</td>
</tr>
<tr>
<td>LATENCY</td>
<td></td>
<td>NUMBER</td>
<td>Maximum wait time to propagate a message</td>
</tr>
<tr>
<td>LAST_DATE</td>
<td></td>
<td>DATE</td>
<td>Most recent date this schedule executed successfully</td>
</tr>
<tr>
<td>LAST_TIME</td>
<td></td>
<td>VARCHAR2</td>
<td>Time of day of the last successful execution of this schedule</td>
</tr>
<tr>
<td>FAILURES</td>
<td></td>
<td>NUMBER</td>
<td>Number of times the execution failed</td>
</tr>
<tr>
<td>DISABLED</td>
<td></td>
<td>VARCHAR2</td>
<td>N if enabled, Y if disabled and schedule will not be executed</td>
</tr>
<tr>
<td>ERROR_TIME</td>
<td></td>
<td>DATE</td>
<td>Date and time of last unsuccessful execution</td>
</tr>
<tr>
<td>ERROR</td>
<td></td>
<td>VARCHAR2</td>
<td>Oracle error number</td>
</tr>
<tr>
<td>ERROR_MSG</td>
<td></td>
<td>VARCHAR2</td>
<td>Error message text</td>
</tr>
<tr>
<td>Column Name</td>
<td>Null?</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------</td>
<td>-------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>THIS_DATE</td>
<td></td>
<td>DATE</td>
<td>Date the current execution of this schedule started. Null if not currently executing.</td>
</tr>
<tr>
<td>THIS_TIME</td>
<td></td>
<td>VARCHAR2</td>
<td>Time of day the current execution of this schedule started. Null if not currently executing.</td>
</tr>
<tr>
<td>SID</td>
<td></td>
<td>VARCHAR2</td>
<td>Session ID of the job executing this schedule. Null if not currently executing.</td>
</tr>
<tr>
<td>TOTAL_TIME</td>
<td></td>
<td>NUMBER</td>
<td>Total time, in seconds, spent by the system in executing this schedule.</td>
</tr>
<tr>
<td>AVG_NUM</td>
<td></td>
<td>NUMBER</td>
<td>Average number of messages propagated in a propagation window.</td>
</tr>
<tr>
<td>MAX_NUM</td>
<td></td>
<td>NUMBER</td>
<td>Maximum number of messages propagated in a propagation window.</td>
</tr>
<tr>
<td>AVG_SIZE</td>
<td></td>
<td>NUMBER</td>
<td>Average size of a propagated message in bytes.</td>
</tr>
<tr>
<td>MAX_SIZE</td>
<td></td>
<td>NUMBER</td>
<td>Maximum size of a propagated message in bytes.</td>
</tr>
<tr>
<td>AVG_TIME</td>
<td></td>
<td>NUMBER</td>
<td>Average time, in seconds, to propagate a message.</td>
</tr>
</tbody>
</table>
### DBA_INBOUND_SCHEDULES:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Null?</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCHEMA</td>
<td>NOT NULL</td>
<td>VARCHAR2(30)</td>
<td>Schema for AQ source queue</td>
</tr>
<tr>
<td>QNAME</td>
<td>NOT NULL</td>
<td>VARCHAR2(30)</td>
<td>AQ source queue name</td>
</tr>
<tr>
<td>SOURCE</td>
<td>NOT NULL</td>
<td>VARCHAR2(128)</td>
<td>Database link to a gateway for access to the source queues at the non-Oracle message queuing system.</td>
</tr>
<tr>
<td>CONSUMER</td>
<td></td>
<td>VARCHAR2</td>
<td>Name of AQ consumer</td>
</tr>
<tr>
<td>PROTOCOL</td>
<td></td>
<td>NUMBER</td>
<td>Propagation protocol number</td>
</tr>
<tr>
<td>EXCEPTION_QUEUE</td>
<td>NOT NULL</td>
<td>VARCHAR2(30)</td>
<td>Propagation-exception queue</td>
</tr>
<tr>
<td>START_DATE</td>
<td></td>
<td>DATE</td>
<td>Date to start propagation</td>
</tr>
<tr>
<td>START_TIME</td>
<td></td>
<td>VARCHAR2(8)</td>
<td>Time to start propagation</td>
</tr>
<tr>
<td>PROPAGATION_WINDOW</td>
<td></td>
<td>NUMBER</td>
<td>Duration, in seconds, for the propagation windows</td>
</tr>
<tr>
<td>NEXT_TIME</td>
<td></td>
<td>VARCHAR2(128)</td>
<td>Function to compute the start of next propagation window</td>
</tr>
<tr>
<td>LATENCY</td>
<td></td>
<td>NUMBER</td>
<td>Maximum wait time to propagate a message</td>
</tr>
<tr>
<td>LAST_DATE</td>
<td></td>
<td>DATE</td>
<td>Most recent date this schedule executed successfully</td>
</tr>
<tr>
<td>LAST_TIME</td>
<td></td>
<td>VARCHAR2</td>
<td>Time of day of the last successful execution of this schedule</td>
</tr>
<tr>
<td>FAILURES</td>
<td></td>
<td>NUMBER</td>
<td>Number of times the execution failed</td>
</tr>
<tr>
<td>DISABLED</td>
<td></td>
<td>VARCHAR2</td>
<td>N if enabled, Y if disabled and schedule will not be executed</td>
</tr>
<tr>
<td>ERROR_TIME</td>
<td></td>
<td>DATE</td>
<td>Date and time of last unsuccessful execution</td>
</tr>
<tr>
<td>ERROR</td>
<td></td>
<td>VARCHAR2</td>
<td>Oracle error number</td>
</tr>
<tr>
<td>ERROR_MSG</td>
<td></td>
<td>VARCHAR2</td>
<td>Error message text</td>
</tr>
</tbody>
</table>
### Column Name | Null? | Type | Description
---|---|---|---
THIS_DATE | | DATE | Date the current execution of this schedule started. Null if not currently executing.
THIS_TIME | | VARCHAR2 | Time of day the current execution of this schedule started. Null if not currently executing.
SID | | VARCHAR2 | Session ID of the job executing this schedule. Null if not currently executing.
TOTAL_TIME | | VARCHAR2 | Total time, in seconds, spent by the system in executing this schedule.
AVG_NUM | | NUMBER | Average number of messages propagated in a propagation window.
MAX_NUM | | NUMBER | Maximum number of messages propagated in a propagation window.
AVG_SIZE | | NUMBER | Average size of a propagated message in bytes.
MAX_SIZE | | NUMBER | Maximum size of a propagated message in bytes.
AVG_TIME | | NUMBER | Average time, in seconds, to propagate a message.
Index

Numerics
8500 error, A-39
8501 error, A-39

A
add_subscriber() procedure, 11-14
administrative package, 9-6
administrative user, creating, 7-26
API
  native, 9-4
  Oracle AQ and heterogeneous propagation, 9-2
  Oracle AQ operational API, 11-2
Application Programming Interface (API)
  also see API, 9-2
AQ_ADMINISTRATOR_ROLE, 10-3
argument
  dequeue_options, 12-9
  message_properties, 11-9, 11-11
  msgid, 11-11, 12-9
  Oracle AQ translations, 11-3, 12-3
  payload, 11-11, 12-9
  queue_name, 11-11, 12-9
authorization for MQSeries objects, 8-4
AUTHORIZATION_MODEL parameter, 7-20
Available Product Components screen, 5-5

B
boot file
  customizing, 7-4
  default file name, 7-4
  required entries, 7-5, 7-6

Bourne shell
  DISPLAY, 4-6
  ORACLE_HOME, 4-4, 4-5
  TMP, 4-7
bugs fixed in this release, 2-5
buttons
  cancel, 6-5
  close, 6-3
  deinstall products, 6-2
  help, 6-3, 6-4
  installed products, 6-2
  location, 6-3
  no, 6-4
  remove, 6-3
  save as, 6-3
  yes, 6-4

C
C shell
  DISPLAY, 4-6
  ORACLE_HOME, 4-4, 4-5
  TMP, 4-7
changes in this release
  bugs fixed, 2-5
  Oracle server dependencies, 2-2
  password encryption utility, 2-2
  channel definition, 7-6
  channel definition, MQSeries, 7-6
  choosing a repository server, 7-22
  closing a queue, A-8
  coded character set, 7-7
  commit-confirm transactions, 8-8
  configuring
gateway, 7-3
Oracle Net, 7-10
with default values, 7-3
without default values, 7-3
correlation, 11-11, 12-9, 12-10
CREATE DATABASE LINK statement, 7-20
ORA-02054, 8-12
ORA-06500, 8-13
ORA-08500, 8-11
ORA-08502, 8-14
Strict model, 8-3
create_queue() procedure, 11-5
create_queue_table() procedure, 11-4
creating
a database link, 7-20
the administrative user, 7-26

D
data dictionary
checked by pgvwbrepos.sql script, 7-25
database link
behavior, 7-19
creating, 7-20
determining available links, 7-22
dropping, 7-22
limiting active links, 7-22
dbms_aq package, 11-3, 11-6, 12-2, 12-4, 12-5, 12-8
DBMS_AQADM package, 11-17, 11-12
dbms_aqadm package, 11-4, 11-5
dbms_hqadm.create_unschedule() procedure, 11-24
DBMS_HQADM.MQSERIES_OUTBOUND protocol, 11-14, 11-16
dbms_hqadm.schedule_inbound_propagation() procedure, 12-19
dbms_hqadm.schedule_propagation() procedure, 11-24
dbms_hqadm.unschedule_inbound_propagation() procedure, 12-19
dbms_hqadm.unschedule_propagation() procedure, 11-24
DBMS_OUTPUT package, 7-24, 7-25, 7-28, 7-30
DBMS_PIPE package, 7-25, 7-28, 7-30
default values
changing during configuration, 7-4
deinstall
the Visual Workbench repository, 7-27
deinstallation, 6-2
confirmation screen, 6-4
Oracle Discoverer 4i Viewer, 6-2
using Oracle Universal Installer, 6-5
dequeue() interface call, 9-4, 9-6
dequeue() procedure, 12-2, 12-4, 12-5, 12-8, 12-9
dequeue_options argument, 12-9
dequing, 11-2, 12-2
DESCRIBE statement, 7-23, 7-24
Developer/2000, 1-7
directories
/tmp, 4-7
directory, script file, 7-23
DISPLAY, 4-6
distributed transactions
commit-confirm, 8-8
recovery requirements, 7-19
DROP DATABASE LINK statement, 7-22
dropping a database link, 7-22

E
End of Installation screen, 5-12
enqueue() interface call, 9-6
enqueue() procedure, 11-3, 11-6, 11-9
enqueuing, 11-2, 12-2
environment variable
MCAUSER, 8-3
MQ_PASSWORD, 8-3
MQ_USER_ID, 8-3
MQSERVER, 7-6
error
8500, A-39
8501, A-39
error codes, MQSeries, 8-10
MQI_CALL_FAILED exception, A-39
NO_MORE_MESSAGES exception, A-39
ORA-08500, 8-10
ORA-20001, 11-19, 11-21, 12-14, 12-16
ORA-20002, 11-19, 12-14
ORA-20003,  11-19, 12-14
ORA-20004,  11-21
ORA-20006,  12-14
ORA-20007,  12-14
ORA-20008,  12-14
ORA-20009,  12-16
errors
  common errors,  8-11
  common MQSeries errors,  8-11
  defined by PGM_SUP, A-39
  from gateway,  8-9
  from MQSeries,  8-9, 8-10
  from Oracle server,  8-9
  gateway message format,  8-10
EXECUTE permission,  10-3
EXECUTIVE privilege,  10-3

F

file
  default gateway initialization file,  7-7
File Locations screen,  5-2, 5-3, 5-4
file transfer program,  7-23
function
  UTL_RAW, B-1
  UTL_RAW.BIT_AND,  B-5
  UTL_RAW.BIT_COMPLEMENT,  B-5
  UTL_RAW.BIT_OR,  B-6
  UTL_RAW.BIT_XOR,  B-6
  UTL_RAW.CAST_TO_RAW,  B-2, B-7
  UTL_RAW.CAST_TO_VARCHAR2,  B-2, B-7
  UTL_RAW.COMpare,  B-8
  UTL_RAW.COMCAT,  B-2, B-8
  UTL_RAW.COnvert,  B-9
  UTL_RAW.COnvERT,  B-9
  UTL_RAW.COPYIES,  B-9
  UTL_RAW.LENGTH,  B-10
  UTL_RAW.OVERTLAY,  B-10
  UTL_RAW.REVERSE,  B-11
  UTL_RAW.SUBSTR,  B-12
  UTL_RAW.TRANslate,  B-12
  UTL_RAW.TRANslitE,  B-14
  UTL_RAW.XRANGE,  B-15

G

gateway
  advantages,  1-6
  boot file,  7-4
  components,  1-7
  configured with default values,  7-3
  configured without default values,  7-3
  default SIDs,  7-3
  description,  1-4
  directories,  1-9
  error message format,  8-10
  error messages,  8-9
  initialization file,  1-5, 7-2, 7-3
    authorization model,  8-2
      default,  7-7
      gateway parameters,  C-2
      with commit-confirm,  8-8
      with transaction log queue,  7-19
  known problems and restrictions,  2-6
  PGM8 package, A-2
  retrieving messages from a queue, A-9
  running environment,  8-1
  security models,  8-2
  SID,  7-4
  starting,  1-9
  structure, initialization file,  1-9
  terms,  1-5
  tracing,  8-15, C-5
  using Visual Workbench,  1-4
  verifying that it works,  8-17
GLOBAL_NAMES initialization parameter,  7-8

H

heterogeneous inbound propagation
  inbound interfaces,  9-3
  introduction,  9-5
  overview,  12-2
Heterogeneous Message Propagation
  unified API,  9-2
Heterogeneous Message Propagation option,  11-1
  chapter,  9-1
  definition,  9-2
  installing,  10-2
  heterogeneous outbound propagation,  9-3, 9-5

Index-3
description, 11-2
homogeneous propagation, 9-5
HQ_ADMINISTRATOR_ROLE, 10-3

I

inbound interfaces, 9-3
initialization file
customizing, 7-7
default filename, 7-7
gateway, 1-5, 7-2, 7-3
authorization model, 8-2
default, 7-7
parameters, C-2
with commit-confirm, 8-8
with transaction log queue, 7-19
gateway structure, 1-9
initialization parameter
GLOBAL_NAMES, 7-8
initsid.ora file, C-2
customizing the gateway initialization file, 7-7
installation log files, 4-8
installation scripts, 7-23
PL/SQL, descriptions, 10-2
Installation Status screen, 5-11
Installation Summary screen, 5-10
installing
the repository, 7-22
interface call
dequeue(), 9-4, 9-6
enqueue(), 9-6
internal scheduling table, 9-6
inventory screen, 6-3
IPC protocol, 7-11, 7-18

K

Korn shell
DISPLAY, 4-6
ORACLE_HOME, 4-4, 4-5
TMP, 4-7

L

limiting database links, 7-22

listener.ora file, 7-10
for IPC adapter, 7-11
for TCP/IP adapter, 7-11
Local Queue Manager Name screen, 5-7
LOG_DESTINATION parameter, 8-15, C-5

M

MCAUSER environment variable, 8-3
message interface package
See MIP, 1-5
message queue interface
See MQI, 1-3
message queues, definition, 1-3
message queuing, description, 1-2
message_properties argument, 11-9, 11-11
MIP
data profiles, B-2
description, 1-4, 1-5
PGM_SUP package, A-1
PGM8 package, A-1
UTL_RAW functions, B-2
MQ_PASSWORD environment variable, 8-3
MQ_USER_ID environment variable, 8-3
MQCLOSE procedure, A-36
description, A-8
MQGET procedure
retrieving short messages, A-9
MQI
definition, 1-3
gateway calls and structures, A-2
MQBACK, A-4
MQCMIT, A-4
MQCONN, A-4
MQDISC, A-4
MQINQ, A-4
MQPUT1, A-4
MQSET, A-4
tracing the calls, C-5
MQI_CALL_FAILED exception error, A-39
MQOPEN procedure
description, A-20
opening a queue, A-20
MQPUT procedure
sending short messages, A-24
MQSeries
access authorization, 8-4
access via PL/SQL, 11-2
channel definition, 7-6
client configuration definition, 1-3
common error messages, 8-11
description, 1-2
error codes, 8-10
MQSeries server, 1-8
MQSeries server definition, 1-3
queue manager definition, 1-3
triggers, 1-3
MQSERVER environment variable, 7-6
msgid argument, 11-11, 12-9

N
NO_MORE_MESSAGES exception error, A-39

O
opening a queue, A-20
option
Oracle AQ, translations, 11-6, 12-4
ORA-08500 error, 8-10
ORA-20001 error, 11-19, 11-21, 12-14, 12-16
ORA-20002 error, 11-19, 12-14
ORA-20003 error, 11-19, 12-14
ORA-20004 error, 11-21
ORA-20006 error, 12-14
ORA-20007 error, 12-14
ORA-20008 error, 12-14
ORA-20009 error, 12-16
ORA-4062 error, bug 1897823 fixed, 2-5
ORA-4062 error, bug 1925901 fixed, 2-5
ORA-4062 error, bug 2026117 fixed, 2-5
ORA-4062 error, bug 2329347 fixed, 2-4, 2-5
Oracle Advanced Queuing (AQ), 9-2
Oracle applications, 1-8
Oracle AQ
administrator’s application, 11-14
Oracle AQ arguments
translations, 11-3, 12-3
Oracle AQ options
translations, 11-6, 12-4
Oracle AQ properties, translations, 11-4, 11-6, 12-5
Oracle Financials, 1-7
Oracle Net
configuring for Oracle server, 7-16
functional part of gateway product set, 2-2
Oracle Procedural Gateway for Message Queuing
Visual Workbench
See Visual Workbench, 1-4
Oracle server
configuring inbound message propagation, 12-6
configuring outbound message propagation, 11-8
connected to gateway, 1-8
error messages, 8-9
Oracle integrating server, 1-5
production Oracle server, definition, 1-5
Oracle server dependencies, 2-2
Oracle Universal Installer, 4-8, 4-9
mounting CD-ROM, 4-9
mounting manually, 4-9
overview, 4-7
starting, 4-9
ORACLE_HOME, 4-3
Bourne shell, 4-4, 4-5
C shell, 4-4, 4-5
Korn shell, 4-4, 4-5
preventing conflicts, 4-4
Oracle9i Application Server
preinstallation tasks, 4-2
reinstallation, 6-5
oraInventory directory, 4-8
location, 4-8
outbound interfaces, 9-2
outbound message propagation
translation of interfaces, 11-2
translations, 11-3
overview
Oracle Universal Installer, 4-7

P
package
administrative, 9-6
dbms_aq, 11-3, 11-6, 12-2, 12-4, 12-5, 12-8
DBMS_AQADM, 11-17, 12-11

Index-5
dbms_aqadm, 11-4, 11-5
DBMS_OUTPUT, 7-24, 7-25, 7-28, 7-30
DBMS_PIPE, 7-25, 7-28, 7-30
PGM_BQM, 7-26, 7-28, 7-30
PGM_SUP, 7-26, 7-28, 7-30, A-29
PL/SQL, 7-28, 7-30
UTL_PG, 7-25, 7-28, 7-30
UTL_RAW, 7-23, 7-25, 7-28, 7-30

parameter
AUTHORIZATION_MODEL, 7-20
LOG_DESTINATION, 8-15, C-5
QUEUE_MANAGER, C-4
TRACE_LEVEL, 8-15, 8-16, C-5
TRANSACTION_LOG_QUEUE, 8-8, 8-12, C-6
TRANSACTION_MODEL, 7-19, C-7
ORA-02047, 8-12
TRANSACTION_RECOVERY_PASSWORD, 8-8, C-8
TRANSACTION_RECOVERY_USER, 8-8, C-10

password encryption utility, 2-2
payload argument, 11-11, 12-9
PGM_BQM package, 7-26, 7-28, 7-30
PGM_SUP package, 7-26, 7-28, 7-30
description, A-29
PGM8 package
description, A-2
error code definitions, A-39
unsupported MQI calls, A-4
PGM8.MQGMO type definition
description, A-19
PGM_SUP constants, A-29
PGM8.MQMD type definition, A-14
PGM_SUP constants, A-30
PGM8.MQOD type definition
description, A-22
PGM_SUP constants, A-35
PGM8.MQOPEN procedure
error condition 2085, 8-13
PGM_SUP constants, A-37
PGM8.MQPMO type definition
description, A-27
PGM_SUP constants, A-36
PGMADMIN, 7-26, 7-27
pgmbqm8.sql script, 7-29
pgmdeploy.sql script, 7-29
pgmdevrole.sql script, 7-29
PGMDEV role, 7-26
PGM.MQOD Type Definition, A-22
pgm.sql, 7-29
pgmsup8.sql script, 7-29
pgmundeploy.sql script, 7-28
pgvwbremove9.sql script, 7-27
pgvwbrepos9.sql script, 7-24, 7-26
Platform Welcome screen, 5-6
PL/SQL
access to MQSeries, 11-2
accessing messaging facilities, 9-2
installation scripts, descriptions, 10-2
installing missing packages, 7-28
package, 7-28, 7-30
removing packages, 7-30
running a script under SQL*Plus, 10-2
verifying packages exist, 7-28
preinstallation
environment variables, 4-3
DISPLAY, 4-6
TMP, 4-7
printing conventions, font styles, xv
private access privileges, 7-26
private repository, 7-27
privilege
EXECUTIVE, 10-3
privileges, private access, 7-26
privileges, public access, 7-26
procedural gateway
database link for access, 7-24
procedure
add_subscriber(), 11-14
create_queue(), 11-5
create_queue_table(), 11-4
dbms_hqadm.create_unschedule(), 11-24
dbms_hqadm.schedule_inbound_propagation(), 12-19
dbms_hqadm.schedule_propagation(), 11-24
dbms_hqadm.unschedule_inbound_propagation(), 12-19
dbms_hqadm.unschedule_propagation(), 11-24
dequeue(), 12-2, 12-4, 12-5, 12-8, 12-9
equeue(), 11-3, 11-6, 11-9
MQCLOSE, A-8, A-36
MQGET, A-9
MQOPEN, A-20
MQPUT, A-24
PGM8.MQOPEN, 8-13, A-37
SCHEDULE_INBOUND_PROPAGATION, 12-1
1, 12-12
UNSCHEDULE_INBOUND_PROPAGATION, 12-11, 12-15
product set for MQSeries gateway, 2-2
production Oracle server, definition, 1-5
program
  file transfer, 7-23
propagation exception queue, 12-19
propagation proxy, 9-5
property
  Oracle AQ, translations, 11-4, 11-6, 12-5
protocol
  DBMS_HQADM.MQSERIES_OUTBOUND, 11-14, 11-16
  IPC, 7-11, 7-18
  number 0, 11-14
prvtpg.sql script, 7-29
prvtrawb.plb script, 7-23
public access privileges, 7-26

Q
queue
  closing, A-8
  opening, A-20
queue manager, definition, 1-3
QUEUE_MANAGER parameter, C-4
queue_name argument, 11-11, 12-9

R
reinstallation
  Oracle9i Application Server, xvi
related publications, 8-2
relaxed security model, defined, 8-2
Remote Procedure Call (RPC), 1-4
Remote Queue Manager Name screen, 5-8
repository
  choosing a server, 7-22
  deinstall, 7-27
development privileges, 7-26
installation scripts, 7-23
installing, 7-22
installing the repository, 7-22
private, 7-27
server, definition, 7-22
requirements
  hardware, 3-2
  software, 3-3
retrieving messages
  shorter than 32\767 bytes, A-9
role
  PGMDEV, 7-26
roles
  access to Heterogeneous Message Propagation operations, 10-3
  AQ_ADMINISTRATOR_ROLE, 10-3
  HQ_ADMINISTRATOR_ROLE, 10-3
root.sh script, 5-13, 8-14

S
SCHEDULE_INBOUND_PROPAGATION
  procedure, 12-11, 12-12
scheduling table
  internal, 9-6
  outbound propagation, 9-3
screens
  Available Product Components screen, 5-5
  confirmation, 6-4
  End of Installation screen, 5-12
  File Locations screen, 5-2, 5-3, 5-4
  Installation Status screen, 5-11
  Installation Summary screen, 5-10
  inventory, 6-3
  Local Queue Manager Name screen, 5-7
  Platform Welcome screen, 5-6
  Remote Queue Manager Name screen, 5-8
script
  file directory, 7-23
  pgmbqm8.sql, 7-29
  pgmdeploy8.sql, 7-28
  pgmdeploy.sql, 7-29
  pgmsup8.sql, 7-29
  pgmundeploy.sql, 7-28

Index-7
Index-8

pgvwbremove9.sql, 7-27
pgvwbrepos9.sql, 7-24, 7-26
prvtpg.sql, 7-29
prvtrawb.plb, 7-23
root.sh, 8-14
test.sql, 8-17
utlpq.sql, 7-29
utlraw.sql, 7-23

sending messages
shorter than 32 767 bytes, A-24

setting
DISPLAY, 4-6
TMP, 4-7

SID
default values, 7-3
description, 7-3
length, 7-4

single-site transactions, 8-7

software requirements, 3-3

SQL*Net
configuring, 7-10
configuring for gateway, 7-10
purpose, 1-8

SQL*Plus
running a PL/SQL script under SQL*Plus, 10-2
starting, 4-9
Oracle Universal Installer, 4-9

statement
CREATE DATABASE LINK, 7-20
ORA-02054, 8-12
ORA-08500, 8-11, 8-13
ORA-08502, 8-14
Strict model, 8-3
DESCRIBE, 7-23, 7-24
DROP DATABASE LINK, 7-22

strict security model
defined, 8-3
running root.sh, 5-13

system ID
See SID, 7-3

T
TCP/IP protocol, 7-11, 7-17
test.sql script, 8-17

TMP, 4-7
TNS listener, 7-10
checking status, 7-14
starting, 7-14
stopping, 7-14
tnsnames.ora file, 7-16
trace feature, 8-15
TRACE_LEVEL parameter, 8-15, 8-16, C-5
transaction levels
commit-confirm, 8-8
single-site, 8-7
transaction log queue
creating, 7-19
TRANSACTION_LOG_QUEUE parameter, 8-8,
8-12, C-6
TRANSACTION_MODEL parameter, 7-19, C-7
ORA-02047, 8-12
TRANSACTION_RECOVERY_PASSWORD
parameter, 8-8, C-8
TRANSACTION_RECOVERY_USER
parameter, C-10
TRANSACTION_RECOVERY_USER
parameters, 8-8
translating, 11-2, 12-2
triggers, 1-3
MQSeries, 1-3
tuple, 11-2, 11-23, 12-2, 12-18

U
UNSUBSCRIBE_INBOUND_PROPAGATION
procedure, 12-11, 12-15
UTL_PG package, 7-25, 7-28, 7-30
UTL_RAW package, 7-23, 7-25, 7-28, 7-30
eample of using functions, B-2
function syntax, B-2
UTL_RAW.BIT_AND function, B-5
UTL_RAW.BIT_COMPLEMENT function, B-5
UTL_RAW.BIT_OR function, B-6
UTL_RAW.BIT_XOR function, B-6
UTL_RAW.CAST_TO_RAW function, B-2, B-7
UTL_RAW.CAST_TO_VARCHAR2 function, B-2,
B-7
UTL_RAW.COMpare function, B-8
UTL_RAW.CONCAT function, B-2, B-8
UTL_RAW.CONVERT function, B-9
UTL_RAW.COPIES function, B-9
UTL_RAW.LENGTH function, B-10
UTL_RAW.OVERLAY function, B-10
UTL_RAW.REVERSE function, B-11
UTL_RAW.SUBSTR function, B-12
UTL_RAW.TRANSLATE function, B-12
UTL_RAW.TRANSLITERATE function, B-14
UTL_RAW.XRANGE function, B-15
utlpg.sql script, 7-29
utlraw.sql script, 7-23

variable
environment
MCAUSER, 8-3
MQ_PASSWORD, 8-3
MQ_USER_ID, 8-3
Visual Workbench
deinstall repository, 7-27
description, 1-4
development privileges for repository, 7-26
installing the repository, 7-22
MIP, A-1
UTL_RAW functions, B-1