Tekelec EAGLE® 5 Signaling Application System (SAS)

Provisioning Database Interface Manual

Table of Chapters

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

Chapter 2. Functional Description

Chapter 3. PDBI Request/Response Messages

Chapter 4. PDBI Sample Sessions

Appendix A. PDBI Message Error Codes

Index

910-0348-001 Rev A, September 2005

Tekelec EAGLE® 5
Signaling Application System

Provisioning Database Interface Manual

910-0348-001 Revision A
September 2005

TEKELEC

Notice
Information in this documentation is subject to change without notice. Unauthorized use or copying of this documentation can result in civil or criminal penalties.

Any export of Tekelec products is subject to the export controls of the United States and the other countries where Tekelec has operations.

No part of this documentation may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying or recording, for any purpose without the express written permission of an authorized representative of Tekelec.

Other product names used herein are for identification purposes only, and may be trademarks of their respective companies.

Trademarks
Multi Purpose Server (MPS) is a trademark of Tekelec.

The Tekelec logo, EAGLE, G-Flex, G-Port, IP^7, IP^{7}Edge, IP^7 Secure Gateway, and TALI are registered trademarks of Tekelec, Inc. TekServer is a trademark of Tekelec, Inc. All other trademarks are the property of their respective owners.

Patents
This product is covered by one or more of the following U.S. and foreign patents:

U.S. Patent Numbers:

5,008,929, 5,953,404, 6,167,129, 6,324,183, 6,327,350, 6,606,379, 6,639,981, 6,647,113, 6,662,017, 6,735,441, 6,745,041, 6,765,990, 6,795,546, 6,819,932, 6,836,477, 6,839,423, 6,885,872

Ordering Information
Additional copies of this document can be ordered from:

Tekelec Network Signaling Group
Attention: Central Logistics
5200 Paramount Parkway
Morrisville, North Carolina, 27560

Or e-mail your request to *CentralLogistics@tekelec.com*.

Table of Contents

Chapter 1. Introduction

Overview 1-1
Scope and Audience 1-1
Manual Organization 1-2
Related Publications 1-2
Documentation Packaging, Delivery, and Updates 1-7
Documentation Admonishments 1-8
Customer Assistance 1-9
Customer Contact Center 1-9
Acronyms 1-10

Chapter 2. Functional Description

General Description 2-1
EPAP (EAGLE Provisioning Application Processor) 2-1
DSM (Database Services Module) 2-2
Introduction to Platform Services 2-3
Introduction to the Data Model on the Platform 2-4
Data Organization 2-5
System Architecture 2-6
System Overview and Terminology 2-8
Network Connections 2-8
EPAP Status Reporting and Alarm Handling 2-9
Provisioning Database Interface Description 2-10
Socket Based Connection 2-10
String-Based Messages 2-10
Security 2-11
Transaction-Oriented API 2-11
Batch-Oriented/Bulk Load 2-12
Command Atomicity 2-12
Provisioning Ranges of Subscriber Numbers 2-12
Transparency of Redundant Systems 2-13
Logs 2-13
Crash Recovery 2-13
Request IDs 2-13

Multiple Session Connectivity 2-13
Request Queue Management 2-14
Interface Configuration and Installation 2-14
File Formats 2-15
Debug Log 2-15
Import/Export Files 2-15

Chapter 3. PDBI Request/Response Messages

Overview 3-2
Message Definitions 3-2
Request IDs 3-3
Optional Parameters 3-3
Common Response Format 3-3
Number Prefixes 3-4
Common Responses 3-4
Common Response Messages 3-5
Messages 3-8
Connect 3-8
Disconnect 3-10
Begin Transaction 3-11
End Transaction 3-13
Abort Transaction 3-14
Create Subscription 3-15
Update Subscription 3-22
Delete Subscription 3-28
Retrieve Subscription Data 3-30
Create Network Entity 3-36
Update Network Entity 3-40
Delete Network Entity 3-44
Retrieve Network Entity 3-46
Switchover 3-48
PDBA Status Query 3-51
Dump Connections 3-52
Create IMEI Data 3-53
Update IMEI Data 3-58
Delete IMEI Data 3-61
Retrieve IMEI Data 3-64
Request DSM Report 3-68

Chapter 4. PDBI Sample Sessions

Introduction 4-2
Network Entity Creation 4-2
Simple Subscription Data Creation 4-3
Update Subscription Data 4-4
Simple Queries 4-5
Multiple Response Query 4-7
Abort Transaction 4-8
Update Request In Read Transaction 4-9
Write Transaction In Standby Connection 4-10
Simple Subscription Data Creation with Single Txnmode 4-11
Single IMEI Data 4-12
IMEI Block Data 4-13
Asynchronous DSM Report 4-14
Synchronous DSM Report 4-15
DSM List 4-16

Appendix A. PDBI Message Error Codes

Index

List of Figures

Figure 2-1. Example EPAP/PDBA Network 2-2
Figure 2-2. PDBI System Architecture 2-7
Figure 2-3. MPS/EPAP System Configuration 2-9

List of Tables

Table 3-1. Parse Failure Reasons 3-5
Table 3-2. Connect Response Return Codes 3-10
Table 3-3. Disconnect Response Return Codes 3-11
Table 3-4. Begin Transaction Response Return Codes 3-13
Table 3-5. End Transaction Response Return Codes 3-14
Table 3-6. Abort Transaction Response Return Code 3-15
Table 3-7. Create Subscription Response Return Codes 3-22
Table 3-8. Update Subscription Response Return Codes 3-27
Table 3-9. Delete Subscription Response Return Codes 3-30
Table 3-10. Retrieve Subscription Data Response Return Codes ... 3-36
Table 3-11. Create Network Entity Response Return Codes 3-39
Table 3-12. Update Network Entity Response Return Codes 3-44
Table 3-13. Delete Network Entity Response Return Codes 3-45
Table 3-14. Retrieve Network Entity Response Return Codes 3-48
Table 3-15. Switchover Response Return Codes 3-50
Table 3-16. PDBA Status Query Response Return Code 3-52
Table 3-17. Dump Connections Response Return Code 3-53
Table 3-18. Create IMEI Response Return Codes 3-57
Table 3-19. Update IMEI Response Return Codes 3-60
Table 3-20. Update IMEI Response Return Codes 3-64
Table 3-21. Retrieve IMEI Response Return Codes 3-67
Table 3-22. Retrieve DSM Report Response Return Codes 3-70
Table 3-23. Retrieve DSM List Response Return Codes 3-72
Table 4-1. Network Entity Creation Example 4-2
Table 4-2. Simple Subscription Data Creation Example 4-3
Table 4-3. Update Subscription Data Example 4-4
Table 4-4. Simple Queries Example 4-5
Table 4-5. Multiple Response Query Example 4-7
Table 4-6. Abort Transaction Example 4-8
Table 4-7. Update Request in Read Transaction Example 4-9
Table 4-8. Write Transaction in Standby Connection Example 4-10

Table 4-9. Simple Subscription Data Creation with Single Txnmode Example 4-11
Table 4-10. Single IMEI Data Example 4-12
Table 4-11. IMEI Block Data Example 4-13
Table 4-12. Asynchronous DSM Report Example 4-14
Table 4-13. Synchronous DSM Report Example 4-15
Table 4-14. DSM List Example 4-16
Table A-1. PDBI Message Error Codes A-1

1

Introduction

Overview 1–1
Scope and Audience 1–1
Manual Organization 1–2
Related Publications 1–2
Documentation Packaging, Delivery, and Updates 1–7
Documentation Admonishments 1–8
Customer Assistance 1–9
Acronyms 1–10

Overview

The *Provisioning Database Interface Manual* defines the interface that is used to populate the Provisioning Database (PDB) for the G-Flex, G-Port, EIR, and INP features of the EAGLE 5 SAS. The chapters include descriptions of Provisioning Database Interface (PDBI), Provisioning Database Application (PDBA), EAGLE Provisioning Application Processor (EPAP) function, PDBI request and response messages, and PDBI sample sessions.

Scope and Audience

This manual is intended for the application personnel responsible for transferring data from the customer system through the PDBI to a PDBA in an EPAP. Users of this manual and the others in the EAGLE 5 SAS family of documents must have a working knowledge of telecommunications and network installations.

Manual Organization

This document is organized into the following chapters:

- Chapter 1, *"Introduction,"* contains general information about the PDBI documentation, the organization of this manual, and how to get technical assistance.
- Chapter 2, *"Functional Description,"* provides an overview of PDBI, EPAP, PDBA, and DSM functions.
- Chapter 3, *"PDBI Request/Response Messages,"* describes available requests and the possible responses for PDBI request/response messages.
- Chapter 4, *"PDBI Sample Sessions,"* contains example flow scenarios for the PDBI request/response messages.
- Appendix A, "*PDBI Message Error Codes,"* lists the PDBI error codes and text.

Related Publications

The *Provisioning Database Interface Manual* is part of the EAGLE 5 SAS documentation set and may reference related manuals of this set. The documentation set includes the following manuals:

- The *Commands Manual* contains procedures for logging into or out of the EAGLE 5 SAS, a general description of the terminals, printers, the disk drive used on the system, and a description of all the commands used in the system.
- The *Commands Pocket Guide* is an abridged version of the *Commands Manual*. It contains all commands and parameters, and it shows the command-parameter syntax.
- The *Commands Quick Reference Guide* contains an alphabetical listing of the commands and parameters. The guide is sized to fit a shirt-pocket.
- The *Commands Error Recovery Manual* contains the procedures to resolve error message conditions generated by the commands in the *Commands Manual*. These error messages are presented in numerical order.
- The *Database Administration Manual – Features* contains procedural information required to configure the EAGLE 5 SAS to implement these features:
  - — X.25 Gateway
  - — STP LAN
  - — Database Transport Access
  - — GSM MAP Screening

— EAGLE 5 SAS Support for Integrated Sentinel

- The *Database Administration Manual - Gateway Screening* contains a description of the Gateway Screening (GWS) feature and the procedures necessary to configure the EAGLE 5 SAS to implement this feature.
- The *Database Administration Manual – Global Title Translation* contains procedural information required to configure an EAGLE 5 SAS to implement these features:
  - — Global Title Translation
  - — Enhanced Global Title Translation
  - — Variable Length Global Title Translation
  - — Interim Global Title Modification
  - — Intermediate GTT Load Sharing
  - — ANSI-ITU-China SCCP Conversion
- The *Database Administration Manual - IP7 Secure Gateway* contains procedural information required to configure the EAGLE 5 SAS to implement the SS7-IP Gateway.
- The *Database Administration Manual – SEAS* contains the EAGLE 5 SAS configuration procedures that can be performed from the Signaling Engineering and Administration Center (SEAC) or a Signaling Network Control Center (SNCC). Each procedure includes a brief description of the procedure, a flowchart showing the steps required, a list of any EAGLE 5 SAS commands that may be required for the procedure but that are not supported by SEAS, and a reference to optional procedure-related information, which can be found in one of these manuals:
  - — Database Administration Manual – Gateway Screening
  - — Database Administration Manual – Global Title Translation
  - — Database Administration Manual – SS7
- The *Database Administration Manual – SS7* contains procedural information required to configure an EAGLE 5 SAS to implement the SS7 protocol.
- The *Database Administration Manual – System Management* contains procedural information required to manage the EAGLE 5 SAS database and GPLs, and to configure basic system requirements such as user names and passwords, system-wide security requirements, and terminal configurations.

- The *Dimensioning Guide for EPAP Advanced DB Features* is used to provide EPAP planning and dimensioning information. This manual is used by Tekelec personnel and EAGLE 5 SAS customers to aid in the sale, planning, implementation, deployment, and upgrade of EAGLE 5 SAS systems equipped with one of the EAGLE 5 SAS EPAP Advanced Database (EADB) Features.
- The *ELAP Administration Manual* defines the user interface to the EAGLE 5 SAS LNP Application Processor on the MPS/ELAP platform. The manual defines the methods for accessing the user interface, menus, screens available to the user and describes their impact. It provides the syntax and semantics of user input, and defines the output the user receives, including information and error messages, alarms, and status.
- The *EPAP Administration Manual* describes how to administer the EAGLE 5 SAS Provisioning Application Processor on the MPS/EPAP platform. The manual defines the methods for accessing the user interface, menus, and screens available to the user and describes their impact. It provides the syntax and semantics of user input and defines the output the user receives, including messages, alarms, and status.
- The *Feature Manual - EIR* provides instructions and information on how to install, use, and maintain the EIR feature on the Multi-Purpose Server (MPS) platform of the EAGLE 5 SAS. The feature provides network operators with the capability to prevent stolen or disallowed GSM mobile handsets from accessing the network.
- The *Feature Manual - G-Flex C7 Relay* provides an overview of a feature supporting the efficient management of Home Location Registers in various networks. This manual gives the instructions and information on how to install, use, and maintain the G-Flex feature on the Multi-Purpose Server (MPS) platform of the EAGLE 5 SAS.
- The *Feature Manual - G-Port* provides an overview of a feature providing the capability for mobile subscribers to change the GSM subscription network within a portability cluster while retaining their original MSISDNs. This manual gives the instructions and information on how to install, use, and maintain the G-Port feature on the Multi-Purpose Server (MPS) platform of the EAGLE 5 SAS.
- The *Feature Manual - INP* provides the user with information and instructions on how to implement, utilize, and maintain the INAP-based Number Portability (INP) feature on the Multi-Purpose Server (MPS) platform of the EAGLE 5 SAS.
- The *FTP-Based Table Retrieve Application (FTRA) User Guide* describes how to set up and use a PC to serve as the offline application for the EAGLE 5 SAS FTP Retrieve and Replace feature.

- The *Hardware Manual* - EAGLE 5 SAS contains hardware descriptions and specifications of Tekelec's signaling products. These include the EAGLE 5 SAS, OEM-based products such as the ASi 4000 Service Control Point (SCP), the Netra-based Multi-Purpose Server (MPS), and the Integrated Sentinel with Extended Services Platform (ESP) subassembly.

  The Hardware Manual provides an overview of each system and its subsystems, details of standard and optional hardware components in each system, and basic site engineering. Refer to this manual to obtain a basic understanding of each type of system and its related hardware, to locate detailed information about hardware components used in a particular release, and to help configure a site for use with the system hardware.

- The *Hardware Manual - Tekelec 1000 Application Server* provides general specifications and a description of the Tekelec 1000 Applications Server (T1000 AS). This manual also includes site preparation, environmental and other requirements, procedures to physically install the T1000 AS, and troubleshooting and repair of Field Replaceable Units (FRUs).

- The *Hardware Manual - Tekelec 1100 Application Server* provides general specifications and a description of the Tekelec 1100 Applications Server (T1000 AS). This manual also includes site preparation, environmental and other requirements, procedures to physically install the T1100 AS, and troubleshooting and repair of Field Replaceable Units (FRUs).

- The *Installation Manual* - EAGLE 5 SAS contains cabling requirements, schematics, and procedures for installing the EAGLE 5 SAS along with LEDs, Connectors, Cables, and Power Cords to Peripherals. Refer to this manual to install components or the complete systems.

- The *Installation Manual - Integrated Applications* provides the installation information for integrated applications such as EPAP 4.0 or earlier (Netra-based Multi-Purpose Server (MPS) platform) and Sentinel. The manual includes information about frame floors and shelves, LEDs, connectors, cables, and power cords to peripherals. Refer to this manual to install components or the complete systems.

- The *LNP Database Synchronization Manual - LSMS with* EAGLE 5 SAS describes how to keep the LNP databases at the LSMS and at the network element (the EAGLE 5 SAS is a network element) synchronized through the use of resynchronization, audits and reconciles, and bulk loads. This manual is contained in both the LSMS documentation set and in the EAGLE 5 SAS documentation set.

- The *LNP Feature Activation Guide* contains procedural information required to configure the EAGLE 5 SAS for the LNP feature and to implement these parts of the LNP feature on the EAGLE 5 SAS:

  — LNP services

  — LNP options

  - — LNP subsystem application
  - — Automatic call gapping
  - — Triggerless LNP feature
  - — Increasing the LRN and NPANXX Quantities on the EAGLE 5 SAS
  - — Activating and Deactivating the LNP Short Message Service (SMS) feature
- The *Maintenance Manual* contains procedural information required for maintaining the EAGLE 5 SAS and the card removal and replacement procedures. The *Maintenance Manual* provides preventive and corrective maintenance procedures used in maintaining the different systems.
- The *Maintenance Pocket Guide* is an abridged version of the Maintenance Manual and contains all the corrective maintenance procedures used in maintaining the EAGLE 5 SAS.
- The *Maintenance Emergency Recovery Pocket Guide* is an abridged version of the Maintenance Manual and contains the corrective maintenance procedures for critical and major alarms generated on the EAGLE 5 SAS
- The *MPS Platform Software and Maintenance Manual* - EAGLE 5 SAS *with Tekelec 1000 Application Server* describes the platform software for the Multi-Purpose Server (MPS) based on the Tekelec 1000 Application Server (T1000 AS) and describes how to perform preventive and corrective maintenance for the T1000 AS-based MPS. This manual should be used with the EPAP-based applications (EIR, G-Port, G-Flex, and INP).
- The *MPS Platform Software and Maintenance Manual* - EAGLE 5 SAS *with Tekelec 1100 Application Server* describes the platform software for the Multi-Purpose Server (MPS) based on the Tekelec 1100 Application Server (T1100 AS) and describes how to perform preventive and corrective maintenance for the T1100 AS-based MPS. This manual should be used with the ELAP-based application (LNP).
- The *Provisioning Database Interface Manual* defines the programming interface that populates the Provisioning Database (PDB) for the EAGLE 5 SAS features supported on the MPS/EPAP platform. The manual defines the provisioning messages, usage rules, and informational and error messages of the interface. The customer uses the PDBI interface information to write his own client application to communicate with the MPS/EPAP platform.
- The *Previously Released Features Manual* summarizes the features of previous EAGLE, EAGLE 5 SAS, and IP^7 Secure Gateway releases, and it identifies the release number of their introduction.

- The *Release Documentation* contains the following documents for a specific release of the system:
  - — *Feature Notice* - Describes the features contained in the specified release. The Feature Notice also provides the hardware baseline for the specified release, describes the customer documentation set, provides information about customer training, and explains how to access the Customer Support website.
  - — *Release Notice* - Describes the changes made to the system during the lifecycle of a release. The Release Notice includes Generic Program Loads (GPLs), a list of PRs resolved in a build, and all known PRs.

    NOTE: The *Release Notice* is maintained solely on Tekelec's Customer Support site to provide you with instant access to the most up-to-date release information.
  - — *System Overview* - Provides high-level information on SS7, the IP7 Secure Gateway, system architecture, LNP, and EOAP.
  - — *Master Glossary* - Contains an alphabetical listing of terms, acronyms, and abbreviations relevant to the system.
  - — *Master Index* - Lists all index entries used throughout the documentation set.
- The *System Manual – EOAP* describes the Embedded Operations Support System Application Processor (EOAP) and provides the user with procedures on how to implement the EOAP, replace EOAP-related hardware, device testing, and basic troubleshooting information.

Documentation Packaging, Delivery, and Updates

Customer documentation is provided with each system in accordance with the contract agreements. It is updated whenever significant changes that affect system operation or configuration are made. Updates may be issued as an addendum, or a reissue of the affected documentation.

The document part number appears on the title page along with the current revision of the document, the date of publication, and the software release that the document covers. The bottom of each page contains the document part number and date of publication.

Two types of releases are major software releases and maintenance releases. Maintenance releases are issued as addenda with a title page and change bars. On changed pages, the date and document part number are changed; on unchanged pages that accompany the changed pages, the date and document part number are unchanged.

When the software release has a minimum affect on documentation, an addendum is provided. The addendum contains an instruction page, a new title page, a change history page, and replacement chapters with the date of publication, the document part number, and change bars.

If a new release has a major impact on documentation, such as a new feature, the entire documentation set is reissued with a new part number and a new release number.

Documentation Admonishments

Admonishments are icons and text throughout this manual that alert the reader to assure personal safety, to minimize possible service interruptions, and to warn of the potential for equipment damage. This manual has three admonishments, listed in descending order of priority.

	**DANGER:** (This icon and text indicate the possibility of *personal injury*.)
	**CAUTION:** (This icon and text indicate the possibility of *service interruption*.)
	**WARNING:** (This icon and text indicate the possibility of *equipment damage*.)

Customer Assistance

The Tekelec Customer Contact Center offers a point of contact through which customers can receive support for problems. The Tekelec Customer Contact Center is staffed with highly-trained engineers to provide solutions to technical questions and issues seven days a week, twenty-four hours a day. A variety of service programs are available through the Tekelec Customer Contact Center to maximize the performance of Tekelec products that meet and exceed customer needs.

Customer Contact Center

To receive technical assistance, call the Tekelec Customer Contact Center at one of the following locations by one of the following methods:

- Tekelec, UK

  Phone: +44 1784 467804

  Fax: +44 1784 477120

  Email: *ecsc@tekelec.com*

- Tekelec, USA

  Phone (within continental US): (888) 367-8552

  (outside continental US): +1 919-460-2150

  Email: *support@tekelec.com*

When the call is received, a Customer Service Report (CSR) is issued to record the request for service. Each CSR includes an individual tracking number.

Once a CSR is issued, Technical Services determines the classification of the trouble. If a critical problem exists, emergency procedures are initiated. If the problem is not critical, information regarding the serial number of the system, COMMON Language Location Identifier (CLLI), initial problem symptoms (includes outputs and messages) is recorded. A primary Technical Services engineer is also assigned to work on the CSR and provide a solution to the problem. The CSR is closed when the problem is resolved.

Acronyms

ACK .....Acknowledgment message

ANSI .....American National Standards Institute

API .....Application Programming Interface

ASCII.....American Standard Code for Information Interchange

ASM .....Application Services Module

CCGT .....Cancel Called Global Title

CD-ROM .....Compact Disk Read Only Memory

CPS.....Customer Provisioning System

CPU .....Central Processing Unit

DA .....Digit Action

DB.....Database

DCB.....Device Control Block

DCM.....Data Communications Module

DN.....Dialed Number (DN can refer to any mobile or wireline subscriber number, and can include MSISDN, MDN, MIN, or the wireline Dialed Number.)

DSM .....Database Services Module

EIR.....Equipment Identity Register

EPAP .....EAGLE Provisioning Application Processor

ETSI .....European Telecommunications Standards Institute

FSM .....Forward Short Message

GB.....Gigabyte

GDB.....G-Flex/G-Port Data Base

G-Flex.....GSM Flexible Numbering

GMSC .....Gateway Mobile Switching Center

GPL .....Generic Program Load

G-Port .....GSM Mobile Number Portability

GPDB .....G-Port Database

GSM .....Global System for Mobile Telecommunication

GTA.....Global Title Address

GTT .....Global Title Translation

HLR.....Home Location Register

ID.....Identifier

IMEI .....International Mobile Equipment Identity

IMSI .....International Mobile Station Identifier

IN .....Intelligent Network

INAP.....Intelligent Network Application Protocol

INP.....INAP-based Number Portability

IP .....Internet Protocol

IS-ANR .....In Service - Abnormal

ISDN .....Integrated Services Digital Network

IS-NR .....In Service - Normal

ITU .....International Telecommunication Union

KB.....Kilobyte

KSR .....Keyboard Send Receive

LAN .....Local Area Network

LIM .....Link Interface Module

LNP.....Local Number Portability

LSMS.....Local Service Management System

MA .....Mated Application

MAP.....Mobile Application Part

MB.....Megabyte

MDN.....Mobile Dialed Number

MIN.....Mobile Identification Number

MMI .....Man-Machine Interface

MNP.....Mobile Number Portability

MO_FSM .....Mobile Originated Forward Short Message

MPS.....Multi-Purpose Server

MSC .....Mobile Switching Center

MSISDN.....Mobile Switching Integrated Services Digital Network Number

MSU .....Message Signal Unit

MT_FSM .....Mobile Terminated Forward Short Message

MTS .....Message Transfer System

MTSU .....Message Transfer System Utilities

MTT.....Mapped Translation Type

NAK .....Negative Acknowledgment message

NE .....Network Entity

NEBS .....Network Equipment Building Standards

OAM .....Operation Administration & Maintenance

OAP.....Operation System Support/Application Processor

OOS-MT-DSBLD .....Out of Service - Maintenance Disabled

OS .....(1) Operations Systems
(2) Operating System

PC .....Point Code

PDB .....Provisioning Database

PDBA .....Provisioning Database Application

PDBI .....Provisioning Database Interface

PID .....Process Identifier

PPP .....Point-to-Point Protocol

PPSMS .....Prepaid SMS

PT.....Portability Type

RFC.....Request for Comment document

RI .....Routing Indicator

RMTP .....Reliable Multicast Transport Protocol

RN .....Routing Number

RTDB.....Real-Time Database

SCCP .....Signaling Connection Control Part

SEAC.....Signaling Engineering and Administration Center

SMS .....Short Message Service

SMSC ..... Short Message Service Center

SNCC ..... Signaling Network Control Center

SP ..... Signalling Point

SRF ..... Signaling Relay Function

SRI ..... Send Routing Information

SS7 ..... Signaling System #7

SSN ..... Subsystem Number

STP ..... Signaling Transfer Point

SVN ..... Software Version Number

TCP ..... Transmission Control Protocol

TDM ..... Terminal Disk Module

TKLC ..... Tekelec

TSM ..... Translation Services Module

UAM ..... Unsolicited Alarm Message

UDP ..... User Datagram Protocol

UIM ..... Unsolicited Information Message

VMSC ..... Visited Mobile Switching Center

VSCCP ..... VxWorks Signaling Connection Control Part

2

Functional Description

General Description..... 2–1

System Architecture 2–6

Provisioning Database Interface Description 2–10

File Formats 2–15

General Description

The Provisioning Database Interface (PDBI) provides commands that move provisioning information from the customer database to the provisioning database (PDB) in the Active EPAP in an EAGLE 5 SAS. The customer uses the PDBI request/response messages to create a provisioning application that communicates with the EPAP Provisioning Database Application (PDBA) over the customer network.

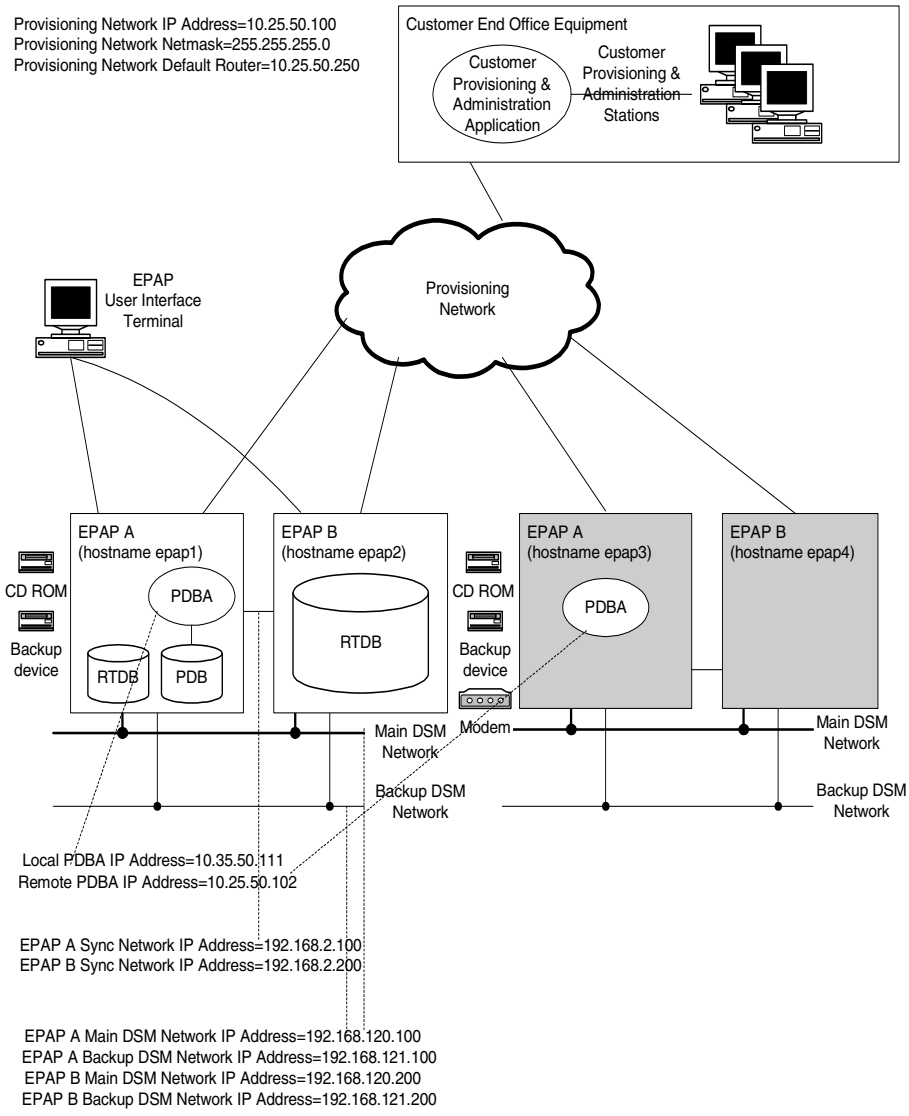
EPAP (EAGLE Provisioning Application Processor)

As shown in Figure 2-1 on page 2-2, the provisioning system contains two mated EPAPs. The primary purpose of the EPAPs is to maintain the PDB and to download copies of the RTDB to the DSM boards. At any given time, only one is actively communicating with the DSM boards. The other EPAP is in Standby mode. The EPAP interfaces to the customer source of provisioning information through the Provisioning Database Interface (PDBI), which provides PDB updates. As the customer submits provisioning requests, the Active EPAP updates the PDB and its copy of the RTDB. Once the update is applied to the Active PDB, it is sent to the Standby EPAP. Each EPAP maintains a copy of the RTDB in B-Tree format. When a DSM needs a copy of the RTDB, the EPAP downloads the B-Tree file to the DSM. Each DSM then uses the B-Tree file to create its own copy of the RTDB database.

DSM (Database Services Module)

As Figure 2-1 shows, the provisioning system uses up to 25 DSM cards. The DSMs run a version of the SCCP application that has been ported to the VxWorks OS. To differentiate the DSM-VxWorks-SCCP from the SCCP that runs on the TSM cards, the DSM version has been named VSCCP. The extra memory is required to hold a copy of the RTDB. Multiple DSMs are used to provide a means of load balancing in high-traffic situations. The database is in a B-Tree format to facilitate rapid lookups.

Figure 2-1. Example EPAP/PDBA Network



Each DSM contains an identical database. The DSM RTDBs need to be identical to the one maintained by the EPAPs. However, there are several reasons why the various databases might not be identical. When a DSM card is initialized, it has to download a current copy of the B-Tree RTDB file from the EPAP. While that card is being downloaded, it cannot be used to provide VSCCP services. Another condition that leads to the databases being out-of-sync occurs when the EPAP processes update from its provisioning source. Those updates are applied immediately to the EPAP PDB database as they are received, but there can be a delay before the updates get sent to the DSMs.

Two possible scenarios lead to the condition where a DSM might not have enough memory to hold the entire database. In the first case, the database is downloaded successfully to the DSM but subsequent updates eventually increase the size of the database beyond the DSM memory capacity. In this situation, it is desirable to continue processing provisioning traffic, even though the database might not be as up-to-date as it could be. The other case occurs when a DSM card is booted. At that time, if it is determined that the card does not have sufficient memory to hold the entire database, the database is not loaded on that card. The DSM is responsible for recognizing and reporting out-of-memory conditions.

Introduction to Platform Services

The Provisioning DataBase Interface (PDBI) allows one or several independent information systems supplied and maintained by the network operator to be used for provisioning the G-Flex, G-Port, INP, and/or EIR databases and for configuring the G-Flex, G-Port, INP and/or EIR systems. Through the PDBI, the independent information systems can add, delete, change or retrieve information about any IMSI/MSISDN/SP association or portability information.

The active/standby status of the PDBA can also be changed. For the G-Flex and G-Port features, SP generally refers to an HLR. Also note that the terms MSISDN and DN are used interchangeably throughout this document.

The G-Flex feature allows mobile network operators to optimize the use of subscriber numbers (IMSIs and MSISDNs) and number ranges by providing a logical link between any MSISDN and any IMSI. This allows subscribers to be easily moved from one HLR to another. It also allows each HLR to be filled to 100 percent capacity by allowing MSISDN/IMSI ranges to be split over different HLRs and individual MSISDNs/IMSIs to be assigned to any HLR. G-Flex also eliminates the need to maintain subscriber routing information at every MSC in the network.

The G-Port MNP feature implements mobile number portability for GSM networks and supports the SRF-based MNP solution as defined in ETSI standards. G-Port allows the subscriber to retain the MSISDN number when changing subscription networks. The user's IMSI is not portable. For call-related messages, G-Port acts as a “NP HLR”, in the case where the number has been exported, by responding to the switch with a MAP SRI ack message. For calls to imported numbers and non-call related messages, G-Port performs message relay.

The INP (INAP-based Number Portability) feature implements IN-based number portability (using INAP protocol). It is also used by wireline network operators in accordance with ITU Number Portability supplements, or by wireless network operators in accordance with ETSI NP standards. INP provides both query/response and message relay functionality.

The EIR (Equipment Identity Register) feature implements handset security within the GSM network. It does this by allowing carriers to provision IMEIs (International Mobile Equipment Identity) in the database and assigning them a list type. List types are Black, Gray, and White. When an IMEI is placed on the Black list, the carrier is able to prevent the handset from accessing their network. A White listed IMEI is allowed access to the network, while a Gray may require additional screening but is typically allowed access to the network.

The PPSMS (Prepaid Short Message Service) feature uses the G-Port DN portability type (PT) field to identify two types of prepaid subscribers whose originated short messages (as part of SMS) need to be intercepted and forwarded to a corresponding intelligent network platform for verification. Since these subscribers are either ported in or not ported, the new PT values do not logically overlap the existing values. The subscribers being ported in or not ported for PPSMS is also the reason for the rule that the PT cannot be set when the DN is associated with an SP is removed. In order to minimize changes to the interface, the PT field is not being added to the commands where an IMSI is provided as input. PPSMS is a part of G-Port that is activated separately.

The IS-41 to GSM Migration feature uses the G-Port MSISDN portability type (PT) field to identify subscribers that have migrated from IS-41 to GSM, but maintain only a single GSM handset. This category also includes new subscribers who sign up for GSM service only and have only one handset, but are given a number from the existing IS-41 number range. Since these subscribers are either migrated or not migrated, the new PT values do not logically overlap the existing values.

G-Port, G-Flex and INP share the same database when operated together on a single node.

Introduction to the Data Model on the Platform

The PDB uses an object-oriented approach for data organization. The data is organized into three independent “objects” that correspond to MSISDNs, IMSIs and SPs/RNs. These “objects” are a subset of the database. Associations are established between an IMSI and MSISDN, IMSI and SP/RN, MSISDN and SP/RN or IMSI, MSISDN and SP/RN through the use of pointers between the objects.

The database is created as follows:

- When an IMSI, MSISDN or NE (that is, an SP identifier) is created, this data is added to the corresponding object, which is a subset of the database.
- When an IMSI, MSISDN or NE is deleted, the related data is removed from the corresponding object.

- When an association is established between an IMSI, MSISDN and SP/RN, pointers are set up between the appropriate objects.
- When an association is removed, the pointers between the objects are removed.

For example, assume that the database already contains several IMSIs, MSISDNs and SP addresses, but that no associations have been established. The IMSIs exist in the 'IMSI object,' that is, the IMSI portion of the database. Likewise, the MSISDNs exist in the 'MSISDN object' and the SP addresses exist in the 'SP object.' When the **`ent_sub`** or **`upd_sub`** commands are used to establish an association between an IMSI and an MSISDN, a pointer is created that points to the correct location in the 'MSISDN object,' that is, the correct portion of the database where the MSISDNs reside. The same process occurs when other associations are established, such as IMSI pointing to SP, MSISDN pointing to SP, or IMSI pointing to MSISDN pointing to SP.

The EIR feature introduces the IMEI to the database. The IMEI for EIR may be associated with up to 8 IMSIs, but it is important to note that this IMSI has no relationship to the existing IMSI used by the G-Port/G-Flex feature (ent_sub, upd_sub,dlt_sub, rtrv_sub) commands. In other words, IMSIs provisioned for EIR are strictly added to the EIR database only. An IMSI may appear in both the G-Port/G-Flex database and the EIR database, but must be provisioned by both sets of commands (ent-eir and ent-sub).

Data Organization

MSISDN data is provisioned into two tables: a single instance table (Single DNs) and a block instance table (DN Blocks). The database considers both Single DNs and DN Blocks as entities in their own right. Therefore, a distinction must be made between the terms 'DN range' and 'DN Block' as they are used in this document. A DN Block is considered to be an autonomous entity, just as a Single DN is. A DN range is just a range of numbers. Within a specified DN range, several Single DNs and also several DN Blocks may exist. For instance, consider the following example:

Assume the following single DNs are provisioned:

10050

10080

10900

Also assume the following DN blocks are provisioned:

10000-10100

10500-10800

11000-12000

Some commands accept a DN range as a requesting parameter (for example, the **rtrv_sub** command). Assume the following DN ranges are used in a command:

10080-10600

10850-11500

10000-10040

10400-13000

Then the following relationships are true:

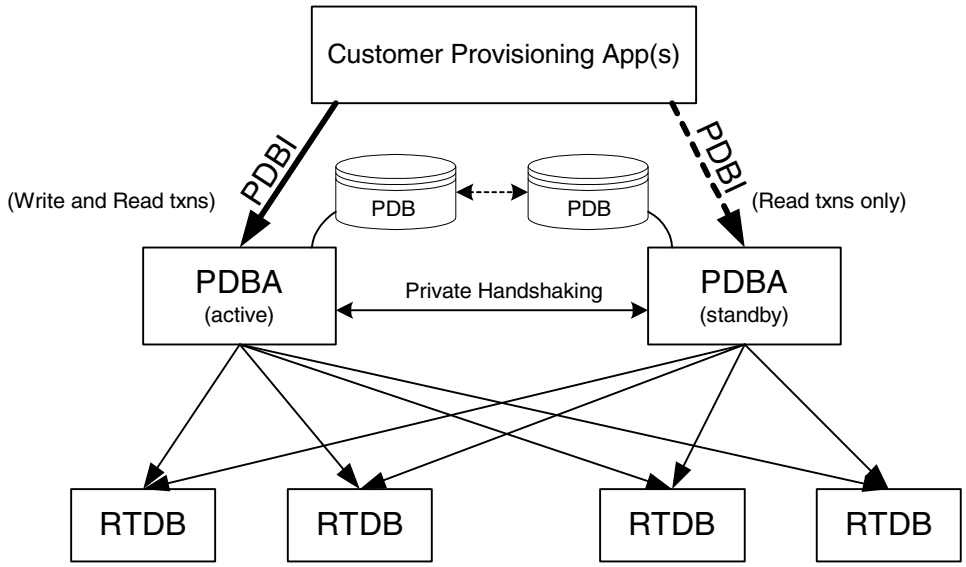
- DN range 10080-10600 encompasses the Single DN 10080 and DN Blocks 10000-10100 and 10500-10800
- DN range 10850-11500 encompasses the Single DN 10900 and DN Block 11000-12000
- DN range 10000-10040 encompasses no Single DNs and DN Block 10000-10100
- DN range 10400-13000 encompasses the Single DN 10900 and DN Blocks 10500-10800 and 11000-12000

The IMEI is also provisioned into two tables: a single instance table (Individual IMEIs) and a block instance table (IMEI Blocks). IMEIs work the same way as the DNs and DN Blocks.

System Architecture

There are two Provisioning Database Applications (PDBAs), one in EPAP A on each EAGLE 5 SAS. They follow an Active/Standby model. These processes are responsible for updating and maintaining the Provisioning Database (PDB). Customer provisioning applications connect to the Active PDBA and use PDBI request/response messages to populate and query the PDB. The PDBA then forwards the updates to the EPAP real-time database (RTDB). See Figure 2-2.

Figure 2-2. PDBI System Architecture



Updates that are sent to the active PDBA are also sent asynchronously to the standby PDBA after being successfully committed into the active PDB. This methodology allows for provisioning to be performed quickly from the PDBI client's point of view because the client receives the success message as soon as the update is committed to the active database. The client does not have to wait for the update to be forwarded across their WAN and replicated on the standby database.

This design contains an inherent short delay between the time the active PDB receives the update and when the standby PDB does. Because of this delay, clients only reading the database might be better off reading from the standby PDBA. It should also be noted that both PDBA clients must be up for the asynchronous replication to occur.

NOTE: The active/standby status of the two PDBA processes can be switched through a PDBI command or through the configuration user interface for the PDBA.

Also, the PDBA uses 5873 as its well-known listen port, although this value is modifiable through a command line argument.

It can be configured from which PDBA each RTDB receives its input. Due to the asynchronous nature of the PDBA replication, it is recommended that the RTDBs select the standby PDBA. This configuration ensures that there are no problems with differing levels if the active PDBA is stopped while there are many levels left to send to the standby PDBA. The RTDBs are guaranteed to always be on the PDBA that has the lower level number.

System Overview and Terminology

Figure 2-3 on page 2-9 shows a block diagram of the MPS/EPAP platform. It also shows a mated pair of EAGLE 5 SASs. The EAGLE 5 SASs are the large blocks at the bottom. The MPSs, which are attached to the EAGLE 5 SASs, are above the EAGLE 5 SASs and contain EPAP A and EPAP B.

An MPS system consists of two MPS servers and associated hardware, including a modem, CD-ROM, etc. Each EAGLE 5 SAS in a mated pair has one MPS system attached. The two MPS systems are referred to as a mated MPS system. Within one MPS system (i.e., the MPS system for one EAGLE 5 SAS), the two MPS servers are considered mated MPS servers and are referred to as MPS A (the upper server) and MPS B (the lower server).

The application bundle that runs G-Flex, G-Port, and INP is referred to as the EPAP. The EPAP consists of software applications needed to provision the databases, including the Provisioning database. That is the database referred to as the PDB. In terms of G-Flex, G-Port, and INP provisioning, the MPS upper and lower servers are called simply EPAP A and EPAP B or MPS A and MPS B.

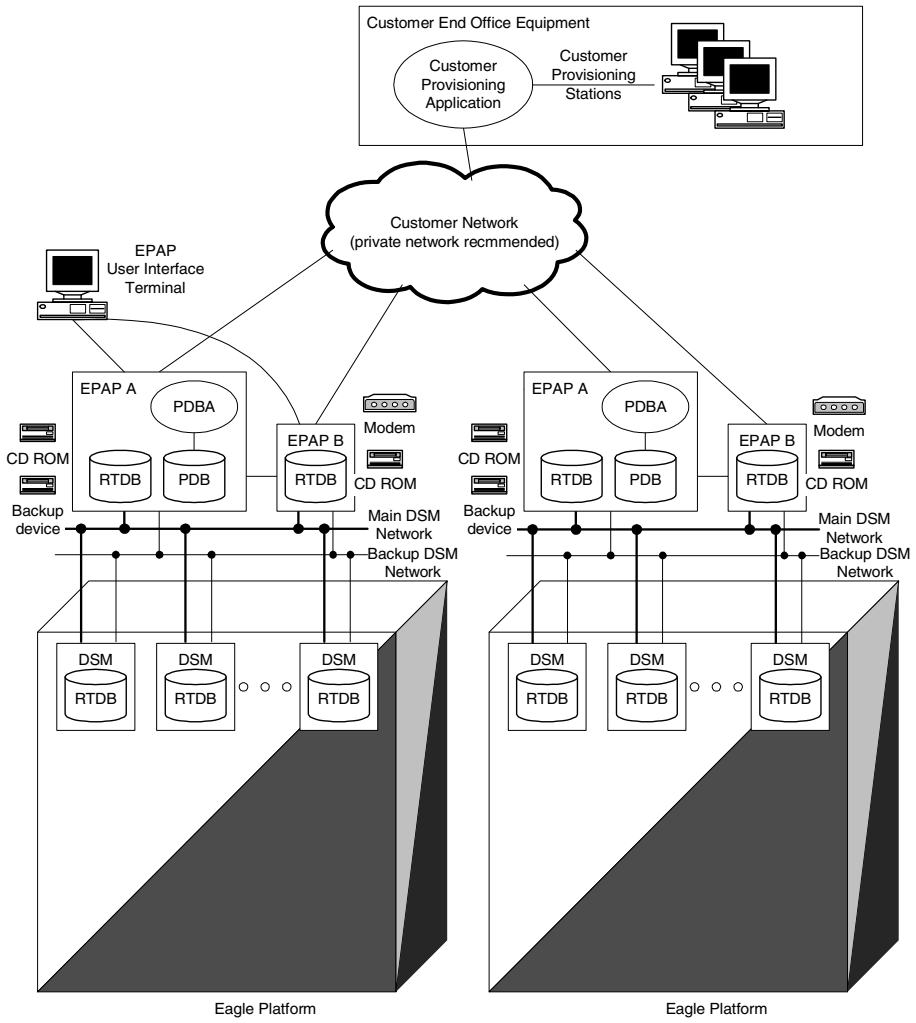
EPAP A and EPAP B are slightly different in their configuration. EPAP A runs the PDBA software and thus holds a copy of the PDB. This is the EPAP that is accessed using the PDBI. EPAP A also holds a copy of the RTDB for downloading to the DSM cards. EPAP B contains a redundant copy of the RTDB, but contains none of the PDBA software. This architecture is duplicated on the mated MPS system on the mated EAGLE 5 SAS. Typically the redundant EAGLE 5 SASs are called EAGLE 5 SAS A and EAGLE 5 SAS B.

The EPAPs are connected to the DSM cards via a 10/100 BASE-T ethernet for the downloading of the RTDB; these ethernet connections are called the main and backup DSM networks.

Network Connections

Connections and IP addressing for the customer (or provisioning) network, the main and DSM networks, the RTDB, and the Dialup PPP network are described in greater detail in the *EPAP Administration Manual*.

Figure 2-3. MPS/EPAP System Configuration



EPAP Status Reporting and Alarm Handling

Maintenance, measurements, status, and alarm information is routed from the Active EPAP to a primary DSM through EPAP Maintenance Blocks and DSM Status Requests.

The status reporting, message format, and various alarm messages are discussed in detail in the *EPAP Administration Manual*.

Provisioning Database Interface Description

This section describes the Provisioning Database Interface (PDBI) at a high level. The interface consists of the definition of provisioning messages only.

The customer must write a client application that uses the PDBI request/response messages to communicate with the PDBA. Details of the request/response messages appear in Chapter 3, *"PDBI Request/Response Messages."*

Socket Based Connection

The PDBI messages are sent across a TCP/IP socket. The client application is responsible for connecting to the PDBA well-known port and being able to send and receive the defined messages. It is also the responsibility of the customer's provisioning system to detect and deal with socket errors. Tekelec recommends that the TCP 'keepalive' interval on the customer's socket connection be set such that a socket disconnection problem is promptly detected and reported.

There is a limit to the number of PDBI connections; the default is 16 clients. If an attempt is made to connect more than the current client limit, a response is returned to the client: PDBI_TOO_MANY_CONNECTIONS. After the response is returned, the socket is automatically closed.

NOTE: Although the default limit is 16 PDBI connections, Tekelec is able to configure and support up to 128 connections. If you require more than 16 concurrent client connections, contact Tekelec for information.

String-Based Messages

The PDBI messages (requests and responses) are NULL-terminated strings. This has several benefits.

- It simplifies sending and receiving the messages from any language that has socket capability (for example, Perl or Java).
- It is easier for the PDBA to support any combination of the G-Flex, G-Port, and INP features at previous and new levels. Because the messages are not tied to C structures, differences between previous and new versions of the PDBI calls do not cause possible memory corruption. For example, if a new parameter is added to the `connect(…)` command, a client using the previous version of the command simply receives a parsing error. The same change in a C structure-based interface could result in the new C structure being filled in with wrong data.
- Because the messages are user readable, debugging errors in messages is easier.
- Messages can easily be stored in a request log for review or replay later.

Security

The PDBA maintains a list of IP addresses that are allowed to connect through the PDBI. Any connect request coming from an IP address that is not in the list is rejected. Each IP address in the list has either READ or READ/WRITE permission. IP addresses can be added to and removed from the list and permissions can be modified using the EPAP user interface PDBA menu items (refer to the PDBA Menu description in the *EPAP Administration Manual*.).

Transaction-Oriented API

The PDBI is a transaction-oriented API. This means that all subscription-related commands are sent within the context of a transaction. Two transaction modes are supported, normal and single.

Normal Transaction Mode

The normal transaction mode is the default method and has two main benefits:

- Many updates can be sent in a large transaction, and written to the database all at once when the transaction is completed. This results in a much faster rate of updates per second.
- It provides transaction integrity by allowing updates to be aborted or rolled back if there is an unexpected failure of some kind before the transaction is completed. Updates are not committed to the database until the **`end_txn`** command is issued. If an unexpected failure occurs or if the transaction is manually aborted, the database is maintained in the state before the start of the transaction (refer to “Command Atomicity” on page 2-12).

Single Transaction Mode

When sending a series of single-update transactions in normal transaction mode, considerable overhead is required for sending transaction boundary tags. Because some clients want to send only one update per transaction, an alternative PDBI connection type is available, called 'single transaction mode.'

When using this connection type, PDBI clients can send updates outside of the 'begin' and 'end' transaction delimiters. The PDB treats each single transaction mode update as being its own transaction. However, transaction delimiters are not ignored in 'single mode'. If the PDBI client issues these delimiters, the series of updates encapsulated by them are treated as one transaction, as they are been under the default normal transaction mode. For details on the PDBI connect options, refer to the “Connect” command on page 3-8 and the **`txnmode`** parameter.

Batch-Oriented/Bulk Load

The system can also accept batch files via FTP (File Transfer Protocol) or removable media (that is, MO, CD-R). The preferred method is FTP.

The format of the batch file looks like a series of normal PDBI commands, such as **`ent_sub`**, **`dlt_sub`**, etc. However, the connect/disconnect request and transaction begin/end commands are not included.

During a batch file import, transactions are handled in the same manner as with individual commands: the write transaction must be released by the client doing the batch file import before a new write transaction may be granted. Read transactions are always available, assuming the customer's interface network is available. The import file can contain as many commands as the storage media used to hold the batch file allows. The PDBA does not have a limit on the number of commands allowed.

The batch file is committed in stages; several transactions are opened to import the entire file. There is one commit for approximately every 200 entries. Therefore, it is impossible to rollback or abort a transaction after the import is complete. This also means that the dblevel returned at the end of the import may be increased by several levels since each transaction would increment it. The time needed to complete an import of a batch file depends upon several variables, including the size of the file.

Although the import is processed as a series of transactions, the write transaction is unavailable to other clients for the entire duration of the import, that is until all transactions related to the import have been processed.

Command Atomicity

Commands are atomic, that is, they are uninterruptible. Once a command is begun, it is performed completely or not at all; an atomic command cannot be partly performed or partly completed.

Consequently, if one command in a transaction fails, the results of that one command are not committed to the database upon execution of the **`end_txn`** command. However, all the other commands in the transaction that did execute successfully are committed upon execution of the **`end_txn`** command.

Provisioning Ranges of Subscriber Numbers

Currently, there is no method directly accessible from the PDBI for provisioning ranges of IMSIs (only individual IMSIs are supported), but provisioning MSISDNs Blocks and IMEIs Blocks is supported.

Transparency of Redundant Systems

The network operator is responsible for provisioning to only one PDBA. Once the active PDB is provisioned, the system automatically passes down the data to the active and standby RTDBs on that EAGLE 5 SAS. At the same time, the data is also passed, asynchronously, to the standby PDB and subsequently to the mated RTDBs on the mated EAGLE 5 SAS. Provisioning of redundant systems, therefore, is transparent to the user.

When the active PDBA becomes unavailable, the standby PDBA does not automatically switch to active. The PDBA client must send a switchover command to tell the standby PDBA to become active.

Logs

Several logs are available to the user, including a Command Log, which contains a trace of the commands sent to the PDBA, and an Error Log, which contains a trace of all errors encountered during provisioning, in addition to several other options.

Crash Recovery

If a crash occurs while a transaction is in process and does not cause database corruption, the database remains in the state before the crash after the reboot.

In the event of a catastrophic failure or corruption of a database, several options exist for reloading the data. For more information, refer to the *EPAP Administration Manual*.

Request IDs

Each request has an ID, called the '`iid`', as its first element. Its purpose is to allow responses to be matched up with requests as they arrive back at the client. Its value is an integer between 1 and 4294967295, expressed as a decimal number in ASCII.

The `iid` is optional. If an `iid` is not provided on a request, the corresponding response also does not have one. The `iid` is selected by the client when a command is sent and the selected value is returned by the PDBA in the subsequent response. A different `iid` could be selected for each request.

Multiple Session Connectivity

Multiple information systems can be connected via the PDBI simultaneously. All systems can open read transactions, but only one system at a time can open a write transaction. If more than one system requests a write transaction, contention for write access is handled as follows:

- The first user to submit a write request is granted access, if it is authorized for write access.

- If a second user submits a write request while the first transaction is still open, the second user is either immediately rejected or is queued for a specified timeout period.
- The time out period can be specified by the user in the write request as a value from 0 to 3600 seconds. If the value is not included or is set to 0, the second write request is immediately rejected.
- If the time out value is set to any non-zero value, the second request is held for that time period before being rejected. If the first user releases the write transaction before the second user time out period has expired, the second user is then granted write access.
- If a third user submits a write request after the second user with a specified time out period, the third user's request is queued behind the second user's request. When the first user releases the transaction, the second user is granted access. After the second user releases the transaction, the third user is granted access, and so forth. Of course, whenever any user's time out period expires, his/her request is rejected immediately.
- If the third user sets a time out period longer than the second user and the second user's time out period expires before the first user releases the transaction, the second user's request is dropped from the queue. The third user subsequently moves up in the queue. Thus, if the first user releases the transaction before the third user's time out has expired, the third user is granted access.

Request Queue Management

If multiple command requests are issued simultaneously, each request is queued and processed in the order it was received. The user is not required to wait for a response from one command before issuing another.

Incoming requests, whether multiple requests from a single user or requests from multiple users, are not prioritized. Multiple requests from a single user are handled on a first-in, first-out basis. Simply put, requests are answered in the order in which they are received. Servicing of requests from multiple users is dependent upon traffic in the data network.

Interface Configuration and Installation

In addition to this manual, additional information concerning PDBI installation, configuration and integration with the network operator's information system is provided in the *EPAP Administration Manual*.

File Formats

All file formats described in this section are text files.

The EPAP menu items for importing files to the PDB and exporting files from the PDB are described in the PDBA menu section of the *EPAP Administration Manual*.

Debug Log

The debug log format varies from process to process. Most entries contain a timestamp followed by a description of the logged event and some relevant data.

The EPAP menu for viewing the PDBA debug log is described in the PDBA menu section of the *EPAP Administration Manual*.

Import/Export Files

The Import/Export files use the PDBI **`create`** command format (see "Create Subscription" on page 3-15 and "Create Network Entity" on page 3-36). A carriage return separates each command in the file.

Import File

To achieve faster loading rates, large numbers of PDBI commands can be placed together in a file and loaded into the PDB through the Import option on the EPAP user interface. (For more information, refer to the *EPAP Administration Manual*.) The format of the commands in the file is exactly the same as the PDBI commands specified in this document. The valid commands to be placed in an import file are:

- **`ent_sub`**
- **`upd_sub`**
- **`dlt_sub`**
- **`ent_entity`**
- **`upd_entity`**
- **`dlt_entity`**
- **`ent_eir`**
- **`upd_eir`**
- **`dlt_eir`**

There is no need to place any other commands, such as **`begin_txn`**, in the file. If the PDBI user interface is used to send the import command, the user interface automatically handles establishing a connection with an open **`write`** transaction. Because the import operation has the **`write`** operation throughout its entire duration, normal updates from other PDBI users cannot obtain the write transaction until the import operation is finished.

Any errors encountered while processing the file are logged in the error log file of the PDBA. The processing of the import file continues. When the file is completely processed, the user interface displays a warning that errors were encountered. The error log file of the PDBA can then be viewed through the EPAP user interface. (For more information, refer to the *EPAP Administration Manual*.)

Commands in the import file are handled as though they were received across a normal PDBI connection. It is important that dependencies are listed in the file in the correct order. For example, if a DN is to be created and assigned to a specific NE (either SP/RN), that NE must exist before the DN can be created. The NE could either already exist in the database before the import file was sent, or it could be created in the import file before any DNs that need it.

Since there is limit to the number of commands that can be contained in a single transaction (see "Transaction Too Big Response" on page 3-6), the PDBA may have to break up the import into several separate transactions. This is handled internally in the PDBA. The user may notice only that the database level has grown by more than one.

Blank lines and lines beginning with the '#' character are skipped.

If any PDBI commands other than the six mentioned above are placed in an import file, each occurrence generates a BAD_IMPORT_CMD error internally while parsing the file. The total import error count is incremented, and the processing of the import file continues with the next line. The BAD_IMPORT_ CMD return code never actually is returned to the PDBI client, but it may be seen in the PDBA error log file.

Export File

It is possible to export the contents of the PDB to an ASCII file. Perform this through the Export option on the EPAP user interface. (For more information, refer to the *EPAP Administration Manual*.) The data can be formatted in two ways, either as PDBI commands or as raw delimited ASCII.

Currently, you must have the write transaction to export. This is to ensure that no updates are happening in the middle of the export. Therefore, an export can only occur on the active PDBA.

PDBI Format

Formatting the output as PDBI commands allows the resulting file to be used as an import file. The format of the commands in the file is exactly the same as the PDBI commands specified in this document.

Commands placed in the export file may not be the actual commands that originally created the instances. For example, if a DN was created originally on SP1 and subsequently updated to move to SP2, there would only be one command that creates the DN on SP2.

If the Number Prefix feature is turned on in the EPAP user interface, the generated PDBI commands follow the Number Prefix rules described in the Number Prefix section.

The file is ordered as follows:

1. Network Entities
2. IMSIs (with associated DNs if any exist)
3. Single DNs (that are not associated with any IMSI)
4. DN Blocks
5. IMEIs (with associated IMSIs)
6. IMEI blocks

Raw Delimited ASCII Format

Formatting the output as raw delimited ASCII creates a file that can easily be read by other client applications. The delimiter can be chosen on the EPAP user interface from a short list of possible delimiter types (for example, comma, pipe, space). The resulting file contains four separate sections.

Each section corresponds to a data type. The start of each section has a comment line (line starting with #) as its header. The content of the data lines depends on the section. In an effort to keep the resulting file as small as possible, fields whose values come from enum-like list of strings use only the first character of the choice.

Section 1. Network Entities

The first section in the file contains all of the Network Entities. The data on each line is similar to the data that can be provided on a **`ent_entity`** command.

`<ID>,<Type>,<PCType>,<PC>,<GC>,<RI>,<SSN>,<CCGT>,<NTT>,<NNAI>,`
`<NNP>, <DA>,<SRFIMSI>`

Where:

ID — Identifier for this Network Entity.

Values: 1 to 15 hexadecimal digits expressed using ASCII characters.

Type — Type of Network Entity.

Values: **`S`** - Signal Point

`R` - Routing Number

PCType — Specifies the type of the point code. The absence of a value in this field means that the NE did not have a point code.

Values: **i** - ITU international point code in the form zone-area-id (z-aaa-i).

n - ITU national point code in the form of ITU number (nnnnn).

a - ANSI point code in the form of network-cluster-member (nnn-ccc-mmm).

PC — The point code value. The valid values depend on the PCType parameter. If the PCType field did not have a value, then this field also does not have a value.

Values: For PCType of **i** (intl) the format is zone-area-id (z-aaa-i).

z = **0 - 7**

aaa = **0 - 255**

i = **0 - 7**

Note: The value 0-0-0 is not valid.

For PCType of **n** (natl) the format is number (nnnnn).

nnnnn= **0 - 16383**

For PCType of **a** (ANSI), the format is network-cluster-member (nnn-ccc-mmm).

nnn= **1 - 255**

ccc = **1 - 255** (if network = **1 - 5**)

= **0 - 255** (if network = **6 - 255**)

mmm= **0 - 255**

GC — (Optional) Group Code. This optional parameter is part of the point code value for ITU Duplicate Point Code Support feature.

Values: **aa - zz**

RI — Routing Indicator. This parameter indicates whether a subsequent global title translation is required.

Values: **G** = Global Title. Indicates that a subsequent translation is required.

S = Subsystem Number. Indicates that no further translation is required.

SSN — (Optional) New subsystem number. This parameter identifies the subsystem address that is to receive the message.

Values: **0**, **2 - 255**

CCGT — (Optional) Cancel Called Global Title.

Values: **y** or **n** (default)

NTT — (Optional) New translation type. This parameter identifies the translation type value to replace the received translation type value.

Values: **0 - 255**

NNAI — (Optional) New nature of address.

Values: **0 - 127**

NNP — (Optional) New numbering plan.

Values: **0 - 15**

DA — (Optional) Digit action. The parameter specifies what changes, if any, to apply to the Called Party GTA.

Values: **r** - Replace Called Party GTA with the entity id

p - Prefix Called Party GTA with the entity id

I - Insert Entity Id after country code

SRFIMSI — (Optional) The IMSI returned by a SRF indicating the Subscription Network of the subscriber. This parameter is only used by the G-Port features and only for RNs.

Values: a string with 5 to 15 characters where each character must be a number from **0** to **F**.

For example:

```
101010,s,a,2-2-2,g,100,,,,,r,
```

Section 2. IMSIs

The second section contains the IMSI data. For the raw delimited format, any DNs that an IMSI has are not listed with the IMSI. There is a field on the DN entry that points to the IMSI. This leaves only three pieces of data for the IMSI entries.

<IMSI>,<SP>

Where:

SP — (Optional) Specifies which SP the DN is on. The SP and RN fields do not both have values at the same time.

Values: 1 to 15 hexadecimal digits expressed using ASCII characters.

For example:

1234567890,101010

Since it is not possible to have an IMSI without an SP, at least one field must be populated.

Section 3. Single DNs

The third section in the export file contains the single DNs. The data in the entries is similar to the data in the **ent_sub** command.

<DN>,<IMSI>,<PT>,<SP>,<RN>

Where:

DN — A DN (specified in international format).

Values: a string with 5 to 15 characters where each character must be a number from **0** to **F**.

IMSI — The IMSI to which the DN is associated. This field does not have a value if the DN is not associated with any IMSI.

Values: a string with 5 to 15 characters where each character must be a number from **0** to **F**.

PT — (Optional) The portability type for the created DN. This field is only used by G-Port, IS-41 to GSM migration and PPSMS. For G-Port, it controls number Portability Status encoding in SRI acks. For IS-41 to GSM migration it identifies whether a subscriber has or has not migrated from IS-41 to GSM, (maintaining a single GSM handset). For PPSMS, it identifies a DN as one of two types needing PPSMS intercept.

Values: **0** – not known to be ported

1 – own number ported out (used by G-Port)

2 – foreign number ported to foreign network (used by G-Port)

3 – prepaid 1 (used by PPSMS)

4 – prepaid 2 (used by PPSMS)

5 – migrated with one handset (used by IS-41)

SP (Optional) Specifies which SP the DN is on. The SP and RN fields do not both have values at the same time.

Values: 1 to 15 hexadecimal digits expressed using ASCII characters.

RN (Optional) Specifies which RN the DN is on. The SP and RN fields do not both have values at the same time.

Values: 1 to 15 hexadecimal digits expressed using ASCII characters.

For example,

```
12345,1234567890,,101010
```

Section 4. DN Blocks

The last section in the export file contains the DN Blocks. The data in the entries are similar to the data in the **ent_sub** command.

```
<BDN>,<EDN>,<PT>,<SP>,<RN>
```

Where:

BDN The beginning DN (specified in international format).

Values: 5 to 15 hexadecimal digits expressed in the decimal format using ASCII characters.

EDN The ending DN (specified in international format).

Values: a string with 5 to 15 characters where each character must be a number from **0** to **F**.

PT (Optional) The portability type for the created DN. This field is only used by G-Port, IS-41 to GSM migration and PPSMS. For G-Port, it controls number Portability Status encoding in SRI acks. For IS-41 to GSM migration it identifies whether a subscriber has or has not migrated from IS-41 to GSM, (maintaining a single GSM handset). For PPSMS, it identifies a DN as one of two types needing PPSMS intercept.

Values: **0** – not known to be ported

1 – own number ported out (used by G-Port)

2 – foreign number ported to foreign network (used by G-Port)

3 – prepaid 1 (used by PPSMS)

4 – prepaid 2 (used by PPSMS)

5 – migrated with one handset (used by IS-41)

SP (Optional) Specifies which SP the DN Block is on. The SP and RN fields do not both have values at the same time.

Values: 1 to 15 hexadecimal digits expressed using ASCII characters.

RN (Optional) Specifies which RN the DN Block is on. The SP and RN fields do not both have values at the same time.

Values: 1 to 15 hexadecimal digits expressed using ASCII characters.

For example,

```
9195550000,919555ffff,0,,e1e10
```

Section 5. Single IMEIs

The 5th section in the export file contains the single IMEIs. The data in the entries are similar to the data in the **ent_eir** command. 8 IMSIs can be provided. If 8 IMSIs are not provisioned for that IMEI, then a NULL value is supplied.

```
<IMEI>,<SVN>,<WHITE>,<GRAY>,<BLACK>,<IMSI>,….,<IMSI>
```

Where:

IMEI Specifies the IMEI

Values: a string with 14 characters where each character is a number from **0** to **F**.

SVN (Optional) Specifies the Software Version Number

Values: A 2 digit number (0-99)

WHITE (Optional) Specifies a List Type of White

Values: yes/no

GRAY (Optional) Specifies a List Type of Gray

Values: yes/no

BLACK (Optional) Specifies a List Type of Black

Values: yes/no

IMSI — The IMSI to which the IMEI is associated. This field will not have a value if the IMEI is not associated with any IMSI.

Values: a string with 5 to 15 characters where each character must be a number from **0** to **F**.

Section 6. IMEI Blocks

The 6th section in the export file contains the block IMEIs. The data in the entries are similar to the data in the **`ent_eir`** command.

`<BIMEI>,<EIMI>,<WHITE>,<GRAY>,<BLACK>`

Where:

BIMEI — Specifies the beginning of the IMEI block

Values: a string with 14 characters where each character is a number from **0** to **F**.

EIMEI — Specifies the ending of the IMEI block

Values: a string with 14 characters where each character is a number from **0** to **F**.

WHITE — (Optional) Specifies a List Type of White

Values: yes/no

GRAY — (Optional) Specifies a List Type of Gray

Values: yes/no

BLACK — (Optional) Specifies a List Type of Black

Values: yes/no

3

PDBI Request/Response Messages

Overview 3–2
- Message Definitions 3–2
- Request IDs 3–3
- Optional Parameters 3–3
- Common Response Format 3–3
- Number Prefixes 3–4
- Common Responses 3–4
- Common Response Messages 3–5

Messages 3–8
- Connect 3–8
- Disconnect 3–10
- Begin Transaction 3–11
- End Transaction 3–13
- Abort Transaction 3–14
- Create Subscription 3–15
- Update Subscription 3–22
- Delete Subscription 3–28
- Retrieve Subscription Data 3–30
- Create Network Entity 3–36

Update Network Entity 3–40
Delete Network Entity 3–44
Retrieve Network Entity 3–46
Switchover 3–48
PDBA Status Query 3–51
Dump Connections 3–52
Create IMEI Data 3–53
Update IMEI Data 3–58
Delete IMEI Data 3–61
Retrieve IMEI Data 3–64
Request DSM Report 3–68

Overview

This chapter defines the Provisioning Database Interface (PDBI) request and response messages. The messages are listed in alphabetical order.

Provisioning clients connect to the EAGLE Provisioning Application Processors (EPAPs) through the PDBI. The PDBI consists of commands and their parameters, which allow you to define the messages that provision the G-Flex, G-Port, and INP features and allow the retrieval of feature data.

PDBI messages are sent across a TCP/IP socket. The client application (defined by the customer) is responsible for connecting to the Provisioning Database Application (PDBA) well-known port and being able to send and receive the defined messages.

NOTE: The customer must write his own client application that uses the PDBI to communicate with the PDBA.

PDBI messages (requests and responses) are NULL-terminated strings, which allows sending and receiving the messages from any language that has socket capability (for example, Perl or Java).

Message Definitions

Each message definition consists of one request and one or more responses. A request is a message sent by the client to the client application to invoke a service. A response is a message returned to the client by the client application to confirm that the a requested service has been invoked, the transaction has been completed, or a connection has been established.

Request IDs

Each request has an integer identification (**iid**) as its first element. The client can use the **iid** to match returned responses with the original requests The integer is expressed as a decimal number in ASCII and has a range from 1 to 4294967295. The **iid** is optional. If an **iid** is not provided on a request, the corresponding response also does not have one.

Optional Parameters

Optional parameters are surrounded by square brackets [ ] in the syntax examples. If you want to omit an optional parameter from the request command, omit the entire field including the label, value, and following comma. Do not leave a comma in as a place holder. The parameter labels in the fields that are sent on the request provide enough information to determine which parameters were omitted. However, the field labels must be present on all specified parameters.

For example, examine the following syntax:

```
sample_msg(field1 #, field2 #, [field3 <yes/no>], field4
<0..255>)
```

If you want to omit the **field3** parameter of a request, you might enter the request command using the following syntax:

```
sample_msg(field1 123, field2 456, field4 128)
```

Common Response Format

Responses use the same basic format.

If an integer identification (**iid**) was provided in the request, the response **iid** corresponds to the **iid** of the original request. A return code indicates either success (zero) or failure (non-zero). See Appendix A, “PDBI Message Error Codes,” for a mapping between the return code labels described in this section and the real integer value.

Additionally, an optional **data** element returns request-specific return information.

Each defined response declares the errors it returns (with their meanings) and what the data section should look like for each error. If a response does not require a data section (meaning it is just a simple ACK or NAK), the data section does not appear at all. In that case, the last item in the response is the return code (**rc**).

The following example shows the syntax of the common response format. This format applies to all response messages described in this chapter unless stated otherwise.

```
rsp ([iid <iid from request>,] rc <return code>, [data (. . .)])
```

The format of each command response is shown in this chapter. The response information for each command is described in detail in the *Commands Manual*.

Number Prefixes

The PDBA has the concept of default number prefixes. These are PDBA parameters that are configurable from the *EPAP Administration Manual*. There are two number prefixes, one for DNs and DN Blocks and the other for IMSIs. They are completely separate and can be set or not set independently. When set, the number prefix values are automatically prepended to all DNs and DN Blocks or IMSIs (depending on the prefix type) in PDBI requests. The values are also stripped off of the DNs, DN Blocks and IMSIs in PDBI responses.

For example, if the DN Prefix is set to "34" in the UI and then an **`ent_sub`** request is sent to create DN 12345, the actual DN stored in the database and sent to the EAGLE 5 SAS is "3412345". If a PDBI query is done for DN "12345" while the number prefix is still "34", the "3412345" is found in the database, but only the DN value "12345" is returned in the PDBI response.

It is possible to override a default number prefix. The symbol '#' at the beginning of a DN, DN Block, or IMSI means that it is the actual value and that no number prefix should be applied. This can occur in both requests and responses.

For example, if the PDBI client sends a value "#12345" in a request, it means that he literally means the value "12345", not "3412345" (assuming that "34" is the that type's number prefix). If a PDBI response comes back with a "#12345", it means that the DN, DN Block or IMSI literally had the value "12345", not "3412345" (still assuming that "34" is the Number Prefix). A response with a "#" value is returned if a DN, DN Block or IMSI is found in the database that did not match its type's number prefix.

It is important to note that the "#" number prefix override is only valid for DNs, DN Blocks, and IMSIs. The "#" symbol at the beginning of any other parameter value does not parse.

Since the number prefix and the number prefix override apply to all requests and responses that have DNs, DN Blocks or IMSIs, it is not mentioned on each command separately. It is described only here.

Common Responses

The response code examples given for each message indicate those codes that are specific for that message. Other response codes may apply, such as the more general error responses like PDBI_NOT_CONNECTED, PDBI_NO_ACTIVE_TXN, PDBI_NOT_FOUND, PDBI_BAD_ARGS. These are not repeated for each message for simplicity.

No command can be issued until a connection has been established by issuing the connect request to a PDBA. This restriction includes data provisioning commands such as **`ent_sub`**, **`rtrv_sub`**, etc., as well as query commands such as **`status`**, **`dump_conn`**, etc.

Common Response Messages

Because the PDBI is a string-based API, all requests can return a Parse Failed response message or a Bad Argument response message.

Parse Failed Response

The Parse Failed response message is identified by return code PARSE_FAILED. This response message indicates a syntactical problem with the command received and can have a data section present to provide more information about the parse failure. Table 3-1 lists possible reasons for parse failures.

If the data section exists, two optional parameters are possible. The first parameter is a reason text string stating explicitly what was wrong with the request. The second parameter is a location string containing the place where the error occurs and, surrounded by curly braces, the portion of the original request that contained the error. If no specific information is available, the data section is not present in the response.

The following example shows the syntax of a Parse Failed response message:

```
data ([reason "Missing comma"], [location "XXXXXXX{}
dn XXXXXXXXX"])
```

Table 3-1. Parse Failure Reasons

Reason	Description
Unknown request verb	The request verb did not match any of the known commands.
Space required	A white space character was missing after some element of the request.
Missing paren	An opening or closing parenthesis was missing.
Invalid value	An invalid value was provided for one of the parameters.
<name of parameter> parameter expected	Some mandatory parameter was missing.
Multiple <name of parameter> found	Multiple occurrences of a parameter were found that does not allow multiple occurrences.
Unknown parameter	An unknown parameter was found.
Missing comma	A comma was missing after a parameter.
Value expected	A parameter label was found with no value following it.
Duplicate parameter	A parameter that should have occurred only once was found more than once.

Table 3-1. Parse Failure Reasons (Continued)

Reason	Description
Numeric value too large	The value specified for a numeric parameter specified a number greater than the maximum integer.

Bad Arguments Response

The Bad Arguments response message is identified by return code BAD_ARGS. This response message indicates a semantic problem with the command received (for example, missing mandatory parameters or invalid parameter combinations). The data section of a Bad Arguments response message has a reason string that indicates what problem was encountered.

The following example shows the syntax of a Bad Arguments response message:

```
data (reason "No version provided")
```

Transaction Too Big Response

The internal EAGLE 5 SAS RTDB imposes a transaction size limit on the PDBA. In order to ensure that the PDBA and the EAGLE 5 SAS databases are truly equivalent, this limit must be propagated by the PDBA onto the PDBI clients. As a result, all database changing commands that occur within a write transaction have the potential to fail with a TXN_TOO_BIG error.

The transaction size limit is 200. It limits the number of modifications to the EAGLE 5 SAS database. The limit is 200 EAGLE 5 SAS RTDB updates. Unfortunately, this may not have a one to one correlation to the PDBI update commands. This is because a single PDBI command can result in several changes to the underlying database.

For example, a single PDBI command **`ent_sub`**, which contains IMSI 12345, DN 67890, DN 67891, and SP 101010, is performed by two EAGLE 5 SAS database commands, one for the IMSI and one for the DNs. The worst case number of EAGLE 5 SAS database commands that can occur due to one PDBI command is nine if the **`force`** parameter is not used. If the **`force`** parameter is set to **`yes`**, the highest possible number of EAGLE 5 SAS database commands in a single PDBI command is 17.

Multiple Segmented Responses

For some responses, it is possible that all of the data cannot be returned in one response. In this case, multiple responses for the same request are returned. The first through (N-1)th response have a return code of PARTIAL_SUCCESS to indicate that there should be more following them. The Nth response has the return code SUCCESS to indicate that it is the final response. Multiple responses also use the segment parameter at the beginning of the data section to allow the client to know that no responses have been missed. The segment parameter value

starts at one for the first response and is incremented by 1 in each subsequent response for that request up to and including the final response that contains the SUCCESS return code. For consistency, the segment parameter is also present in single message responses with the value of 1.

Errors Not Returned to Client

Two return codes are not returned to a PDBI client. They are PDBI_INTERRUPTED and PDBI_UNIMPLEMENTED.

- PDBI_INTERRUPTED is used internally to cancel requests that are in progress if the PDBI client abnormally disconnects. Since the return value is only used when the connection is broken, obviously the return code cannot be returned to the client.
- PDBI_UNIMPLEMENTED is used during development of new features and commands to allow a valid return from commands that have been defined but are not implemented yet. Since the PDBA currently implements all of the commands described in this specification, the return code cannot be returned to the client.

DSM Report

The PDBA keeps track of the status of the DSM cards that it has connectivity to in the customer's network. Each card reports its information to the PDBA at regular intervals. The PDBA makes this information available to the PDBI clients in a DSM Report. The DSM Report can be requested by the client in several ways. These ways are spelled out in various commands that actually do the requesting. In all cases, the content and structure of the DSM Report is the same. The intent of the DSM Report is to inform the receiver what percentage of DSM cards are at a specific database level. This information can be used by the client to determine when enough DSM cards have a specific update to consider it safe for traffic.

The report includes the database level being reported on, the percentage of DSM cards that have that level, and the total number of known DSM cards. Also included is a list of all DSM cards whose level did not meet or exceed the mentioned level. For each card in this list, the report provides the CLLI, card location, database status, and database level. If the database status is "loading", a percent loaded status is shown.

The client can either receive this report as a response to the rtrv_dsmrpt request, or it may be periodically received asynchronously if the client specifies the appropriate connect parameters. When the report is sent as a response to a normal synchronous request, the message begins with the normal rsp(…) label. However, when the report is sent as an asynchronous message, it begins with dsmrpt(…) to help identify that this is not a response to any recent request sent.

Messages

The messages described in this section are defined by commands and their parameters defined in the PDBI.

Connect

After a client has established a socket connection with the PDBA on its well-known port (5873), the client must issue a Connect request message. The PDBA returns a response message that indicates whether it is capable of accepting a new connection. There is a limit to the number of PDBI connections; the default is 16 clients. The `connect` command defines the Connect message.

If an attempt is made to connect more than the current client limit, a response is returned to the client: PDBI_TOO_MANY_CONNECTIONS. After the response is returned, the socket is automatically closed.

NOTE: Although the default limit is 16 PDBI connections, Tekelec is able to configure and support up to 128 connections. If more than 16 concurrent client connections are required, contact "Customer Contact Center" on page 1-9 for more information.

Request

The Connect request message is issued by the client to request a connection to the PDBA.

Parameters:

version (Mandatory) Informs the PDBA of the API version this client application knows. This parameter decides whether or not the client application can successfully communicate with the PDBA.

Values: **1.0**

rspsize (Optional) Allows the client to specify the maximum size in Kilobytes of responses that the PDBA sends back. This parameter ensures that retrieve requests that result in a large number of instances being returned come back in manageable-sized responses to the client.

Values: **1 – 32** (default = **4**)

switchactn (Optional) Allows the client to specify what action is to be taken if the Active or Standby status of the PDBA changes as the result of a Switchover request.

Values: **none** – No action taken (default)

close – Terminate this connection by closing the socket

endchar (Optional) Allows the client to specify what character the PDBA uses to terminate responses.

Values: **null** – Responses terminated with a NULL (\0) character (default)

newline – Responses terminated with a newline (\n) character

idletimeout (Optional) Allows the client to specify the number of minutes that the connection can remain idle before the PDBA should terminate it.

Values: **none** – Connection is never terminated by PDBA for idleness (default)

1 – 44640 – Terminate this connection after this many idle minutes.

txnmode (Optional) Transaction mode allows the client to specify whether this connection operates in single transaction or normal transaction mode. This selection determines whether update requests can be sent individually or require the use of **`begin_txn`** and **`end_txn`** boundaries, to be considered write transactions on their own and allowed.

Values: **normal** – All updates must be specified inside **`begin_txn`** and **`end_txn`** boundaries, which is the 'normal transaction mode.' (default)

single – Individual update requests are their own transactions. The use of **`begin_txn`** and **`end_txn`** boundary commands is not required in single transaction mode.

dsmrpt (Optional) Allows the client to specify whether or not it wants to receive the asynchronous DSM Report messages.

Values: **no** – The DSM Reports are not wanted (default).

yes – The DSM Reports are wanted.

dsmrptperc(Optional) Allows the client to specify what percent to use in the DSM Report. This overrides the system wide default DSM Report percent value for this one connection.

Values: **1 – 100**

dsmrptfreq(Optional) Allows the client to specify how often, in seconds, that the connection wants to receive the DSM Report. This overrides the system wide default DSM Report frequency value for this one connection. This parameter is only meaningful if dsmrpt has been specified as yes.

Values: **1 – 86400 –** Send the DSM Report in this many seconds.

Request syntax:

```
connect([iid XXXXX,] version 1.0, [rspsize <1..32>],
[switchactn <none/close>],[endchar <null/newline>],
[idletimeout <none/1..44640>], [txnmode <normal/single>],
[dsmrpt <no/yes>], [dsmrptperc <1..100>],
[dsmrptfreq <1..86400>])
```

Response

The Connect response message indicates whether or not the PDBA is capable and willing to accept a new connection. If the connection is accepted, the data section in the response indicates the connection ID (**`iid`**) that was assigned and whether the PDBA connected to was running as the Active or Standby PDBA.

The return codes listed in Table 3-2 indicate the result of the Connect request.

Table 3-2. Connect Response Return Codes

Return Code	Description	Data Section Contents
SUCCESS	Everything worked.	The assigned connection id and Active status are returned. **`data (connectId ########, side active/standby)`**
UNKNOWN_ VERSION	The specified version is not known or not supported.	NONE
INVALID_ VALUE	One of the fields specified had an invalid value.	The offending field is returned in the data section: **`data (param <field label>)`**
ALREADY_ CONNECTED	A connect request was sent on a socket that already had an established connection.	NONE

Disconnect

The Disconnect message is required to disconnect from the PDBA. The **`disconnect`** command defines the Disconnect message.

Request

The Disconnect request message is issued by the client to request a disconnect from the PDBA. This request tells the PDBA that the client has finished and allows the PDBA to clean up any connection-related data. If the client has a

transaction open, the transaction is automatically aborted and any updates in the transaction are backed out. All Disconnect requests result in the connection being broken.

NOTE: The PDBA behavior is the same if the client neglects to send this request and just closes the socket, or if the client abnormally terminates and the operating system closes the socket.

Request syntax:

```
disconnect([iid XXXXX])
```

Response

The Disconnect response message indicates either that the disconnect was successful without problems or that the disconnect was achieved through aborting a still-active transaction. Aborting an active transaction can occur because there were issues on the PDBA while cleaning up.

The return codes listed in Table 3-3 indicate the result of the Disconnect request.

Table 3-3. Disconnect Response Return Codes

Return Code	Description	Data Section Contents
SUCCESS	Everything worked.	NONE
ACTIVE_TXN	There was a transaction still active. It was aborted.	NONE

Begin Transaction

The Begin Transaction message starts a **read** or **write** transaction, which is required for all data-related commands (to create, update, delete, or retrieve subscriptions). A client connection can only have one transaction open at a time. The **`begin_txn`** command defines the Begin Transaction message.

The following commands are not required to be issued from within a transaction: **`switchover`**, **`status`**, **`dump_conn`**.

Request

By opening a **read** transaction, the client indicates to the PDBA that only data querying requests are sent; no database-changing requests (create, update, or delete) are sent. Any database-changing requests sent in a **read** transaction return a failure. Multiple client applications can have **read** transactions open at the same time. Responses from querying requests are sent back to the client immediately. There is no need to end the **read** transaction until you are through sending requests. **Read** transactions can be sent to either the Active or Standby PDBAs.

Take care when opening a read transaction on the standby PDBA. While two PDBAs are communicating normally, the data on the standby is valid. However, if the connection between the two PDBAs is broken and they cannot communicate, the information contained on the standby PDBA does not contain the new updates written to the active PDBA while the connection was broken. Thus in this case, data obtained from a read transaction on the standby PDBA would not be current and accurate information.

When the connection is re-established, the standby PDB is automatically re-synched to the current level of the active PDB. It is possible to achieve greater performance by sending read transactions to the standby PDB and write transactions to the active PDB. However, the precautions noted above should be considered.

By opening a **write** transaction, the client informs the PDBA that the database is updated in some way. After opening a **write** transaction, the client can send database-changing requests. Each command is evaluated for validity and cached locally.

NOTE: The commands are not saved in the database or sent to the RTDB until the write transaction is ended.

The commands within the transaction can also be aborted (or rolled back) with an **`abort_txn`** command any time before the transaction is ended with the **`end_txn`** command. Only one client is allowed to open a **write** transaction at a time. **Write** transactions can be opened only on the Active PDBA. Attempts to open a **write** transaction on the Standby PDBA result in an error response.

It is possible for a client to make querying requests inside a **write** transaction. In this case, it is important for the client to remember that the data returned can reflect any updates that the **write** transaction has made so far but not yet committed. If the **write** transaction is aborted, the data retrieved from the query might no longer be valid.

The **`begin_txn`** command defines the Begin Transaction request message.

Parameters:

type (Mandatory) Type of transaction to open.

Values: **read** or **write**

timeout (Optional) How many seconds to wait for the **write** transaction if another connection already has it.

Values: **0** (return immediately if not available; default)
1 - **3600** seconds

Request syntax:

```
begin_txn([iid XXXXX,] type <read|write>, [timeout <0..3600>])
```

Response

The return codes in Table 3-4 indicate the result of the Begin Transaction request.

Table 3-4. Begin Transaction Response Return Codes

Return Code	Description	Data Section Contents
SUCCESS	Everything worked.	NONE
INVALID_ VALUE	One of the fields specified had an invalid value.	The offending field is returned in the data section: **`data (param <field label>)`**
NO_WRITE_ PERMISSION	The PDBI client making the connection does not have WRITE access permissions.	NONE
WRITE_ UNAVAIL	Another client already has a **write** transaction open. This is returned only to clients who have WRITE access permissions. Clients who have only READ access receive NO_WRITE_PERMISSION even when another **write** transaction is open.	The IP address information of the client that already has the write transaction. **`data (id <connection id>, ip <ip addr>, port <port num>)`**
ACTIVE_TXN	A **read** or **write** transaction is already open on this connection.	NONE
STANDBY_ SIDE	An attempt to open a **write** transaction occurred on the Standby PDBA.	NONE

End Transaction

The End Transaction message completes a **read** or **write** transaction. The behavior depends on whether the active transaction was a **`read`** or **`write`** transaction. The **`end_txn`** command defines the End Transaction message.

Request

For a **read** transaction, the End Transaction request message informs the PDBA that it is done making queries. There are no database commitment.

For a **write** transaction that had successful updates, the End Transaction request message causes the database changes to be committed and sent to the RTDB. The new database level is returned in the data section of the response. The updates are not committed to the PDB until the **`end_txn`** command is received.

If none of the updates was successful, a NO_UPDATES code is returned, and the dblevel does not change. If any one of the commands was successful, a SUCCESS code is returned, and the dblevel is incremented. Note that the dblevel is incremented to the same value following a transaction with successful updates regardless of whether all updates were successful or only one.

The dblevel indicates the database level of the destination after the database action has occurred. It is incremented after every write transaction. The level is incremented by one after each successful write transaction, regardless of how many commands are sent in the transaction or whether the commands are **`create`**s or **`delete`**s. This value is used by the DSM cards to check consistency with the RTDB.

Request syntax:

```
end_txn([iid XXXXX])
```

Response

The End Transaction response message signals that the database update is done. This response does not imply anything about whether or not the updates have made it to the RTDB yet. If the response contains the SUCCESS return code, then the update was successfully committed in the PDB. If any failure response is returned, the database commit failed. The **`end_txn`** request causes the transaction to end regardless of whether any updates were actually made to the PDB.

The return codes listed in Table 3-5 indicate the result of the End Transaction request.

Table 3-5. End Transaction Response Return Codes

Return Code	Description	Data Section Contents
SUCCESS	Database update was successful.	If the transaction type was **write**, the new database level is returned. **`data (dblevel #######)`**
NO_ ACTIVE_TXN	There was no currently active transaction for this connection.	NONE
NO_ UPDATES	The **write** transaction had no successful updates. No database change occurs, and no new database level is returned.	NONE
DB_EXCEPTION	An unexpected exception was thrown during the database commit. The entire transaction was rolled back to ensure predicable behavior. Contact Tekelec.	NONE

Abort Transaction

The Abort Transaction message aborts a currently executing **read** or **write** transaction. If the transaction was a **`read`** transaction, the transaction is simply closed. The **`abort_txn`** command defines the Abort Transaction message.

Request

This request aborts the currently executing transaction. If the current transaction is a **write** transaction, any updates are rolled back.

NOTE: Sending an abort transaction request while receiving responses from a query request does not cause the query responses to stop.

Request syntax:

```
abort_txn([iid XXXXX])
```

Response

The return codes listed in Table 3-6 indicates the result of the Abort Transaction request.

Table 3-6. Abort Transaction Response Return Code

Return Code	Description	Data Section Contents
SUCCESS	Abort successful.	NONE
NO_ACTIVE_ TXN	There was no currently active transaction for this connection.	NONE

Create Subscription

Create Subscription messages define different combinations of subscriptions by using the **ent_sub** command with a different set of parameters. The following subscriptions can be created:

- Subscription containing a single IMSI with no DNs
- Subscription containing an IMSI and one to eight DNs
- One or more DNs on the same NE with no IMSI
- Subscription porting a block of DNs

Request

Subscription Containing a Single IMSI with no DNs

This command attempts to create an IMSI record that contains no DNs. By default, if the IMSI already exists, the command is rejected. Using the optional **force** parameter changes the default behavior to overwrite an existing IMSI. If the existing IMSI that is overwritten has DNs, those DNs are deleted.

NOTE: Only the G-Flex feature uses this type of subscription data.

The **ent_sub** command defines the request message for a subscription containing a single IMSI with no DNs.

Parameters:

imsi A single IMSI.

Values: 5 to 15 hexadecimal digits expressed using ASCII characters.

sp Specifies which SP the IMSI is on. The **sp** must correspond to an existing SP entity.

Values: 1 to 15 hexadecimal digits expressed using ASCII characters.

force (Optional) Indicates whether the client wants existing instances to be overwritten.

Values: **yes** or **no** (default = **no**).

timeout (Optional) Specify the number of seconds to wait for the write transaction if another connection already has it. Clients waiting for the write transaction with this mechanism are processed in the order that their requests were received. This option is only allowed if the client used the txnmode single option on its connect request.

Values: **0** (return immediately if not available; default)
1 - **3600** seconds

Rules:

1. The **sp** parameter must be specified.

Request syntax:

```
ent_sub([iid XXXXX,] imsi XXXXX [, sp XXXXX]
[, force yes/no] [, timeout <0..3600>])
```

Subscription Containing an IMSI and One to Eight DNs

This command attempts to create a subscription with one IMSI and up to eight DNs. If the IMSI already exists and none of the DNs specified in the request exists as a stand-alone DN or on another IMSI, the request adds the specified DNs to the existing IMSI. If the number of DNs currently existing on the IMSI and the number of DNs specified in the request total more than eight, the request is rejected. If any of the DNs in the request match a DN already existing on the specified IMSI, it is not counted twice toward the eight-DN limit.

The optional **force** parameter allows the client to change the default behavior and overwrite existing entries. If the IMSI already existed, it is deleted and recreated with the data in the request. This means that if the existing IMSI had DNs, those DNs are also deleted. If any of the DNs specified in the request already exist, those existing DNs are changed to point to the new IMSI and removed from the existing IMSI. If removing the DNs results in the previous IMSI having no DNs, the IMSI with no DN is not deleted.

NOTE: This type of subscription data is used only by the G-Flex feature.

The **ent_sub** command defines the request message for a subscription containing one IMSI and one to eight DNs.

Parameters:

imsi A single IMSI.

Values: 5 to 15 hexadecimal digits expressed using ASCII characters.

dn A DN (specified in international format) to be associated with the specified IMSI. There can be up to eight DNs specified.

Values: 5 to 15 hexadecimal digits expressed using ASCII characters.

sp Specifies which SP the IMSI and DNs are on. The **sp** must correspond to an existing SP entity.

Values: 1 to 15 hexadecimal digits expressed using ASCII characters.

force (Optional) Indicates whether the client wants existing instances to be overwritten.

Values: **yes** or **no** (default = **no**)

timeout (Optional) Specify the number of seconds to wait for the write transaction if another connection already has it. Clients waiting for the write transaction with this mechanism are processed in the order that their requests were received. This option is only allowed if the client used the txnmode single option on its connect request.

Values: **0** (return immediately if not available; default)
1 - **3600** seconds

Rules:

1. The **sp** parameter must be specified.

Request syntax:

```
ent_sub([iid XXXXX,] imsi XXXXX, dn XXXXX, …, dn XXXXX
[, sp XXXXX,] [, force yes/no]
[, timeout <0..3600>])
```

One or more DNs on the Same NE with no IMSI

This command attempts to create up to eight single DNs without associating any of them with an IMSI. They are all stand-alone DNs. Specifying more than one DN per request is only for performance reasons. When the request is complete, there is no relationship between them.

By default, if any of the specified DNs conflicts with an existing single DN, the entire command is rejected (including the DNs without conflict). The optional **force** parameter allows you to change the default behavior to overwrite existing DNs.

Stand-alone DNs might or might not be associated with a network entity, but they cannot be associated with both an SP and an RN at the same time.

If the newly created DN falls in the middle of an existing DN block, the new DN is considered to be an exception to the block. The block is still kept intact; it is not split into separate blocks around the new single DN.

The portability type (**pt**) parameter also defines the prepaid type. The prepaid type (portability type value of **3** or **4**) determines which IN platform the short message is directed to. The **pt** parameter can be specified only for DNs; it cannot be specified when an IMSI is in the command.

For example, if a single **ent_sub** request specifies both an IMSI and a DN, you cannot specify **pt**. However, you can use the **upd_sub** request to add a **pt** to the DN. To add a DN and **pt** in one request, the **ent_sub** must not specify an IMSI.

The **pt** values are mutually exclusive, that is, a single subscription cannot have simultaneously value **1** (ported out) and also **3** (prepaid 1). However, there is no effect on the G-Port MNP function if **pt** type **3** or **4** is specified. In these cases, if a message is being processed for G-Port MNP and the DN matches with a **pt** type **3** or **4**, G-Port considers it the same as if the **pt** type is **none**.

The **ent_sub** command defines the request message for one or more DNs on the same NE with no IMSI.

Parameters:

dn A DN (specified in international format). There can be up to eight DNs specified.

Values: 5 to 15 hexadecimal digits expressed using ASCII characters.

pt (Optional) The portability type for the created DN. This field is only used by G-Port, IS-41 to GSM migration and PPSMS. For G-Port, it controls number Portability Status encoding in SRI acks. For IS-41 to GSM migration it identifies whether a subscriber has or has not migrated from IS-41 to GSM, (maintaining a single GSM handset). For PPSMS, it identifies a DN as one of two types needing PPSMS intercept.

Values: **0** – not known to be ported

1 – own number ported out (used by G-Port)

2 – foreign number ported to foreign network (used by G-Port)

3 – prepaid 1 (used by PPSMS)

4 – prepaid 2 (used by PPSMS)

5 – migrated with one handset (used by IS-41)

none – no status (default = none)

sp (Optional) Specifies which SP the DN(s) are on. The **sp** must correspond to an existing SP entity. Most INP-only customers do not need to use SP.

Values: 1 to 15 hexadecimal digits expressed using ASCII characters.

rn (Optional) Specifies which RN the DNs are on. If the requested RN does not already exist, a blank one with no values is automatically created. G-Flex-only customers should not use RNs.

Values: 1 to 15 hexadecimal digits expressed using ASCII characters.

force (Optional) Indicates whether the client wants existing instances to be overwritten.

Values: **yes** or **no** (default = **no**)

timeout (Optional) Specify the number of seconds to wait for the write transaction if another connection already has it. Clients waiting for the write transaction with this mechanism are processed in the order that their requests were received. This option is only allowed if the client used the txnmode single option on its connect request.

Values: **0** (return immediately if not available; default)
1 - **3600** seconds

Rules:

1. It is not valid to specify both **sp** and **rn**.

Request syntax:

```
ent_sub([iid XXXXX,] dn XXXXX, . . ., dn XXXXX,
[pt <0/1/2/3/4/5/none>,] [, sp XXXXX,] [rn XXXXX,]
[force yes/no,] [timeout <0..3600>])
```

Subscription Porting a Block of DNs

This command attempts to create a new DN block. By default, if the new DN block conflicts with any part of an existing DN block, the command is rejected. The optional **force** parameter allows you to change the default behavior to overwrite an existing DN block if its beginning and ending DNs *exactly* match the block specified in the request.

DN blocks might or might not be associated with a network entity, but they cannot be associated with *both* an SP and an RN at the same time.

The **pt** parameter is used to define the prepaid type. The prepaid type (value of **3** or **4**) determines which IN platform the short message is directed to.

The **pt** values are mutually exclusive, that is, a single subscription cannot have simultaneously a value '1' (ported out) and also '3' (prepaid 1). However, there is no effect on the G-Port MNP function if **pt** type **3** or **4** is specified. In these cases, if a message is being processed for G-Port MNP and the DN block matches with a **pt** type **3** or **4**, G-Port considers it the same as if the **pt** type = **none**.

This command is used only in the G-Port (and by extension PP SMS) and INP features; the portability type parameter applies only to G-Port and PP SMS.

The **ent_sub** command defines the request message for a block of DNs.

Parameters:

bdn The beginning DN (specified in international format).

Values: 5 to 15 hexadecimal digits expressed using ASCII characters.

edn The ending DN (specified in international format).

Values: 5 to 15 hexadecimal digits expressed using ASCII characters.

pt (Optional) The portability type for the created DN. This field is only used by G-Port, IS-41 to GSM migration and PPSMS. For G-Port, it controls number Portability Status encoding in SRI acks. For IS-41 to GSM migration it identifies whether a subscriber has or has not migrated from IS-41 to GSM, (maintaining a single GSM handset). For PPSMS, it identifies a DN as one of two types needing PPSMS intercept.

Values: **0** – not known to be ported

1 – own number ported out (used by G-Port)

2 – foreign number ported to foreign network (used by G-Port)

3 – prepaid 1 (used by PPSMS)

4 – prepaid 2 (used by PPSMS)

5 – migrated with one handset (used by IS-41)

none – no status (default = none)

sp (Optional) Specifies which SP the DN(s) are on. The **sp** must correspond to an existing SP entity.

Values: 1 to 15 hexadecimal digits expressed using ASCII characters.

rn (Optional) Specifies which RN the DNs are on. If the requested RN does not already exist, a blank one with no values is automatically created.

Values: 1 to 15 hexadecimal digits expressed using ASCII characters.

force (Optional) Indicates whether the client wants existing instances to be overwritten.

Values: **yes** or **no** (default = **no**)

timeout (Optional) Specify the number of seconds to wait for the write transaction if another connection already has it. Clients waiting for the write transaction with this mechanism are processed in the order that their requests were received. This option is only allowed if the client used the txnmode single option on its connect request.

Values: **0** (return immediately if not available; default)
1 - **3600** seconds

Rules:

1. The **bdn** and **edn** parameter values must have the same number of digits.

2. It is not valid to specify both **sp** and **rn**.

Request syntax:

```
ent_sub([iid XXXXX,] bdn XXXXX, edn XXXXX, [sp XXXXX,]
[pt <0/1/2/3/4/5/none>,] [rn XXXXX,] [force yes/no]
[, timeout <0..3600>])
```

Response

The return codes in Table 3-7 may result from the Create Subscription request.

Table 3-7. Create Subscription Response Return Codes

Return Code	Description	Data Section Contents
SUCCESS	Everything worked.	NONE
INVALID_ VALUE	One of the fields specified had an invalid value.	The offending field is returned in the data section: `data (param <field label>)`
WRITE_IN_ READ_TXN	The **`create`** command was sent on a **`read`** transaction.	NONE

CONFLICT_ FOUND	An entry was found already in database matching an element of this request. If **`force yes`** parameter is used, this error is not returned. Rather, existing instances are overwritten.	The offending existing element is returned. The type depends on the type of request.
		`data (dn XXXXX)`
		`data (imsi XXXXX)`
		`data (bdn XXXXX, edn XXXXX)`

NE_NOT_ FOUND	The specified NE does not exist.	NONE
IMSI_DN_ LIMIT	The addition of DNs specified in request would cause IMSI to have more than eight DNs	NONE
TXN_TOO_BIG	This request would cause current transaction to be larger than limit.	NONE
NO_UPDATES	The database already contains data in request. No update was necessary.	NONE
NO_ACTIVE_ TXN	There was no currently active transaction for this connection.	NONE
NO_WRITE_ PERMISSION	The PDBI client making request does not have write access permissions.	NONE
WRITE_ UNAVAIL	Another client already has a write transaction open.	IP address information of client that already has the write transaction. `data (id <connection id>, ip <ip addr>, port <port num>)`

Update Subscription

This command modifies existing subscription data. Specific scenarios are described in the usage variations below. Although all of the usage variations are called **`upd_sub`**, the existence of certain parameters changes what is meant.

Request

Modify the SP for a specific IMSI

This command attempts to modify the **sp** field for a specific IMSI. If the IMSI has any DNs associated with it, the DNs are modified to use the specified SP.

NOTE: This type of subscription data is used only by the G-Flex feature.

Parameters:

imsi A single IMSI.

Values: 5 to 15 hexadecimal digits expressed using ASCII characters.

sp Specifies which SP the IMSI is being moved to. The SP must correspond to an existing SP entity.

Values: 1 to 15 hexadecimal digits expressed using ASCII characters.

timeout (Optional) Specify the number of seconds to wait for the write transaction if another connection already has it. Clients waiting for the write transaction with this mechanism are processed in the order that their requests were received. This option is only allowed if the client used the txnmode single option on its connect request.

Values: **0** (return immediately if not available; default)
1 - **3600** seconds

Rules:

1. The **sp** parameter must be specified.

Request syntax:

```
upd_sub([iid XXXXX,] imsi XXXXX, [ sp XXXXX,]
[timeout <0..3600>])
```

Modify the subscription data of a single DN

This command attempts to modify the SP, RN and portability type fields of a specific DN. If the DN is a stand-alone DN, the fields are simply modified. If the DN is associated with an IMSI and a new NE was specified in the request, the DN is removed from the IMSI and changed to use the specified NE directly.

Updating a DN by setting the **sp** and **rn** to the value **none** results in a DN that is not associated with a network entity.

This command is used for G-Flex, G-Port, and INP features, although some of the specific parameters are only meaningful on specific features.

Parameters:

dn A single DN (specified in international format).

Values: 5 to 15 hexadecimal digits expressed using ASCII characters.

pt (Optional) The portability type for the created DN. This field is only used by G-Port, IS-41 to GSM migration and PPSMS. For G-Port, it controls number Portability Status encoding in SRI acks. For IS-41 to GSM migration it identifies whether a subscriber has or has not migrated from IS-41 to GSM, (maintaining a single GSM handset). For PPSMS, it identifies a DN as one of two types needing PPSMS intercept.

Values: **0** – not known to be ported

1 – own number ported out (used by G-Port)

2 – foreign number ported to foreign network (used by G-Port)

3 – prepaid 1 (used by PPSMS)

4 – prepaid 2 (used by PPSMS)

5 – migrated with one handset (used by IS-41)

none – no status (default = none)

sp (Optional) Specifies which SP the DN is being moved to. The **`sp`** must correspond to an existing SP entity. Most INP only customers do not need to use **`sp`**.

Values: 1 to 15 hexadecimal digits expressed using ASCII characters.

none – Sets the **sp** to not point to any network entity.

rn (Optional) Specifies which RN the DN is being moved to. If the requested RN does not already exist, a blank one with no values is automatically created. G-Flex-only customers should not use RNs.

Values: 1 to 15 hexadecimal digits expressed using ASCII characters.

none – Sets the **rn** to not point to any network entity.

timeout (Optional) Specify the number of seconds to wait for the write transaction if another connection already has it. Clients waiting for the write transaction with this mechanism are processed in the order that their requests were received. This option is only allowed if the client used the txnmode single option on its connect request.

Values: **0** (return immediately if not available; default)
1 - 3600 seconds

Rules:

1. It is not valid to specify both **sp** and **rn**, unless they are both set to **none**.

Request syntax:

```
upd_sub([iid XXXXX,] dn XXXXX, [pt <0/1/2/3/4/5/none>,]
[sp XXXXX,] [rn XXXXX,] [, timeout <0..3600>])
```

Move an existing DN to an existing IMSI

This command attempts to move an existing DN to an existing IMSI. The DN is changed to use the SP of the IMSI.

Parameters:

dn A single DN (specified in international format).

Values: 5 to 15 hexadecimal digits expressed using ASCII characters.

imsi A single IMSI.

Values: 5 to 15 hexadecimal digits expressed using ASCII characters.

timeout (Optional) Specify the number of seconds to wait for the write transaction if another connection already has it. Clients waiting for the write transaction with this mechanism are processed in the order that their requests were received. This option is only allowed if the client used the txnmode single option on its connect request.

Values: **0** (return immediately if not available; default)
1 - **3600** seconds

Request syntax:

```
upd_sub([iid XXXXX,] dn XXXXX, imsi XXXXX
[, timeout <0..3600>])
```

Modify the subscription information for a DN block

This command attempts to modify the subscription data for a DN block. The block specified must exactly match an existing block. It cannot span multiple blocks or into unused DNs. It cannot be a subset of an existing block.

DN blocks might or might not be associated with a network entity, but they cannot be associated with *both* an SP and an RN at the same time. Updating a DN block by setting both **sp** and **rn** to the value **none** results in a DN block that is not associated with a network entity.

NOTE: DN blocks are used only in the G-Port, PPSMS, and INP features.

Parameters:

bdn The beginning DN (specified in international format).

Values: 5 to 15 hexadecimal digits expressed using ASCII characters.

edn The ending DN (specified in international format).

Values: 5 to 15 hexadecimal digits expressed using ASCII characters.

pt (Optional) The portability type for the created DN. This field is only used by G-Port, IS-41 to GSM migration and PPSMS. For G-Port, it controls number Portability Status encoding in SRI acks. For IS-41 to GSM migration it identifies whether a subscriber has or has not migrated from IS-41 to GSM, (maintaining a single GSM handset). For PPSMS, it identifies a DN as one of two types needing PPSMS intercept.

Values: **0** – not known to be ported

1 – own number ported out (used by G-Port)

2 – foreign number ported to foreign network (used by G-Port)

3 – prepaid 1 (used by PPSMS)

4 – prepaid 2 (used by PPSMS)

5 – migrated with one handset (used by IS-41)

none – no status (default = none)

sp (Optional) Specifies which SP the DN is being moved to. The **sp** must correspond to an existing SP entity.

Values: 1 to 15 hexadecimal digits expressed using ASCII characters.

none – Sets the **sp** to not point to any network entity.

rn (Optional) Specifies which RN the DN is being moved to. If the requested RN does not already exist, a blank one with no values is automatically created.

Values: 1 to 15 hexadecimal digits expressed using ASCII characters.

none – Sets the **rn** to not point to any network entity.

timeout (Optional) Specify the number of seconds to wait for the write transaction if another connection already has it. Clients waiting for the write transaction with this mechanism are processed in the order that their requests were received. This option is only allowed if the client used the txnmode single option on its connect request.

Values: **0** (return immediately if not available; default)
1 - 3600 seconds

Rules:

1. The **bdn** and **edn** parameter values must have the same number of digits.

Request syntax:

```
upd_sub([iid XXXXX,] bdn XXXXX, edn XXXXX,
[pt <0/1/2/3/4/5/none>,] [sp XXXXX,] [rn XXXXX,]
[timeout <0..3600>])
```

Response

The return codes listed in Table 3-8 indicate the result of the Update Subscription request.

Table 3-8. Update Subscription Response Return Codes

Return Code	Description	Data Section Contents
SUCCESS	Everything worked.	NONE
INVALID_VALUE	One of the fields specified had an invalid value.	The offending field is returned in data section: `data (param <field label>)`
WRITE_IN_ READ_TXN	The command was sent on a **read only** transaction.	NONE
NOT_ FOUND	The requested DN, DN block, or IMSI was not found.	NONE
IMSI_DN_ LIMIT	The IMSI already has maximum number of DNs.	NONE
TXN_TOO_BIG	This request would cause current transaction to be larger than limit.	NONE
NO_UPDATES	Database already contains date in request. No update is necessary.	NONE
NO_ACTIVE_TXN	There is no currently active transaction for this connection.	NONE
NE_NOT-FOUND	NE specified does not exist.	NONE
NO_WRITE_ PERMISSION	PDBI client making request does not have write access permissions.	NONE
WRITE_ UNAVAIL	Another client already has a write transaction open.	IP address information of client that already has write transaction. `data (id <connection id>, ip <ip addr>, port <port num>)`

Delete Subscription

This command deletes subscription records. The specific usage variations follow. Although all of the usage variations are called **`dlt_sub`**, the existence of certain parameters change what is meant.

Request

Delete an IMSI

This command attempts to delete the specified IMSI. If the IMSI has any DNs associated with it, the DNs are also deleted.

Parameters:

imsi The IMSI to delete.

Values: 5 to 15 hexadecimal digits expressed using ASCII characters.

timeout (Optional) Specify the number of seconds to wait for the write transaction if another connection already has it. Clients waiting for the write transaction with this mechanism are processed in the order that their requests were received. This option is only allowed if the client used the txnmode single option on its connect request.

Values: **0** (return immediately if not available; default)
1 - 3600 seconds

Request syntax:

```
dlt_sub([iid XXXXX,] imsi XXXXX, [timeout <0..3600>])
```

Delete a single DN

This command attempts to delete a single DN. The DN is deleted even if the DN is associated with an IMSI. The IMSI remains even if this operation results in no DNs being associated with the IMSI.

Parameters:

dn The single DN to delete (specified in international format).

Values: 5 to 15 hexadecimal digits expressed using ASCII characters.

timeout (Optional) Specify the number of seconds to wait for the write transaction if another connection already has it. Clients waiting for the write transaction with this mechanism are processed in the order that their requests were received. This option is only allowed if the client used the txnmode single option on its connect request.

Values: **0** (return immediately if not available; default)
1 - **3600** seconds

Request syntax:

```
dlt_sub([iid XXXXX,] dn XXXXX, [timeout <0..3600>])
```

Delete a DN block

This command attempts to delete an existing DN block. The block specified must exactly match an existing block. It cannot span multiple blocks or into unused DNs. It cannot be a subset of an existing block.

Parameters:

bdn The beginning DN of the block to delete (specified in international format).

Values: 5 to 15 hexadecimal digits expressed using ASCII characters.

edn The ending DN of the block to delete (specified in international format).

Values: 5 to 15 hexadecimal digits expressed using ASCII characters.

timeout (Optional) Specify the number of seconds to wait for the write transaction if another connection already has it. Clients waiting for the write transaction with this mechanism are processed in the order that their requests were received. This option is only allowed if the client used the txnmode single option on its connect request.

Values: **0** (return immediately if not available; default)
1 - **3600** seconds

Rules:

1. The **bdn** and **edn** parameter values must have the same number of digits.

Request syntax:

```
dlt_sub([iid XXXXX,] bdn XXXXX, edn xxxxx, [timeout <0..3600>])
```

Response

The return codes listed in Table 3-9 indicate the result of the Delete Subscription request.

Table 3-9. Delete Subscription Response Return Codes

Return Code	Description	Data Section Contents
SUCCESS	Everything worked.	NONE
WRITE_IN_ READ_TXN	The command was sent on a **read only** transaction.	NONE
NOT_ FOUND	The requested DN, DN block, or IMSI was not found.	NONE
TXN_TOO_ BIG	This request would cause current transaction to be larger than the limit.	NONE
NO_ACTIVE_ TXN	There is no currently active transaction for this connection.	NONE
NO_WRITE_ PERMISSION	PDBI client making request does not have write access permissions.	NONE
WRITE_ UNAVAIL	Another client already has a write transaction open.	IP address information of client that already has the write transaction. `data (id <connection id>, ip <ip addr>, port <port num>)`

Retrieve Subscription Data

The command allows the client to retrieve information about the existing subscription data. The specific usage variations follow. Although all of the usage variations are called **`rtrv_sub`**, the existence of certain parameters change what is meant.

When multiple filtering parameters are specified (**pt**, **sp**, and **rn**), any output data must pass ALL of the filters specified. For example, if **sp** is specified, only instances referencing the specified SP values will be returned.

Request

Retrieve subscription information about a specific DN

This command retrieves the subscription information for a specific DN. If the G-Port or INP feature is available and the specific DN is not found, the PDBA also tries to find a DN block that the DN is in.

Parameters:

dn The specific DN to retrieve (specified in international format).

Values: 5 to 15 hexadecimal digits expressed using ASCII characters.

data (Optional) Lets the requestor specify the type of output data to be returned. See "Response" on page 3-34 for additional information.

Values: **all** – Return all known data for each instance. (default)
neonly – Return only the Network Element information for each instance.

Request syntax:

```
rtrv_sub([iid XXXXX,] dn XXXXX, [data <all/neonly>]
```

Retrieve subscription information for a range of DNs

This command retrieves all of the subscription data within a range of DNs.

Parameters:

bdn The starting **dn** for the DN range (specified in international format).

Values: 5 to 15 hexadecimal digits expressed using ASCII characters.

edn The ending **dn** for the DN range (specified in international format).

Values: 5 to 15 hexadecimal digits expressed using ASCII characters.

type (Optional) Whether to report the DN blocks or the single DNs.

Values: **block** – Searches the DN Block table. Reports only DN Blocks, regardless of whether any provisioned Single DNs fall within the specified number range

single (default) - Searches the Single DN table. Returns only DNs that were provisioned as Single DNs, regardless of whether the DN number falls within the number range of a provisioned DN block)

pt (Optional) The portability type for the created DN. This field is only used by G-Port, IS-41 to GSM migration and PPSMS. For G-Port, it controls number Portability Status encoding in SRI acks. For IS-41 to GSM migration it identifies whether a subscriber has or has not migrated from IS-41 to GSM, (maintaining a single GSM handset). For PPSMS, it identifies a DN as one of two types needing PPSMS intercept.

Values: **0** – not known to be ported

1 – own number ported out (used by G-Port)

2 – foreign number ported to foreign network (used by G-Port)

3 – prepaid 1 (used by PPSMS)

4 – prepaid 2 (used by PPSMS)

5 – migrated with one handset (used by IS-41)

none – no status (default = none)

sp (Optional) Filters the request to just retrieve the DNs in the range that are on the provided SP.

Values: 1 to 15 hexadecimal digits expressed using ASCII characters.

none – Filters for instances not on an SP.

rn (Optional) Filters the request to just retrieve the DNs in the range that are on the provided RN.

Values: 1 to 15 hexadecimal digits expressed using ASCII characters.

none – Filters for instances not on an RN.

data (Optional) Lets the requestor specify the type of output data to be returned. See "Response" on page 3-34 for additional information.

Values: **all**– Return all known data for each instance (default).

neonly – Return only the Network Element information for each instance.

count – Return only a single instance count of all instances matching the query.

num (Optional) Allows the client to limit the number of items that are returned.

Values: **1 - 40000000**

Rules:

1. Specifying both **`sp`** and **`rn`** is not allowed because it would always result in no instances being found.

2. The **type** parameter acts to filter the responses based on how the data was provisioned. For instance, if **block** is specified, only DN blocks (that were provisioned as blocks) are returned. If **single** is specified, only DNs that were provisioned as single DNs are returned.

NOTE: If a substantial number of records are requested, there is a significant delay before responses start coming back.

There is no association between the position of a DN in the database and the chronological order in which it is provisioned. Thus, the response to the **retrieve** command gives no indication of which DNs were provisioned first and which were provisioned last. It simply returns all DNs associated with an IMSI. Likewise, the provisioning order of DNs have no effect on where DNs are physically placed in the database.

The instance count values are given as optional because they are not returned for every procedure call. For instance, IMSIs are not returned for commands related to retrieval of DNs.

Request syntax:

```
rtrv_sub([iid XXXXX,] bdn XXXXX, edn XXXXX, [type
<block/single>,] [pt <0/1/2/3/4/5/none>], [sp XXXXX,]
[rn XXXXX,] [data <all/neonly/count>,]
[num <1..40000000>])
```

Retrieve subscription information about a specific IMSI

This command retrieves the subscription information for a specific IMSI. If the IMSI is associated with DNs, they are returned as well.

Parameters:

imsi The specific IMSI to retrieve.

Values: 5 to 15 hexadecimal digits expressed using ASCII characters.

data (Optional) Lets the requestor specify the type of output data to be returned. See "Response" on page 3-34 for additional information.

Values: **all** – Return all known data for each instance (i.e., list the DNs on each IMSI) (default).

neonly – Return only the Network Element information for each instance.

Request syntax:

```
rtrv_sub([iid XXXXX,] imsi XXXXX, [data <all/neonly>])
```

Retrieve subscription information for a range of IMSIs

This command retrieves all of the subscription data within a range of IMSIs.

Parameters:

bimsi The starting IMSI for the IMSI range.

Values: 5 to 15 hexadecimal digits expressed using ASCII characters.

eimsi The ending IMSI for the IMSI range.

Values: 5 to 15 hexadecimal digits expressed using ASCII characters.

sp (Optional) Filters the request to just retrieve the DNs in the range that are on the provided SP.

Values: 1 to 15 hexadecimal digits expressed using ASCII characters.

data (Optional) Lets the requestor specify the type of output data to be returned. See "Response" on page 3-34 for additional information.

Values: **all** – Return all known data for each instance (i.e., list the DNs on each IMSI) (default).

neonly – Return only the Network Element information for each instance.

count – Return only a single instance count of all instances matching the query.

num (Optional) Allows the client to limit the number of items to be returned.

Values: **1 - 40000000**

NOTE: If a substantial number of records are requested, there is a significant delay before the responses start coming back.

Request syntax:

```
rtrv_sub([iid XXXXX,] bimsi XXXXX, eimsi XXXXX, [sp XXXXX,]
[data <all/neonly/count>,] [num <1..40000000>])
```

Response

The syntax of the data section of responses to a successful Retrieve Subscription Data request depends on the type of records being returned. DN records, DN block records, IMSI records, or instance counts can be returned. Each type of data being returned has a different syntax.

The responses that actually return instance data also have optional relationship information that can be present. For example, in an IMSI response there is a list of the DNs that are on that IMSI. If all the requestor cares about is the IMSI-to-SP mapping, this additional DN relationship information can be omitted from the

IMSI section of the response by specifying the value **neonly** in the **data** parameter. A **data** value of **all** returns all of the optional information that is present in the instances. The same type of relationship information is also present in the DN section and the same parameter has the effect of omitting it.

- Response syntax for an IMSI query:

```
data ([segment XXXXX], imsis (imsi (id XXXXX,
[dns ( XXXXX, . . .),]  sp XXXXX) ),
. . .
(. . .) ) )
```

- Response syntax for a DN query:

```
data ([segment XXXXX], dns (dn (id XXXXX, [imsi XXXXX,]
[pt <0/1/2/3/4/5/none>], [sp XXXXX,] [rn XXXXX]) ),
. . .,
(. . .) ) )
```

- Response syntax for a DN block query:

```
data ([segment XXXXX], dnblocks (dnblock (bdn XXXXX, edn XXXXX,
[pt <0/1/2/3/4/5/none>], [sp XXXXX,] [rn XXXXX]),
. . .,
(. . .) ) )
```

- Response syntax for a count query:

  Requests that specify a **data** parameter of **count** gets just one response that contains the instance count for the type of subscription data that they are querying. Only one of the optional counts would be present in the response.

```
data (counts ([imsi #####,] [dn #####,] [dnblock #####]) )
```

The return codes listed in Table 3-10 indicate the result of the Retrieve Subscription Data request.

Table 3-10. Retrieve Subscription Data Response Return Codes

Return Code	Description	Data Section Contents
SUCCESS	The request succeeded and this is the last (or only) response.	Depends on the request type, etc.
INVALID_VALUE	One of the fields specified had an invalid value.	Offending field is returned in data section: **`data (param <field label>)`**
PARTIAL_SUCCESS	The request has succeeded, but this is only one of many responses.	Depends on the request type, etc.
NOT_FOUND	The requested DN, DN block, or IMSI was not found.	NONE
NO_ACTIVE_TXN	There is no currently active transaction for this connection.	NONE
NE_NOT-FOUND	An NE to filter was specified, but the NE does not exist.	NONE

Create Network Entity

The **`ent_entity`** command creates an entity object (such as an SP) and its corresponding global title translation. There is a limit of 150,000 network entity instances.

It is valid for entities of different types to have the same **id**.

Request

Parameters:

id Identifier for this network entity.

Values: 1 to 15 hexadecimal digits expressed using ASCII characters.

type Type of network entity being created.

Values: **SP** – Signal Point

RN – Routing Number.
(G-Flex-only customers would not use RNs.)

pctype Specifies the type of the point code.

Values: **intl** - ITU international point code in the form zone-area-id (z-aaa-i).

natl - ITU national point code in the form of ITU number (nnnnn).

nl24 - ITU national 24-bit point code in the form of msa-ssa-sp (mmm-sss-ppp).

ansi - ANSI point code in the form of network-cluster-member (nnn-ccc-mmm).

none - No point code specified. (Only valid for RNs.)

pc Point code value. The valid values depend on the **pctype** parameter.

Values:

For **pctype** of **intl** the format is zone-area-id (z-aaa-i).

z= **0** – **7**

aaa= **0** – **255**

i= **0** - **7**

Note: The value 0-0-0 is not valid.

For **pctype** of **natl** the format is number (nnnnn).

nnnnn= **1** – **16383**

For **pctype** of **ansi**, the format is network-cluster-member (nnn-ccc-mmm).

nnn= **1** – **255**

ccc= **1** – **255** (if network = **1** – **5**)

= **0** – **255** (if network = **6** – **255**)

mmm= **0** – **255**

For **pctype** of **none**, the **pc** parameter is not allowed.

gc (Optional) Group code. This optional parameter is part of the point code value for ITU Duplicate Point Code Support feature.

Values: **aa** - **zz**

ri Routing indicator. This parameter indicates whether a subsequent global title translation is required.

Values: **GT** = Global Title. Indicates that a subsequent translation is required.

SSN = Subsystem Number. Indicates that no further translation is required.

ssn (Optional) New subsystem number. This parameter identifies the subsystem address that is to receive the message.

Values: **0, 2 – 255**

none (default)

ccgt (Optional) Cancel called global title.

Values: **yes** or **no** (default)

ntt (Optional) New translation type. This parameter identifies the translation type value to replace the received translation type value.

Values: **0 – 255**

none (default)

nnai (Optional) New nature of address.

Values: **0 – 127**

none (default)

nnp (Optional) New numbering plan.

Values: **0 – 15**

none (default)

da (Optional) Digit action. The parameter specifies what changes, if any, to apply to the Called Party GTA.

Values: **none** – No change to the Called Party GTA (default)

replace – Replace Called Party GTA with the entity id

prefix – Prefix Called Party GTA with the entity id

insert – Insert entity id after country code (CC + Entity Id + NDC + SC)

delccprefix – Delete country code, then prepend the entity id.

delcc – Delete country code.

spare1 – No change to GTA. Digit action value of 6 passed to Eagle.

spare2 – No change to GTA. Digit action value of 7 passed to Eagle.

srfimsi (Optional) The IMSI returned by a SRF indicating the Subscription Network of the subscriber. This parameter is only used by the G-Port features and only for RNs.

Values: 5 to 15 hex digits expressed using ASCII characters.

timeout (Optional) Specify the number of seconds to wait for the write transaction if another connection already has it. Clients waiting for the write transaction with this mechanism are processed in the order that their requests were received. This option is only allowed if the client used the txnmode single option on its connect request.

Values: **0** (return immediately if not available; default)
1 - **3600** seconds

Rules:

1. If the **pctype** is **none**, none of the optional parameters can be specified.
2. If **ri** is **GT, ccgt** must be **no**.
3. If **ccgt** is **yes**, the parameters **ntt**, **nnai**, **nnp**, and **da** cannot be set.
4. The maximum number of network entities (150,000) must not be reached.
5. Parameter **gc** can be specified only when **pctype = natl**.

Request syntax:

```
ent_entity([iid XXXXX,] id XXXXX, type <SP/RN>, pctype
<intl/natl/ansi/none>, [pc <pc value>,], [gc <gc value>,]
[ri <GT/SSN>,] [ssn <0/2..255/none>,] [ccgt <yes/no>,]
[ntt <0..255/none>,] [nnai <0..127/none>,] [nnp <0..15/none>,]
[da <none/replace/prefix/insert/delccprefix/delcc/spare1
/spare2>,] [srfimsi XXXXX] [, timeout <0..3600>])
```

Response

The return codes listed in Table 3-11 indicate the result of the Create network entity request.

Table 3-11. Create Network Entity Response Return Codes

Return Code	Description	Data Section Contents
SUCCESS	Everything worked.	NONE
INVALID_ VALUE	One of the fields specified had an invalid value.	Offending field is returned in data section: **data (param <field label>)**
WRITE_IN_ READ_TXN	The command was sent on a **read only** transaction.	NONE

Table 3-11. Create Network Entity Response Return Codes

Return Code	Description	Data Section Contents
ITEM_EXISTS	The network entity already exists.	
TXN_TOO_BIG	This request would cause current transaction to be larger than the limit.	NONE
NO_ACTIVE_TXN	There is no currently active transaction for this connection.	NONE
NO_WRITE_PERMISSION	PDBI client making request does not have write access permissions.	NONE
WRITE_UNAVAIL	Another client already has a write transaction open.	IP address information of client that already has the write transaction. `data (id <connection id>, ip <ip addr>, port <port num>)`

Update Network Entity

The **`upd_entity`** command modifies an entity object (such as an SP) and its corresponding global title translation.

Request

Parameters:

id Global title address for this network entity.

Values: 1 to 15 hexadecimal digits expressed using ASCII characters.

type Type of network entity being updated.

Values: **SP** – Signal Point

RN – Routing Number.
(G-Flex only customers do not use RNs.)

pctype (Optional) Specifies the type of the point code. If the **pctype** of an existing NE is changed, then the **pc** parameter must also be specified.

Values: **intl** - ITU international point code in the form zone-area-id (z-aaa-i).

natl - ITU national point code in the form of ITU number (nnnnn).

nl24 - ITU national 24-bit point code in the form of msa-ssa-sp (mmm-sss-ppp).

ansi - ANSI point code in the form of network-cluster-member (nnn-ccc-mmm).

none - No point code specified. (Valid only for RNs.)

pc (Optional) Point code value. The valid values depend on the **pctype** parameter. If no **pctype** parameter is specified, the **pctype** of the existing instance is used.

Values:

For **pctype** of **intl** the format is zone-area-id (z-aaa-i).

z= **0** – **7**

aaa= **0** – **255**

i= **0** – **7**

Note: The value 0-0-0 is not valid.

For **pctype** of **natl** the format is number (nnnnn).

nnnnn= **1** – **16383**

For **pctype** of **ansi**, the format is network-cluster-member (nnn-ccc-mmm).

nnn= **1** – **255**

ccc= **1** – **255** (if network = **1** – **5**)

= **0** – **255** (if network = **6** – **255**)

mmm= **0** – **255**

For **pctype** of **none**, the **pc** parameter is not allowed.

gc (Optional) Group code. This optional parameter is part of the point code value for ITU Duplicate Point Code Support feature.

Values: **aa** – **zz**

ri (Optional) Routing indicator. This parameter indicates whether a subsequent global title translation is required.

Values: **GT** = Global Title. Indicates that a subsequent translation is required.

SSN = Subsystem Number. Indicates that no further translation is required.

ssn (Optional) Subsystem number. This parameter identifies the subsystem address that is to receive the message.

Values: **0**, **2** – **255**

none

ccgt (Optional) Cancel called global title.

Values: **yes** or **no**

ntt (Optional) New translation type. This parameter identifies the type of global title translation to replace the received global title.

Values: **0 – 255**

none

nnai (Optional) New nature of address.

Values: **0 – 127**

none

nnp (Optional) New numbering plan.

Values: **0 – 15**

none

da (Optional) Digit action. The parameter specifies what changes, if any, to apply to the Called Party GTA.

Values: **none** – No change to the Called Party GTA (default)

replace – Replace Called Party GTA with the entity **id**

prefix – Prefix Called Party GTA with the entity **id**

insert – Insert entity **id** after country code (CC + Entity Id + NDC + SC)

delccprefix – Delete country code, then prepend the entity id.

delcc – Delete country code.

spare1 – No change to GTA. Digit action value of 6 passed to Eagle.

spare2 – No change to GTA. Digit action value of 7 passed to Eagle.

srfimsi (Optional) The IMSI returned by a SRF indicating the Subscription Network of the subscriber. This parameter is used only with the G-Port features and only on RNs.

Values: 5 to 15 hex digits expressed using ASCII characters.

timeout (Optional) Specify the number of seconds to wait for the write transaction if another connection already has it. Clients waiting for the write transaction with this mechanism are processed in the order that their requests were received. This option is only allowed if the client used the txnmode single option on its connect request.

Values: **0** (return immediately if not available; default)
1 – **3600** seconds

Rules:

1. Parameter **id** must already exist.
2. If the **pctype** is **none**, none of the optional parameters can be specified.
3. If **ri** is **GT**, **ccgt** must be **no**.
4. If **ccgt** is **yes**, the parameters **ntt**, **nnai**, **nnp**, and **da** cannot be set.
5. Parameter **gc** can be specified only when **pctype = natl**.

Request syntax:

```
upd_entity([iid XXXXX,] id XXXXX, type <SP/RN>, [pctype
<intl/natl/ansi/none>,] [pc <pc value>,] [gc <gc value>,]
[ri <GT/SSN>,] [ssn <0/2..255/none>,] [ccgt <yes/no>,]
[ntt <0..255/none>,] [nnai <0..127/none>,] [nnp <0..15/none>,]
[da <none/replace/prefix/insert/delccprefix/delcc/spare1
/spare2>,] [srfimsi XXXXX,] [timeout <0..3600>])
```

Response

The return codes listed in Table 3-12 indicate the result of the Update Network Entity request.

Table 3-12. Update Network Entity Response Return Codes

Return Code	Description	Data Section Contents
SUCCESS	Everything worked.	NONE
INVALID_VALUE	One of the fields specified had an invalid value.	Offending field is returned in data section: `data (param <field label>)`
WRITE_IN_READ_TXN	The command was sent on a **read only** transaction.	NONE
NOT_FOUND	The requested SP was not found.	NONE
TXN_TOO_BIG	This request would cause current transaction to be larger than the limit.	NONE
NO_ACTIVE_TXN	There is no currently active transaction for this connection.	NONE
NO_UPDATES	Database already contains the data in this request. No update necessary.	NONE
NO_WRITE_PERMISSION	PDBI client making request does not have write access permissions.	NONE
WRITE_UNAVAIL	Another client already has a write transaction open.	IP address information of client that already has the write transaction. `data (id <connection id>, ip <ip addr>, port <port num>)`

Delete Network Entity

The **`dlt_entity`** command deletes an entity object and its corresponding global title translation.

Request

This command fails if the entity does not exist or the entity is referenced by a subscription.

Parameters:

id Global title address for this network entity.

Values: 1 to 15 hexadecimal digits expressed using ASCII characters.

type Type of network entity being deleted.

Values: **SP** – Signal Point

RN – Routing Number (available only with G-Port and INP features)

timeout (Optional) Specify the number of seconds to wait for the write transaction if another connection already has it. Clients waiting for the write transaction with this mechanism are processed in the order that their requests were received. This option is only allowed if the client used the txnmode single option on its connect request.

Values: **0** (return immediately if not available; default)
1 – **3600** seconds

Request syntax:

```
dlt_entity([iid XXXXX,] id XXXXX, type <SP/RN>
[, timeout <0..3600>])
```

Response

The return codes listed in Table 3-13 indicates the result of the Delete Network Entity request.

Table 3-13. Delete Network Entity Response Return Codes

Return Code	Description	Data Section Contents
SUCCESS	Everything worked.	NONE
INVALID_ VALUE	One of the fields specified had an invalid value.	Offending field is returned in data section: **`data (param <field label>)`**
WRITE_IN_ READ_TXN	The command was sent on a **read only** transaction.	NONE
CONTAINS_ SUBS	The NE to be deleted still contains subscription data.	The counts for each type of subscription data on the NE are returned. **`data (counts([imsi #####,] [dn #####,] [dnblock #####,]))`**
NOT_ FOUND	The requested SP was not found.	NONE
TXN_TOO_BIG	The request would cause the current transaction t be larger than the limit.	NONE
NO_ACTIVE_ TXN	There is no currently active transaction for this connection.	NONE

Retrieve Network Entity

This command retrieves one or all of the network entities. The specific usage variations follow. Although all of the usage variations are called **`rtrv_entity`**, the existence of certain parameters change what is meant.

Request

Retrieve the information for a specific NE

Parameters:

id Global title address for this network entity.

Values: 1 to 15 hexadecimal digits expressed using ASCII characters.

type Type of network entity to be retrieved.

Values: **SP** – Signal Point

RN – Routing Number (available only with G-Port and INP features)

Request syntax:

```
rtrv_entity([iid XXXXX,] id XXXXX, type <SP/RN>)
```

Retrieve the information for a range of NEs

Parameters:

bid Global title address for the first network entity in the range.

Values: 1 to 15 hexadecimal digits expressed using ASCII characters.

eid Global title address for the last network entity in the range.

Values: 1 to 15 hexadecimal digits expressed using ASCII characters.

type (Optional) Type of network entity being deleted.

Values: **SP** – Signal Point

RN – Routing Number (available only with G-Port and INP features)

data (Optional) Lets the requestor specify the type of output data to be returned. See the response section for additional information.

Values: **all** – Return all known data for each instance (default)

neonly – Return just the ID/type for each instance

count – Return only a instance count of all instances matching the query.

num (Optional) Limits the number of entities to be returned. If the **num** parameter is omitted, all entities in the range are returned.

Values: **1 – 150000**

Request syntax:

```
rtrv_entity([iid XXXXX,] bid XXXXX, eid XXXXX, [type <SP/RN>,]
[[data ,all/neonly/count>], num <1..1000>])
```

Retrieve the information for all NEs

Parameters:

num (Optional) Limits the number of entities to be returned. If the **num** parameter is omitted, all entities are returned.

Values: **1 – 150000**

Request syntax:

```
rtrv_entity([iid XXXXX,] [num <1..150000>])
```

Response

The data section for the responses of all **rtrv_entity** request types depends on the **data** parameter type specified in the request. If the **data** value is **all**, the data section contains a list of all instances that matched the request. It contains a segment parameter similar to the one in **rtrv_sub** for large range retrievals, followed by a list of network entities (**news**). With the exception of **pctype**, parameters whose values are **none** are not present in the response.

```
data ( segment #####, nes( (id XXXXX, type <SP/RN>, pctype
<intl/natl/ansi/none>, [pc <point code>,] [gc <group code>,]
ri <GT/SSN>, [ssn <0,2..225>,] ccgt <yes/no>, [ntt <0..255>,]
[nna <0..127>,] [nnp <0..15>,]
[da <replace/prefix/insert/delcc/delccprefix/spare1/spare2>,]
[srfimsi XXXXX,] counts([imsi ###,] [dn ###,] [dnblock ###])),
( ... ) )
```

As with the responses for retrieving subscriptions, the response can be broken up into multiple responses due to size constraints. Intermediate responses have the return code PARTIAL_SUCCESS.

If the **data** value is **count**, the data section contains only the number of instances that matched the query.

```
data (counts(ne ###))
```

The return codes listed in Table 3-14 indicate the result of the Retrieve Network Entity request.

Table 3-14. Retrieve Network Entity Response Return Codes

Return Code	Description	Data Section Contents
SUCCESS	The request succeeded and this is the last (or only) response.	See data description above.
INVALID_ VALUE	One of the fields specified had an invalid value.	Offending field is returned in data section: **`data (param <field label>)`**
PARTIAL_ SUCCESS	The request has succeeded, but this is only one response in many.	See data description above.
NOT_ FOUND	The requested **`id`** was not found.	NONE
NO_ACTIVE_ TXN	There is no currently active transaction for this connection.	NONE
NO_WRITE_ PERMISSION	PDBI client making request does not have write access permissions.	NONE
WRITE_ UNAVAIL	Another client already has a write transaction open.	IP address information of client that already has the write transaction. **`data (id <connection id>, ip <ip addr>, port <port num>)`**

Switchover

The **`switchover`** command causes the two PDBAs to switch Active/Standby status. By default, the command works like a toggle switch. The PDBA receiving the request changes its status from Active to Standby or from Standby to Active and informs the other PDBA to do the opposite.

The **`side`** parameter in the request specifies the desired status for the receiving PDBA. If this parameter is used, the receiving PDBA attempts to set itself to the requested status and tells the mate PDBA to set itself to the opposite status. If the two PDBAs are already in the desired states, no action is taken.

Because the goal of the switchover command is to change the Active/Standby status of the PDBA, and because **write** transactions can be done only on the Active PDBA, it is a requirement that no **write** transactions be active for a switchover to be performed. It is also a requirement that all asynchronous replication be completed before the switchover is permitted.

The **switchover** command has a **timeout** parameter (similar to the **begin_txn** command) to allow the command to wait for any existing **write** transactions to complete. If the **switchover** command is being sent to the standby side and the active side has a **write** transaction that was left open, sending the **switchover** command with the **force** parameter set to **yes** overrides the open write transaction; it also allows the switchover to occur. While this behavior is permitted, it is extremely dangerous to steal the **write** transaction from an active client.

If the **write** transaction is truly hung for whatever reason, it is much safer to stop and restart the PDBA that has the **write** transaction hung. If the **switchover** command is being sent to the active side while another client has the **write** transaction open, the **switchover** is unsuccessful with WRITE_UNAVAIL, even if the **force** option is used. The **force** option is also ignored if the databases are not yet synchronized.

By default, if a PDBA application receives a **switchover** request but the PDBA is unable to communicate with its mate PDBA, the request fails. An optional **force** parameter can be used to cause the receiving PDBA to ignore the fact that it cannot communicate with its mate and perform the switchover anyway. This option can be useful if communication between the two PDBAs has been broken, but the PDBA that was previously Standby needs to become Active.

Use the **switchover** command very carefully. It is possible to use this command in such a way that causes the two PDBs to be out of synch. When the PDBAs are successfully communicating, changing the Active/Standby status of either PDBA causes the other PDBA to change as well. However, if the two PDBAs are unable to communicate, then a **switchover** command received by one of them fails because it cannot inform the mate that a switchover is taking place. This failure ensures that both PDBAs do not think that they are the Active PDBA.

If the **force** parameter is used and both PDBAs become Active, it is the client's responsibility to ensure that they are not both written to.

If updates are sent to both PDBs while they are not communicating with each other, the databases can become irreversibly out of synch. When the PDBAs see each other again, they detect the synchronization problem and force both PDBAs to be in Standby mode. Any attempt to switchover either PDBA to be active fails with the DB_MAINT_REQD return code until the problem is corrected. At that point, one PDB would have to be recreated from the other PDB, and the RTDB processes connected to the PDBA with the recreated PDB must reload (causing the cards on the EAGLE 5 SASs also to reload).

The PDBA that receives the switchover request attempts to change the state of the remote PDBA first and then change its own state. In the unlikely event that one of the PDBAs terminates during the handling of the switchover request, it is theoretically possible for the two PDBAs to be set to the same state. If this were to happen, the PDBAs automatically fix the situation when the software is restarted.

This command can be issued only by client that have WRITE permission. It cannot be issued from inside a transaction, nor can any other PDBI clients (on either PDBA) have the **write** transaction open.

Request

Parameters:

side (Optional) Specifies whether the receiving side is to be set to: Active or Standby. Without this parameter, the switchover command works like a toggle switch.

Values: **active** – Set receiving side to Active.

standby – Set receiving side to Standby.

timeout (Optional) Specifies how long to wait for an existing write transaction to complete.

Values: **0** (return immediately if not available; default)
1 – **3600** seconds

force (Optional) Forces the switch on the receiving side. This is useful when the two PDBA processes are unable to communicate (due to network problems or remote PDBA down) and you need to make the local PDBA Active anyway. By default, the local PDBA rejects a switchover request if it cannot communicate with the remote PDBA.

Values: **yes** and **no** (default)

Request syntax:

```
switchover([iid XXXXX,] [side <active/standby>],
[timeout <0-3600>], [force <yes/no>])
```

Response

The return codes listed in Table 3-15 indicate the result of the Switchover request.

Table 3-15. Switchover Response Return Codes

Return Code	Description	Data Section Contents
SUCCESS	Switchover worked.	NONE
INVALID_ VALUE	One of the fields specified had an invalid value.	Offending field is returned in data section: `data (param <field label>)`
NO_MATE	The PDBA could not negotiate a switchover with its mate. The switchover was denied.	NONE

Table 3-15. Switchover Response Return Codes (Continued)

Return Code	Description	Data Section Contents
WRITE_ UNAVAIL	There is another connection on this PDBA with the **write** transaction open.	The IP address information of the client that already has the **write** transaction. `data (id <connection id>, ip <ip addr>, port <port num>)`
NO_WRITE_ PERMISSION	The connection requesting the switchover does not have WRITE permission.	NONE
ACTIVE_TXN	The command was issued from within a transaction.	NONE
MATE_BUSY	The mate PDBA currently has a **write** transaction open.	NONE
DB_MAINT_ REQD	Replication is unable to get the two databases in synch. Call Tekelec.	NONE

PDBA Status Query

The **`status`** command queries status information from the PDBA. This command is not required to be framed inside a transaction. However, a connection must first be opened.

If the status request is made from within a transaction, the Number Prefix fields contain the values configured when the transaction started. Changes to the Number Prefixes from the user interface do not affect currently existing transactions.

If the status request is made from outside a transaction, the Number Prefixes contain the actual currently configured values. In either case, if there is no configured Number Prefixes in the user interface, the **`dnprefix`** and **`imsiprefix`** parameters are omitted to ensure backward compatibility.

Instance counts are shown as optional because certain entities/subscription types may not exist in the PDBA. For example, for clients that provision only NEs and DNs and no IMSIs (that is, G-Port), the IMSI counts are not returned.

Request

Parameters: None

Request syntax:

```
status([iid XXXXX])
```

Response

The data section of a successful PDBA Status Query contains the following information:

- PDBA version number
- Active/Standby status
- Mate connectivity – Whether or not this PDBA is connected to its mate PDBA.
- DN prefix – The default number prefix that is currently configured for DNs and DN Blocks, if any.
- IMSI prefix – The default number prefix that is currently configured for IMSI, if any.
- DB Level
- Birthdate – UNIX time_t value for time that the PDB was originally created.
- Instance counts
  - IMSI
  - DN
  - DN block
  - NE
  - ReplLog

```
data (version 1.0, side <active/standby>, mate
<present/absent>, dblevel #####, [dnprefix ####,]
[imsiprefix ####,] birthdate ##########, counts
([imsi #####,] [dn #####,] [dnblock #####,] [ne #####],)
[repllog #####])
```

The return code listed in Table 3-16 indicates the result of the PDBA Status Query request.

Table 3-16. PDBA Status Query Response Return Code

Return Code	Description	Data Section Contents
SUCCESS	Status query successful.	See description above.

Dump Connections

The **dump_conn** command requests the PDBA to dump connection information for debugging. This command is not required to be framed inside a transaction. However, a connection must first be opened.

Request

Parameters:

type Which type of connection to display information for.

Values: **PDBI** – PDBI Clients

RTDB – RTDB Clients

MAINT – Maintenance Clients

MATE – PDBA Mate

all – PDBI, RTDB, MAINT, and MATE (default)

Request syntax:

```
dump_conn(iid XXXXX, [type <PDBI/RTDB/MAINT/MATE/all>])
```

Response

The data section of a successful Dump Connections request contains the following syntax. The optional **access** parameter is returned only for PDBI connections.

```
data(connections((type <PDBI/RTDB/MAINT/MATE>, [id <connId>,]
ip <IP Addr>, port ####, [access <read/write>]), . . .
(type <PDBI/RTDB/MAINT/MATE>, [id <connId>,] ip <IP Addr>,
port ####, [access <read/write>]) ))
```

The return code listed in Table 3-17 indicates the result of the Dump Connections request.

Table 3-17. Dump Connections Response Return Code

Return Code	Description	Data Section Contents
SUCCESS	Connection list returned.	See above.
INVALID_ VALUE	One of the fields specified had an invalid value.	Offending field is returned in data section: `data (param <field label>)`

Create IMEI Data

This command creates either a single IMEI with its appropriate list type or a block of IMEIs with the associated list type. This command is also used to add additional IMSIs to a particular IMEI.

Request

Create a single entry IMEI

This command is used to create a new IMEI. Using the optional force parameter changes the default behavior to overwrite any existing entry with the new data.

The **ent_eir** command defines the request message for a single entry IMEI.

Parameters:

imei A single IMEI.

Values: 14 or 15 hexadecimal digits expressed using ASCII characters. Only the first 14 digits of the IMEI are stored and displayed on retrieval.

svn (Optional) Software Version Number.

Values: A 2-digit number **0-9** (default = **0**).

white (Optional) Select list type of White.

Values: **yes** or **no** (default = **no**).

gray (Optional) Select list type of Gray.

Values: **yes** or **no** (default = **no**).

black (Optional) Select list type of Black.

Values: **yes** or **no** (default = **no**).

imsi The IMSI(s) to be associated with an IMEI.

Values: 5 to 15 hexadecimal digits expressed using ASCII characters. Up to 8 IMSIs can be provisioned for an IMEI.

force (Optional) Indicates whether the client wants existing instances to be overwritten.

Values: **yes** or **no** (default = **no**)

timeout (Optional) Specify the number of seconds to wait for the write transaction if another connection already has it. Clients waiting for the write transaction with this mechanism are processed in the order that their requests were received. This option is only allowed if the client used the txnmode single option on its connect request.

Values: **0** (return immediately if not available; default)
1 – **3600** seconds

Rules

1. Each **imei** provisioned must reside on at least one list type (**white**, **gray**, or **black**) and can also reside on any combination of 1, 2, or 3 lists concurrently.

2. If the **imei** includes the optional 15th character (the check digit), the check digit is provided by the Customers Client Software and must match the EPAPs (via calculated algorithm). The check digit is not stored; it is only used to verify the **imei**.

3. A individual **imei** supercedes an **imei** block range.

Request syntex:

```
ent_eir( [iid XXXXX] imei XXXXX, [svn 0..99,] [white yes/no,]
[grey yes/no,] [black yes/no,] [imsi XXXXX, ..., imsi XXXXX]
[force yes/no,] [timeout <0..3600>])
```

Create a block entry of IMEIs

The **ent_eir** command defines the request message for a block entry of IMEIs.

Parameters:

bimei The beginning IMEI in a block.

Values: 14 or 15 hexadecimal digits expressed using ASCII characters. Only the first 14 digits of the IMEI are stored and displayed on retrieval.

eimei The ending IMEI in a block.

Values: 14 or 15 hexadecimal digits expressed using ASCII characters. Only the first 14 digits of the IMEI are stored and displayed on retrieval.

white (Optional) Select list type of White.

Values: **yes** or **no** (default = **no**).

gray (Optional) Select list type of Gray.

Values: **yes** or **no** (default = **no**).

black (Optional) Select list type of Black.

Values: **yes** or **no** (default = **no**).

timeout (Optional) Specify the number of seconds to wait for the write transaction if another connection already has it. Clients waiting for the write transaction with this mechanism are processed in the order that their requests were received. This option is only allowed if the client used the txnmode single option on its connect request.

Values: **0** (return immediately if not available; default)
1 - **3600** seconds

Rules

1. Each **imei** provisioned must reside on at least one list type (**white**, **gray**, or **black**) and can also reside on any combination of 1, 2, or 3 lists concurrently.

2. If the **imei** includes the optional 15th character (the check digit), the check digit is provided by the Customers Client Software and must match the EPAPs (via calculated algorithm). The check digit is not stored; it is only used to verify the **imei**. The check digit is run on both the **bimei** and **eimei** when applicable.

3. The **svn** is not provisionable on an **imei** block entry.

Request syntex:

```
ent_eir([[iid XXXXX] bimei XXXXX, eimei XXXXX,  [white yes/no,]
[grey yes/no,] [white yes/no]  [timeout <0..3600>])
```

Create a new IMSI and associate it with an existing IMEI

The **ent_eir** command defines the request message to create a new IMSI and associate it with an existing IMEI. This is the EIR specific IMSI, not the G-Port/G-Flex IMSI.

Parameters:

imei A single IMEI.

Values: 14 or 15 hexadecimal digits expressed using ASCII characters. Only the first 14 digits of the IMEI are stored and displayed on retrieval.

imsi The IMSI(s) to be associated with an IMEI.

Values: 5 to 15 hexadecimal digits expressed using ASCII characters. Up to 8 IMSIs can be provisioned for an IMEI.

timeout (Optional) Specify the number of seconds to wait for the write transaction if another connection already has it. Clients waiting for the write transaction with this mechanism are processed in the order that their requests were received. This option is only allowed if the client used the txnmode single option on its connect request.

Values: **0** (return immediately if not available; default)
1 – **3600** seconds

Rules:

1. If the **imei** includes the optional 15th character (the check digit), the check digit is provided by the Customers Client Software and must match the EPAPs (via a calculated by algorithm). The check digit is not stored; it is only used to verify the **imei**.

Request syntex:

```
ent_eir( [iid XXXXX] imei XXXXX, [imsi XXXXX, …, imsi XXXXX]
[timeout <0..3600>])
```

Response

The return codes in Table 3-18 may result from the Create IMEI request.

Table 3-18. Create IMEI Response Return Codes

Return Code	Description	Data Section Contents
SUCCESS	Everything worked.	NONE
INVALID_ VALUE	One of the fields specified had an invalid value.	The offending field is returned in the data section: `data (param <field label>)`
WRITE_IN_ READ_TXN	The **`create`** command was sent on a **read** only transaction.	NONE
CONFLICT_ FOUND	An entry was found already in database matching an element of this request. If **`force yes`** parameter is used, this error is not returned. Rather, existing instances are overwritten.	The offending existing element is returned. `data (imei XXXXX)`
CHECK DIGIT ERROR	The check digit provisioned did not match the calculated check digit.	imei, bimei, or eimei data (param imei xxxxx)
MAX_IMEI_ LIMIT	Exceeded the maximum number of individual IMEIs.	NONE
MAX_IMEI_ BLK_LIMIT	Exceeded the maximum number of IMEI blocks.	NONE
IMEI_IMSI_ LIMIT	Would cause more than 8 IMSIs to be provisioned on an IMEI.	NONE
TXN_TOO_BIG	This request would cause current transaction to be larger than limit.	NONE
NO_UPDATES	The database already contains data in request. No update was necessary.	NONE
NO_ACTIVE_ TXN	There was no currently active transaction for this connection.	NONE
NO_WRITE_ PERMISSION	The PDBI client making request does not have write access permissions.	NONE
WRITE_ UNAVAIL	Another client already has a write transaction open.	IP address information of client that already has the write transaction. `data (id <connection id>, ip <ip addr>, port <port num>)`

Update IMEI Data

This command allows the list types for an IMEI or SVN to be changed.

Request

Update a single entry IMEI

This command is used to update an existing single entry IMEI.

The **`upd_eir`** command defines the request message to update a single entry IMEI.

Parameters:

imei A single IMEI.

Values: 14 or 15 hexadecimal digits expressed using ASCII characters. Only the first 14 digits of the IMEI are stored and displayed on retrieval.

svn (Optional) Software Version Number.

Values: A 2-digit number **0-9** (default is not changed).

white (Optional) Select list type of White.

Values: **yes** or **no** (default is not changed).

gray (Optional) Select list type of Gray.

Values: **yes** or **no** (default is not changed).

black (Optional) Select list type of Black.

Values: **yes** or **no** (default is not changed).

timeout (Optional) Specify the number of seconds to wait for the write transaction if another connection already has it. Clients waiting for the write transaction with this mechanism are processed in the order that their requests were received. This option is only allowed if the client used the txnmode single option on its connect request.

Values: **0** (return immediately if not available; default)
1 - 3600 seconds

Rules

1. The resulting **imei** must have at least 1 list type (white, gray or black) turned on.

2. If the **imei** includes the optional 15th character (the check digit), the check digit is provided by the Customers Client Software and must match the EPAPs (via calculated algorithm). The check digit is not stored; it is only used to verify the **imei**.

Request syntex:

```
upd_eir([iid XXXXX] imei XXXXX, [svn 0..99,] [white
yes/no,][grey yes/no,] [black yes/no]  [timeout <0..3600>])
```

Update a block entry of IMEIs

The **upd_eir** command defines the request message to update a block entry of IMEIs.

Parameters:

bimei The beginning IMEI in a block.

Values: 14 or 15 hexadecimal digits expressed using ASCII characters. Only the first 14 digits of the IMEI are stored and displayed on retrieval.

eimei The ending IMEI in a block.

Values: 14 or 15 hexadecimal digits expressed using ASCII characters. Only the first 14 digits of the IMEI are stored and displayed on retrieval.

white (Optional) Select list type of White.

Values: **yes** or **no** (default is not changed).

gray (Optional) Select list type of Gray.

Values: **yes** or **no** (default is not changed).

black (Optional) Select list type of Black.

Values: **yes** or **no** (default is not changed).

timeout (Optional) Specify the number of seconds to wait for the write transaction if another connection already has it. Clients waiting for the write transaction with this mechanism are processed in the order that their requests were received. This option is only allowed if the client used the txnmode single option on its connect request.

Values: **0** (return immediately if not available; default)
1 - **3600** seconds

Rules

1. The resulting **imei** must have at least 1 list type (white, gray or black) turned on.

2. If the **imei** includes the optional 15th character (the check digit), the check digit is provided by the Customers Client Software and must match the EPAPs (via calculated algorithm). The check digit is not stored; it is only used to verify the **imei**.

Request syntex:

```
upd_eir([iid XXXXX] bimei XXXXX, eimei XXXXX,  [white
yes/no,][grey yes/no,] [black yes/no,] [timeout <0..3600>]))
```

Response

The return codes in Table 3-19 may result from the Update IMEI request.

Table 3-19. Update IMEI Response Return Codes

Return Code	Description	Data Section Contents
SUCCESS	Everything worked.	NONE
INVALID_ VALUE	One of the fields specified had an invalid value.	The offending field is returned in the data section: `data (param <field label>)`
NO_LIST_ FOR_IMEI	A minimum of 1 list type must be provisioned for the resulting IMEI.	NONE
CHECK DIGIT ERROR	The check digit provisioned did not match the calculated check digit.	imei, bimei, or eimei data (param imei xxxxx)
NOT_FOUND	The requested IMEI or IMEI block was not found.	NONE
WRITE_IN_ READ_TXN	The **`update`** command was sent on a **read** only transaction.	NONE
TXN_TOO_BIG	This request would cause current transaction to be larger than limit.	NONE
NO_UPDATES	The database already contains the data in the request. No update was necessary.	NONE
NO_ACTIVE_ TXN	There was no currently active transaction for this connection.	NONE
NO_WRITE_ PERMISSION	The PDBI client making request does not have write access permissions.	NONE

Table 3-19. Update IMEI Response Return Codes (Continued)

Return Code	Description	Data Section Contents
WRITE_ UNAVAIL	Another client already has a write transaction open.	IP address information of client that already has the write transaction. `data (id <connection id>, ip <ip addr>, port <port num>)`

Delete IMEI Data

This command is used to delete an individual IMEI or a IMEI block. This command is also used to delete an IMSI from the associated IMEI.

Request

Delete a single entry IMEI

This command is used to delete a single entry IMEI and all associated IMSIs.

The **`dlt_eir`** command defines the request message to delete a single entry IMEI.

Parameters:

imei A single IMEI.

Values: 14 or 15 hexadecimal digits expressed using ASCII characters. Only the first 14 digits of the IMEI are stored and displayed on retrieval.

timeout (Optional) Specify the number of seconds to wait for the write transaction if another connection already has it. Clients waiting for the write transaction with this mechanism are processed in the order that their requests were received. This option is only allowed if the client used the txnmode single option on its connect request.

Values: **0** (return immediately if not available; default)
1 - **3600** seconds

Rules

1. If the **imei** includes the optional 15th character (the check digit), the check digit is provided by the Customers Client Software and must match the EPAPs (via calculated algorithm). The check digit is not stored; it is only used to verify the **imei**.

Request syntex:

```
dlt_eir([iid XXXXX] imei XXXXX)
```

Delete a block of IMEIs

The **dlt_eir** command defines the request message to delete an IMEI block.

Parameters:

bimei The beginning IMEI in a block.

Values: 14 or 15 hexadecimal digits expressed using ASCII characters. Only the first 14 digits of the IMEI are stored and displayed on retrieval.

eimei The ending IMEI in a block.

Values: 14 or 15 hexadecimal digits expressed using ASCII characters. Only the first 14 digits of the IMEI are stored and displayed on retrieval.

timeout (Optional) Specify the number of seconds to wait for the write transaction if another connection already has it. Clients waiting for the write transaction with this mechanism are processed in the order that their requests were received. This option is only allowed if the client used the txnmode single option on its connect request.

Values: **0** (return immediately if not available; default)
1 - 3600 seconds

Rules

1. If the **imei** includes the optional 15th character (the check digit), the check digit is provided by the Customers Client Software and must match the EPAPs (via calculated algorithm). The check digit is not stored; it is only used to verify the **imei**.

Request syntex:

```
dlt_eir([iid XXXXX,] bimei XXXXX, eimei XXXXX)
```

Delete IMSI(s) from the associated IMEI

This command is used to delete the IMSI from the specified IMEI. This is the EIR specific IMSI, not the G-Port/G-Flex IMSI.

The **dlt_eir** command is used to delete the IMSI from the associated IMEI.

Parameters:

imei A single IMEI.

Values: 14 or 15 hexadecimal digits expressed using ASCII characters. Only the first 14 digits of the IMEI are stored and displayed on retrieval.

imsi The IMSI(s) to be associated with an IMEI.

Values: 5 to 15 hexadecimal digits expressed using ASCII characters. Up to 8 IMSIs can be provisioned for an IMEI.
all - used to remove all IMSIs associated withan IMEI.

timeout (Optional) Specify the number of seconds to wait for the write transaction if another connection already has it. Clients waiting for the write transaction with this mechanism are processed in the order that their requests were received. This option is only allowed if the client used the txnmode single option on its connect request.

Values: **0** (return immediately if not available; default)
1 - **3600** seconds

Request syntex:

```
dlt_eir([iid xxxxx,]imei XXXXX, imsi XXXXX[,….,imsi XXXXX])
```

Delete the IMSI from all IMEIs

This command is used to delete the IMSI from all IMEIs. This is the EIR specific IMSI, not the G-Port/G-Flex IMSI.

The **`dlt_eir`** command is used to delete the IMSI from all IMEIs.

Parameters:

imsi The IMSI(s) reference to be deleted from the IMEI.

Values: 5 to 15 hexadecimal digits expressed using ASCII characters. Up to 8 IMSIs can be provisioned for an IMEI.

timeout (Optional) Specify the number of seconds to wait for the write transaction if another connection already has it. Clients waiting for the write transaction with this mechanism are processed in the order that their requests were received. This option is only allowed if the client used the txnmode single option on its connect request.

Values: **0** (return immediately if not available; default)
1 - **3600** seconds

Request syntex:

```
dlt_eir([iid XXXXX,]imsi XXXXX)
```

Response

The return codes in Table 3-20 may result from the Delete IMEI request.

Table 3-20. Update IMEI Response Return Codes

Return Code	Description	Data Section Contents
SUCCESS	Everything worked.	NONE
INVALID_ VALUE	One of the fields specified had an invalid value.	The offending field is returned in the data section: `data (param <field label>)`
CHECK DIGIT ERROR	The check digit provisioned did not match the calculated check digit.	imei, bimei, or eimei data (param imei xxxxx)
NOT_FOUND	The requested IMEI or IMEI block was not found.	NONE
IMSI_NOT_ FOUND	IMSI not found on specified IMEI.	NONE
WRITE_IN_ READ_TXN	The **`delete`** command was sent on a **read** only transaction.	NONE
TXN_TOO_BIG	This request would cause current transaction to be larger than limit.	NONE
NO_UPDATES	The database already contains the data in the request. No update was necessary.	NONE
NO_ACTIVE_ TXN	There was no currently active transaction for this connection.	NONE
NO_WRITE_ PERMISSION	The PDBI client making request does not have write access permissions.	NONE
WRITE_ UNAVAIL	Another client already has a write transaction open.	IP address information of client that already has the write transaction. `data (id <connection id>, ip <ip addr>, port <port num>)`

Retrieve IMEI Data

This command displays the provisioned IMEI data.

Request

Retrieve all the data associated with a single IMEI entry

This command is used to retrieve the IMEI data specified. If the IMEI specified is not found in the individual entry table but resides in an IMEI block, then that IMEI block will be displayed.

The **rtrv_eir** command defines the request message to retrieve all the data associated with a single IMEI entry.

Parameters:

imei A single IMEI.

Values: 14 or 15 hexadecimal digits expressed using ASCII characters. Only the first 14 digits of the IMEI are stored and displayed on retrieval.

Rules

1. If the **imei** includes the optional 15th character (the check digit), the check digit is provided by the Customers Client Software and must match the EPAPs (via calculated algorithm). The check digit is not stored; it is only used to verify the **imei**.

Request syntex:

```
rtrv_eir([iid XXXXX,] imei XXXXX)
```

Retrieve a range of IMEIs

This command is used to retrieve either a range of individual IMEI(s) or IMEI blocks.

Parameters:

bimei The beginning IMEI in a block.

Values: 14 or 15 hexadecimal digits expressed using ASCII characters. Only the first 14 digits of the IMEI are stored and displayed on retrieval.

eimei The ending IMEI in a block.

Values: 14 or 15 hexadecimal digits expressed using ASCII characters. Only the first 14 digits of the IMEI are stored and displayed on retrieval.

type (Optional) IMEI blocks or single IMEI.

Values: **block** - Searches the IMEI Block table. Only reports IMEI Blocks, regardless of whether any provisioned Single IMEIs fall within the specified number range.
single (default) - Searches the Single IMEI table. Only IMEIs provisioned as Single IMEIs are returned, regardless of whether the IMEI number falls within the number range of a provisioned IMEI block.

white (Optional) Filters request to retrieve the IMEIs found on the White list.

Values: **yes** or **no** (default = **no filter**).

gray (Optional) Filters request to retrieve the IMEIs found on the Gray list.

Values: **yes** or **no** (default = **no filter**).

black (Optional) Filters request to retrieve the IMEIs found on the Black list.

Values: **yes** or **no** (default = **no filter**).

imsi (Optional) Filters request to retrieve the IMEIs on the specified IMSI.

Values: 5 to 15 hexadecimal digits expressed using ASCII characters (default= **none**). Only valid when type is **single**.

data (Optional) Specifies type of output data.

Values: **all** (default) - Returns all known data for each instance.
count - Return a single instance count of all instances matching the query.

num (Optional) Limits number of entities returned. If omitted, all entities are returned.

Values: **0 - 40000000**

Rules

1. If the **imei** includes the optional 15th character (the check digit), the check digit is provided by the Customers Client Software and must match the EPAPs (via calculated algorithm). The check digit is not stored; it is only used to verify the **imei**.

Request syntex:

```
rtrv_eir([iid XXXXX,] bimei XXXXX, eimei XXXXX, [type
<block/single>,] [white <yes/no>,] [gray <yes/no>,] [black
<yes/no>,] [imsi XXXXX] [data <all/count>] [num 0..40000000])
```

Response

The syntax of the data section of responses to a successful Retrieve IMEI request depends on the type of records being returned. Both single IMEI and range IMEIs data is supported. Each type of data being returned has a different syntax.

- Response syntax for an IMEI single query:

```
data (segment XXXXX,

imeis (imei(id XXXXX, svn ##, white yes/no, gray yes/no, black
yes/no, [imsis (xxxxx, …  xxxxx….)),

. . .

( . . . ) ) )
```

- Response syntax for an IMEI block query:

```
data (segment XXXXX,

meiblock (imeiblock (bimei xxxxx, eimei xxxxx, white yes/no,
gray yes/no, black yes/no, ),

. . .

( . . . ) ) )
```

- Response syntax for an IMEI count query:

```
data (counts (imei #####)

data (counts (imeiblock #####)
```

The return codes listed in Table 3-21 indicate the result of the Retrieve IMEI request.

Table 3-21. Retrieve IMEI Response Return Codes

Return Code	Description	Data Section Contents
SUCCESS	Everything worked.	NONE
INVALID_VALUE	One of the fields specified had an invalid value.	Offending field is returned in data section: `data (param <field label>)`
IMSI_NOT_FOUND	The IMSI requested as part of the filter does not exist .	NONE
CHECK DIGIT ERROR	The check digit provisioned did not match the calculated check digit.	imei, bieme or eimei data (param imei xxxxx)
PARTIAL_SUCCESS	The request has succeeded, but this is only one of many responses.	Depends on the request type, etc.
NOT_FOUND	IMEI (block) not found.	NONE
NO_ACTIVE_TXN	There is no currently active transaction for this connection.	NONE

Request DSM Report

This command is used to retrieve the DSM Report in a synchronous manner. This command is not required to be sent from inside a transaction.

Request

Retrieve the DSM Report

There are two parameters that adjust what percent or level to run the report for. The two parameters are mutually exclusive. If neither is specified, then the report will be run with the default percent value for the connection. The caller can also specify whether or not they want the DSM exception list.

The **`rtrv_dsmrpt`** command defines the request message to retrieve the DSM report data.

Parameters:

percent (Optional) The percent to use for this one report. Cannot be specified with level.

Values: **1 – 100**

level (Optional) Specific database level to use for this one report. Cannot be specified with percent.

Values: **1 – 4294967295**

data (Optional) Lets the requestor specify whether or not they want to the see list of DSM cards that were not at the main database level mentioned in the report.

Values: **status** (default) - Return the list of DSM cards.
none – Do not return the list of DSM cards.

Request syntex:

```
rtrv_dsmrpt([iid XXXXX,] [percent ###], [level #####], [data
<none/except>])
```

Response

The data section of a successful DSM report request contains the following information:

Parameters:

segment This parameter contains the message segment number.
Values: ≥ 1 – Incrementing integer starting from 1.

level The database level that the report is referring to. The DSM cards that satisfy the report have levels equal to or greater than this value.
Values: **0 – 4294967295**

percent The percentage of known DSM cards that meet or exceed the level specified.
Values: **0 – 100**

numdsmsThe total number of known DSM cards in the customer's network.
Values: **0 – ####**

dsms List of DSM cards that did not have a database level equal to or larger than main message's level. Each DSM element contains the following information.

clli Identifier for the DSM card's EAGLE 5 SAS node.
Values: String up to 11 characters long

cardloc Location identifier for the DSM card in the EAGLE 5 SAS node.
Values: Four digit number

status The database status of the DSM card.
Values: **loading** - The card is currently loading the database.
resync - The card is loaded, but catching up to current provisioning stream.
coherent - The database is loaded and receiving normal provisioning
incoherent - Internal error on DSM card (write failed to database)
inconsistent - Data mismatch between EPAP RTDB and DSM RTDB.
corrupt - Internal error on DSM card checksum failure).
level - The database level for this card.
Values: **0 – 4294967295**
loadperc - The percent of the database that has been loaded during initial booting of the card. This field is only meaningful when the status is loading, so it will only appear then.
Values: **0 – 100**

```
rsp([iid XXXX,] rc 0, data (segment ###, level ####, percent <0..100>,
numdsms ####,
    dsms (
     dsm (clli AAAA, cardloc ####, status <values below>, level ####
[, loadperc <0..100>]),
     . . .
     dsm (. . .) ) )
```

The return codes listed in Table 3-22 indicate the result of the Retrieve DSM report request.

Table 3-22. Retrieve DSM Report Response Return Codes

Return Code	Description	Data Section Contents
SUCCESS	Everything worked.	See above.
INVALID_VALUE	One of the fields specified had an invalid value.	Offending field is returned in data section: `data (param <field label>)`
PARTIAL_SUCCESS	The request has succeeded, but this is only one of many responses.	Depends on the request type, etc.

Request

Retrieve a list the DSM cards

This command is used to retrieve all or a subset of the DSM cards known to the PDBA. It does not need to be sent from inside a transaction. This request is different that the DSM Report in that it does not attempt to determine any percent complete at a given level. It simply returns all of the DSM cards that meet the filter criteria.

The **`rtrv_dsmlist`** command defines the request message to retrieve the DSM card data.

Parameters:

clli (Optional) Retrieve only the DSM cards on the specified EAGLE 5 SAS node.

Values: 1 - 11 alphanumeric characters, hyphen, or underscore.

cardloc (Optional) Retrieve only the DSM cards that are in the specified card location.

Values: Four digit number.

status (Optional) Retrieve only the DSM cards that have the specified database status.

Values: **loading** - The card is currently loading the database.
resync - The card is loaded, but catching up to current provisioning stream.
coherent - The database is loaded and receiving normal provisioning
incoherent - Internal error on DSM card (i.e. write failed to database)
inconsistent - Data mismatch between EPAP RTDB and DSM RTDB.
corrupt - Internal error on DSM card (checksum failure)

Request syntex:

```
rtrv_dsmlist([iid XXXXX,] [clli XXXX], [cardloc ####], [status
<value list above>])
```

Response

The data section of a successful PDBA Status Query request contains the following information:

Parameters:

segment This parameter contains the message segment number.
Values: ≥ 1 - Incrementing integer starting from 1.

dsms List of DSM cards that did not have a database level equal to or larger than main message's level. Each DSM element contains the following information.

clli Identifier for the DSM card's EAGLE 5 SAS node.
Values: String up to 11 characters long

cardloc Location identifier for the DSM card in the EAGLE 5 SAS node.
Values: Four digit number

status The database status of the DSM card.
Values: **loading** - The card is currently loading the database.
resync - The card is loaded, but catching up to current provisioning stream.
coherent - The database is loaded and receiving normal provisioning
incoherent - Internal error on DSM card (write failed to database)
inconsistent - Data mismatch between EPAP RTDB and DSM RTDB.
corrupt - Internal error on DSM card checksum failure).
level - The database level for this card.
Values: **0 – 4294967295**
loadperc - The percent of the database that has been loaded during initial booting of the card. This field is only meaningful when the status is loading, so it will only appear then.
Values: **0 – 100**

```
rsp([iid XXXX,] rc 0, data (segment ###, level ####, percent
<0..100>, numdsms ####,
    dsms (
     dsm (clli AAAA, cardloc ####, status <values below>, level ####
[, loadperc <0..100>]),
     . . .
     dsm (. . .) ) )
```

The return codes listed in Table 3-23 indicate the result of the Retrieve DSM list report request.

Table 3-23. Retrieve DSM List Response Return Codes

Return Code	Description	Data Section Contents
SUCCESS	Everything worked.	See above.
NOT_FOUND	There were no DSM cards found. If one or more filters were specified, then there were no cards that matched the filter	NONE
INVALID_VALUE	One of the fields specified had an invalid value.	Offending field is returned in data section: `data (param <field label>)`
PARTIAL_SUCCESS	The request has succeeded, but this is only one of many responses.	Depends on the request type, etc.

4

PDBI Sample Sessions

Introduction 4–2
Network Entity Creation 4–2
Simple Subscription Data Creation 4–3
Update Subscription Data 4–4
Simple Queries 4–5
Multiple Response Query 4–7
Abort Transaction 4–8
Update Request In Read Transaction 4–9
Write Transaction In Standby Connection 4–10
Simple Subscription Data Creation with Single Txnmode 4–11
Single IMEI Data 4–12
IMEI Block Data 4–13
Asynchronous DSM Report 4–14
Synchronous DSM Report 4–15
DSM List 4–16

Introduction

This chapter contains sample usages of the PDBI. The message exchanges are shown in Tables 4-1 to 4-14. All scenarios assume that a TCP/IP connection has already been established between the client and the PDBA.

The first column in the tables shows the direction the message is going.

- Messages going from the client to the PDBA (requests) are indicated by →.
- Messages going from the PDBA to the client (responses) are indicated by ←.

The strings displayed in the Message column are the actual ASCII that would flow over the socket.

Network Entity Creation

This example connects to the PDBA and creates the Network Entities that are needed for all subsequent examples.

Table 4-1. Network Entity Creation Example

	Message	Description
→	connect (iid 1, version 1.0)	A PDBI connection has been established to the Active PDBA.
←	rsp (iid 1, rc 0, data (connectId 1, side active))	
→	begin_txn(iid 2, type write)	A **write** transaction has been opened.
←	rsp (iid 2, rc 0)	
→	ent_entity (iid 3, id 9195550000, type SP, pctype ansi, pc 123-456-789, ri GT, ntt 222, ccgt no)	The SP Network Entity for SP 9195550000 has been created.
←	rsp (iid 3, rc 0)	
→	ent_entity (iid 4, id 9195555555, type SP, pctype intl, pc 1-234-5, ri SSN, ssn 32, ccgt no)	The SP Network Entity for SP 9195555555 has been created.
←	rsp (iid 4, rc 0)	
→	ent_entity (iid 5, id 9195556666, type SP, pctype natl, pc 12345, ri GT, ccgt no)	The SP Network Entity for SP 9195556666 has been created.
←	rsp (iid 5, rc 0)	
→	end_txn (iid 6)	The **write** transaction has been ended. The updates have been written to the PDB and will be sent to the EAGLE 5 SAS.
←	rsp (iid 6, rc 0, data (dblevel 1))	
→	disconnect (iid 7)	The client is done and has disconnected.
←	rsp (iid 7, rc 0)	

Simple Subscription Data Creation

This example shows a normal connection with the creation of a few different kinds of subscriptions.

Table 4-2. Simple Subscription Data Creation Example

	Message	Description
→	connect (iid 1, version 1.0)	A PDBI connection has been established.
←	rsp (iid 1, rc 0, data (connectId 1, side active))	
→	begin_txn (iid 2, type write)	A **write** transaction has been opened.
←	rsp (iid 2, rc 0)	
→	ent_sub (iid 3, imsi 9195551000, dn 9195551212, dn 9195551213, sp 9195550000)	A multi-DN subscription has been created for IMSI 919551000. The DNs associated with the IMSI are 9195551212 and 9195551213. The subscription is on SP 9195550000.
←	rsp (iid 3, rc 0)	
→	ent_sub (iid 4, imsi 9195552000, sp 9195550000)	A IMSI-only subscription has been created. The IMSI is 9195552000. The subscription is on SP 9195550000.
←	rsp (iid 4, rc 0)	
→	ent_sub (iid 4, imsi 9195552001, sp 9195550000)	Another IMSI-only subscription has been created. The IMSI is 9195552001. The subscription is on SP 9195550000. This IMSI will be used in a later example.
←	rsp (iid 4, rc 0)	
→	ent_sub (iid 5, dn 9195551500, dn 9195551501, dn 9195551502, dn 9195551503, sp 9195555555)	Four separate single DN subscriptions were created. All four DNs are on SP 9195555555.
←	rsp (iid 5, rc 0)	
→	end_txn (iid 6)	The **write** transaction has been ended. The updates have been written to the PDB and will be sent to the EAGLE 5 SAS.
←	rsp (iid 6, rc 0, data (dblevel 1))	
→	disconnect (iid 7)	The client is done and has disconnected.
←	rsp (iid 7, rc 0)	

Update Subscription Data

This example shows how to:

- Add new DNs to an existing IMSI
- Move all of the records for a multi-dn IMSI to a new SP
- Move one or more existing stand-alone DNs to a new SP, and
- Move existing DNs to an existing IMSI.

Table 4-3. Update Subscription Data Example

	Message	Description
→	connect (iid 1, version 1.0)	A PDBI connection has been established to the Active PDBA.
←	rsp (iid 1, rc 0, data (connectId 3, side active))	
→	begin_txn (iid 2, type write)	A **write** transaction has been opened.
←	rsp (iid 2, rc 0)	
→	ent_sub (iid 3, imsi 9195551000, dn 9195551214, dn 9195551215, sp 9195550000)	Two new DNs are being added to the existing IMSI 9195551000. The IMSI already had two DNs (9195551212 and 9195551213) from the previous creation example scenario, giving it a total of four DNs.
←	rsp (iid 3, rc 0)	
→	upd_sub (iid 4, imsi 9195551000, sp 9195556666)	This command moves the specified IMSI and its four DNs to the SP 9195556666.
←	rsp (iid 4, rc 0)	
→	upd_sub (iid 5, dn 9195551502, dn 9195551503, sp 9195550000)	This command moves the two specified single DNs to SP 9195550000.
←	rsp (iid 5, rc 0)	
→	upd_sub (iid 6, dn 9195551501, imsi 9195552001)	This command moves the specified standalone DN to be associated with the specified IMSI. The DN will now get its SP from the IMSI.
←	rsp (iid 6, rc 0)	
→	end_txn (iid 7)	The **write** transaction has been ended. The updates will be written to the database.
←	rsp (iid 7, rc 0, data (dblevel 2))	
→	disconnect (iid 8)	The client is done and has disconnected.
←	rsp (iid 8, rc 0)	

Simple Queries

This example shows a **`read`** transaction that queries the data populated in a previous example.

Table 4-4. Simple Queries Example

	Message	Description
→	connect (iid 1, version 1.0)	A PDBI connection has been established to the Active PDBA.
←	rsp (iid 1, rc 0, data (connectId 4, side active))	
→	begin_txn (iid 2, type read)	A **read** transaction has been opened.
←	rsp (iid 2, rc 0)	
→	rtrv_sub (iid 3, bimsi 9195550000, eimsi 9195559999)	A query for all IMSIs within the range from 9195550000 to 9195559999 was sent. This query does not contain the **data** parameter; the default value of **all** is used. This means that the list of DNs associated with IMSI should be present in the response. The response that comes back has just the three IMSIs that were created and updated in the previous examples.
←	rsp (iid 3, rc 0, data (imsis ((imsi 9195551000, dns(9195551212, 9195551213, 9195551214, 9195551215), sp 9195556666), (imsi 9195552000, sp 9195550000), (imsi 9195552001, dns(9195551501), sp 9195550000))))	
→	rtrv_sub (iid 4, bimsi 9195550000, eimsi 9195559999, data neonly)	This query is almost the same as the one above. This difference is that this one specifies the value **neonly** for the **data** parameter. That means that the list of DNs will be omitted from the IMSI information.
←	rsp (iid 4, rc 0, data (imsis ((imsi 9195551000, sp 9195556666), (imsi 9195552000, sp 9195550000), (imsi 9195552001, sp 9195550000))))	
→	rtrv_sub (iid 5, bimsi 9195550000, eimsi 9195559999, sp 9195550000, data neonly)	This query is almost the same as the two above. In addition to specifying the **neonly** value, this one also provides an **sp** parameter to filter for only IMSIs on the specified SP.
←	rsp (iid 5, rc 0, data (imsis ((imsi 9195552000, sp 9195550000), (imsi 9195552001, sp 9195550000))))	
→	rtrv_sub (iid 6, bdn 9195550000, edn 9195559999)	A query for all DNs within the range from 9195550000 to 9195559999 was sent. The query does not contain the **data** parameter, so the default value of **all** is used. This means that the IMSI value for each DN (if not a stand-alone DN) will be present in the response. The response that comes back has just the eight DNs that were created and updated in the previous examples.
←	rsp (iid 6, rc 0, data (dns ((dn 9195551212, imsi 9195551000, sp 9195556666), (dn 9195551213, imsi 9195551000, sp 9195556666), (dn 9195551214, imsi 9195551000, sp 9195556666), (dn 9195551215, imsi 9195551000, sp 9195556666), (dn 9195551500, sp 9195555555), (dn 9195551501, imsi 9195552001, sp 9195550000), (dn 9195551502, sp 9195550000), (dn 9195551503, sp 9195550000))))	

Table 4-4. Simple Queries Example (Continued)

	Message	Description
→	rtrv_sub(iid 7, bdn 9195550000, edn 9195559999, sp 9195556666, data neonly)	This query is almost the same as the one above. The differences are that it specifies both the neonly value and it provides an sp parameter to filter only for DNs on the specified SP.
←	rsp (iid 7, rc 0, data (dns ((dn 9195551212, sp 9195556666), (dn 9195551213, sp 9195556666), (dn 9195551214, sp 9195556666), (dn 9195551215, sp 9195556666))))	
→	end_txn(iid 8)	The **read** transaction has been ended.
←	rsp (iid 8, rc 0)	
→	disconnect(iid 8)	The client is done and has disconnected.
←	rsp (iid 8, rc 0)	

Multiple Response Query

This example shows a Retrieve command that results in multiple responses coming back. This would happen when there are so many subscriptions matching the query that a single response would be too big to handle. The single response is broken into many smaller responses. The real response size limit is 4KB, but for the purposes of this example it is much smaller.

Table 4-5. Multiple Response Query Example

	Message	Description
→	connect(iid 1, version 1.0)	A PDBI connection has been established to the Active PDBA.
←	rsp (iid 1, rc 0, data (connectId 5, side active))	
→	begin_txn(iid 2, type read)	A **read** transaction has been opened.
←	rsp (iid 2, rc 0)	
→	rtrv_sub(iid 3, bdn 9195550000, edn 9195559999)	A query for all single DNs within the range from 919550000 to 9195559999 was sent. For this example, the result information will come back in three separate responses.
←	rsp (iid 3, rc 1016, data (segment 1, dns (dn 9195551212, imsi 9195551000 sp 9195556666), (dn 9195551213, imsi 9195551000, sp 9195556666), (dn 9195551214, imsi 9195551000, sp 9195556666))))	
←	rsp (iid 3, rc 1016, data (dns (segment 2, dns ((dn 9195551215, imsi 9195551000, sp 9195556666), (dn 9195551500, sp 9195555555), (dn 9195551501, imsi 9195552001, sp 9195550000))))	
←	rsp (iid 3, rc 0, data (dns (segment 3, dns ((dn 9195551502, sp 9195550000), (dn 9195551503, sp 9195550000))))	
→	end_txn(iid 4)	The **read** transaction has been ended.
←	rsp (iid 4, rc 0)	
→	disconnect(iid 5)	The client is done and has disconnected.
←	rsp (iid 5, rc 0)	

Abort Transaction

This example shows a **write** transaction that receives an error on one of its update requests and then aborts the transaction.

Table 4-6. Abort Transaction Example

	Message	Description
→	connect(iid 1, version 1.0)	A PDBI connection has been established to the Active PDBA.
←	rsp (iid 1, rc 0, data (connectId 6, side active))	
→	begin_txn(iid 2, type write)	A **write** transaction has been opened.
←	rsp (iid 2, rc 0)	
→	ent_sub(iid 3, dn 9195557000, sp 9195556666)	The DN has been created.
←	rsp (iid 3, rc 0)	
→	ent_sub(iid 4, dn 9195558000, sp 9195550000)	Another DN has been created.
←	rsp (iid 4, rc 0)	
→	ent_sub(iid 5, dn 9195551213, sp 919555555)	The request to create a stand-alone DN 9195551213 failed because the DN already exists.
←	rsp (iid 5, rc 1014, data (dn 9195551213))	
→	abort_txn(iid 6)	The client decided to abort the transaction because the previous update failed. This will cause the two DNs created in **iid** 3 and **iid** 4 to be rolled back. No data is updated. Note that the client did not have to abort the transaction here. The transaction could have just been ended normally, and the first two DNs would have been created successfully.
←	rsp (iid 6, rc 0)	
→	disconnect(iid 7)	The client is done and has disconnected.
←	rsp (iid 7, rc 0)	

Update Request In Read Transaction

This example shows a client opening a **read** transaction and then trying to send a command to modify data.

Table 4-7. Update Request in Read Transaction Example

	Message	Description
→	connect(iid 1, version 1.0)	A PDBI connection has been established to the Active PDBA.
←	rsp (iid 1, rc 0, data (connectId 7, side active))	
→	begin_txn(iid 2, type read)	A **read** transaction has been opened.
←	rsp (iid 2, rc 0)	
→	rtrv_sub(iid 3, dn 9195551500)	A query for DN 9195551500 was sent. A response comes back verifying that the DN exists and showing what SP it is on.
←	rsp (iid 3, rc 0, data (dns ((dn 9195551500, sp 9195555555))))	
→	upd_sub(iid 4, dn 9195551500, sp 9195556666)	The client now tries to move the DN block that was returned in the previous Retrieve request to SP 9195556666. The Update request fails because the client currently has a **read** transaction open instead of a **write** transaction.
←	rsp (iid 4, rc 1011)	
→	end_txn(iid 5)	The **read** transaction has been ended.
←	rsp (iid 5, rc 0)	
→	disconnect(iid 6)	The client is done and has disconnected.
←	rsp (iid 6, rc 0)	

Write Transaction In Standby Connection

This example shows the error scenario of a client trying to open a **write** transaction in a connection to the Standby PDBA.

Table 4-8. Write Transaction in Standby Connection Example

	Message	Description
→	connect(iid 1, version 1.0)	A PDBI connection has been established to the Standby PDBA.
←	rsp (iid 1, rc 0, data (connectId 8, side standby))	
→	begin_txn(iid 2, type write)	The client has attempted to open a **write** transaction on the Standby PDBA. An error is returned. There is no need to end the transaction because it was never successfully started.
←	rsp (iid 2, rc 1006)	
→	disconnect(iid 3)	The client is done and has disconnected.
←	rsp (iid 3, rc 0)	

Simple Subscription Data Creation with Single Txnmode

This example shows a connection using the **txnmode** single connect option with the creation of a few different kinds of subscriptions..

Table 4-9. Simple Subscription Data Creation with Single Txnmode Example

	Message	Description
→	connect(iid 1, version 1.0, txnmode single)	A PDBI connection has been established.
←	rsp (iid 1, rc 0, data (connectId 1, side active))	
→	ent_sub(iid 2, imsi 9195551000, dn 9195551212, dn 9195551213, sp 9195550000, timeout 10)	A multi-dn subscription has been created for IMSI 919551000. The DNs associated with the IMSI are 9195551212 and 9195551213. The subscription is on SP 9195550000.
←	rsp (iid 2, rc 0, data (dblevel 1))	
→	ent_sub(iid 3, imsi 9195552000, sp 9195550000)	A IMSI only subscription has been created. The IMSI is 9195552000. The subscription is on SP 9195550000.
←	rsp (iid 3, rc 0, data (dblevel 2))	
→	ent_sub(iid 4, imsi 9195552001, sp 9195550000)	A IMSI only subscription has been created. The IMSI is 9195552001. The subscription is on SP 9195550000. This is exactly the same type of command as the previous item. It is being done so that the IMSI can be used in a later example.
←	rsp (iid 4, rc 0, data (dblevel 3))	
→	ent_sub(iid 5, dn 9195551500, dn 9195551501, dn 9195551502, dn 9195551503, sp 9195555555)	Four separate single DN subscriptions were created. All four DNs are on SP 9195555555.
←	rsp (iid 5, rc 0, data (dblevel 4))	
→	disconnect(iid 6)	The client is done and has disconnected.
←	rsp (iid 6, rc 0)	

Single IMEI Data

This example shows a normal connection with the creation, update and deletion of a few different kinds of IMEIs..

Table 4-10. Single IMEI Data Example

	Message	Description
→	connect(iid 1, version 1.0)	A PDBI connection has been established to the Active PDBA.
←	rsp (iid 1, rc 0, data (connectId 1, side active))	
→	begin_txn(iid 2, type write)	A **write** transaction has been opened.
←	rsp (iid 2, rc 0)	
→	ent_eir(iid 3, imei 12345678901234, white yes, gray yes, imsi 9199301234, imsi 9199302266)	A single IMEI is created on both the white and gray lists with 2 IMSIs associated with it. SVN is 0.
←	rsp (iid 3, rc 0)	
→	upd_eir(iid 4, imei 12345678901234, white no, black yes)	The lists associated with the IMEI is now changed to be Black and Gray. The White list is now turned off.
←	rsp (iid 4, rc 0)	
→	dlt_eir(iid 5, imei 12345678901234, imsi 9199302266)	The specified IMSI is no longer associated with the IMEI.
←	rsp (iid 5, rc 0)	
→	dlt_eir(iid 6, imei 12345678901234)	The IMEI and it's associated IMSI are removed.
←	rsp (iid 6, rc 0)	
→	end_txn(iid 7)	The **write** transaction has ended. The updates will be written to the database.
←	rsp (iid 7, rc 0, data (dblevel 1))	
→	disconnect(iid 8)	The client is done and has disconnected.
←	rsp (iid 8, rc 0)	

IMEI Block Data

This example shows a normal connection with the creation, update and deletion of a few different kinds of IMEI blocks.

Table 4-11. IMEI Block Data Example

	Message	Description
→	connect(iid 1, version 1.0)	A PDBI connection has been established to the Active PDBA.
←	rsp (iid 1, rc 0, data (connectId 1, side active))	
→	begin_txn(iid 2, type write)	A **write** transaction has been opened.
←	rsp (iid 2, rc 0)	
→	ent_eir(iid 3, bimei 12345678901000, eimei 12345678901999, black yes)	An IMEI Block is created with the black list.
←	rsp (iid 3, rc 0)	
→	upd_eir(iid 4, bimei 12345678901000, eimei 12345678901999, gray yes)	The lists associated with the IMEI block are now changed to be Black and Gray. Note: the black list was turned on in the previous step
←	rsp (iid 4, rc 0)	
→	dlt_eir(iid 5, bimei 12345678901000, eimei 12345678901999)	The IMEI block is removed.
←	rsp (iid 5, rc 0)	
→	end_txn(iid 6)	The **write** transaction has been ended. The updates have been written to the PDB and will be sent to the EAGLE 5 SAS.
←	rsp (iid 6, rc 0, data (dblevel 1))	
→	disconnect(iid 7)	The client is done and has disconnected.
←	rsp (iid 7, rc 0)	

Asynchronous DSM Report

This example shows a connection that has asked to receive the DSM Report every 10 seconds with a report complete percent of 90..

Table 4-12. Asynchronous DSM Report Example

	Message	Description
→	connect(version 1.0, dsmrpt yes, dsmrptfreq 10, dsmrptperc 90)	A PDBI connection has been established. The connection has asked for DSM Reports every 10 second with a report complete percent of 90.
←	rsp (rc 0, data (connectId 1, side active))	
→		No requests sent. 10 seconds later, a report comes out. One DSM is not up with the others because it is loading. Another card is excluded because it is corrupt.
←	dsmrpt (rc 0, data (segment 1, level 2912, percent 90, numdsms 20, dsms (dsm (clli atlanta, cardloc 1405, status loading, level 0, loadperc 98), dsm (clli lakemary, cardloc 2104, status corrupt, level 2652))))	
→		10 seconds later, another report comes out. Now, the loading card is finished, but the corrupt card is still corrupt.
←	dsmrpt (rc 0, data (segment 1, level 2917, percent 95, numdsms 20, dsms (dsm (clli lakemary, cardloc 2104, status corrupt, level 2652))))	
→	disconnect(iid 7)	The client is done and has disconnected.

Synchronous DSM Report

This example shows a connection that uses the rtrv_dsmrpt request to receive the DSM Report.

Table 4-13. Synchronous DSM Report Example

	Message	Description
→	connect(version 1.0)	A normal PDBI connection has been established. At this point, it has not expressed any desire to get a DSM Report.
←	rsp (rc 0, data (connectId 1, side active))	
→	rtrv_dsmrpt(percent 90)	The connection requests a DSM Report with a specific percent complete value of 90. The level 2912 is returned because only two DSM cards out of the known 20 have not reached that level.
←	rsp (rc 0, data (segment 1, level 2912, percent 90, numdsms 20, dsms (dsm (clli atlanta, cardloc 1405, status coherent, level 2910), dsm (clli lakemary, cardloc 2104, status corrupt, level 2652))))	
→	rtrv_dsmrpt(percent 95)	The connection requests the DSM Report again, this time with a more stringent percent complete value of 95. This time, a smaller level of 2910 is returned in order to satisfy the higher percent value.
←	rsp (rc 0, data (segment 1, level 2910, percent 95, numdsms 20, dsms (dsm (clli lakemary, cardloc 2104, status corrupt, level 2652))))	
→	disconnect(iid 7)	The client is done and has disconnected.

DSM List

This example shows a connection requesting to see the list of DSM cards. There are only five cards in this example to make the responses smaller..

Table 4-14. DSM List Example

	Message	Description
→	connect(version 1.0)	A normal PDBI connection has been established.
←	rsp (rc 0, data (connectId 1, side active))	
→	rtrv_dsmlist()	The connection requests to see the DSM list. No filter is used, so all DSM cards should be returned.
←	rsp (rc 0, data (dsms (dsm (clli atlanta, cardloc 1405, status coherent, level 2910), dsm (clli atlanta, cardloc 1407, status coherent, level 2918), dsm (clli lakemary, cardloc 2104, status corrupt, level 2652), dsm (clli lakemary, cardloc 2106, status coherent, level 2920), dsm (clli lakemary, cardloc 2108, status coherent, level 2920))))	
→	rtrv_dsmlist(clli atlanta)	The connection requests the DSM list with a filter for CLLIs with the value of "atlanta".
←	rsp (rc 0, data (dsms (dsm (clli atlanta, cardloc 1405, status coherent, level 2910), dsm (clli atlanta, cardloc 1407, status coherent, level 2918))))	
→	rtrv_dsmlist(clli atlanta, cardloc 1407)	The connection requests the DSM list with a filter for CLLIs with the value of "atlanta" and a cardloc of 1407.
←	rsp (rc 0, data (dsms (dsm (clli atlanta, cardloc 1407, status coherent, level 2918))))	
→	rtrv_dsmlist(status corrupt)	The connection requests to see the DSM list with a status filter of corrupt. Only corrupt cards should be returned.
←	rsp (rc 0, data (dsms (dsm (clli lakemary, cardloc 2104, status corrupt, level 2652))))	
→	rtrv_dsmlist(status loading)	The connection requests to see the DSM list with a status filter of loading. Since there aren't any cards currently loading, NOT_FOUND is returned.
←	rsp (rc 1013)	
→	disconnect(iid 7)	The client is done and has disconnected.

A

PDBI Message Error Codes

Table A-1 lists the error codes and associated text generated by the PDBI.

Table A-1. PDBI Message Error Codes

Error Code	Text
0	PDBI_SUCCESS =0
1001	PDBI_INTERNAL_ERROR = 1001
1002	PDBI_NOT_CONNECTED
1003	PDBI_ALREADY_CONNECTED
1004	PDBI_PARSE_FAILED
1005	PDBI_WRITE_UNAVAIL
1006	PDBI_NO_WRITE_PERMISSION
1007	PDBI_NO_MATE
1008	PDBI_STANDBY_SIDE
1009	PDBI_NO_ACTIVE_TXN
1010	PDBI_ACTIVE_TXN
1011	PDBI_WRITE_IN_READ_TXN
1012	PDBI_INVALID_VALUE
1013	PDBI_NOT_FOUND
1014	PDBI_CONFLICT_FOUND
1015	PDBI_ITEM_EXISTS
1016	PDBI_PARTIAL_SUCCESS
1017	PDBI_NO_UPDATES
1018	PDBI_INTERRUPTED
1019	PDBI_BAD_ARGS
1020	PDBI_TOO_MANY_CONNECTIONS

Table A-1. PDBI Message Error Codes

Error Code	Text
1021	PDBI_NE_NOT_FOUND
1022	PDBI_CONTAINS_SUBS
1023	PDBI_UNKNOWN_VERSION
1024	PDBI_OUT_OF_MEMORY
1025	PDBI_UNIMPLEMENTED
1026	PDBI_MATE_BUSY
1027	PDBI_IMSI_DN_LIMIT
1028	PDBI_BAD_IMPORT_CMD
1029	PDBI_TXN_TOO_BIG
1030	PDBI_DB_MAINT_REQD
1031	PDBI_DB_EXCEPTION
1032	PDBI_MAX_IMSI_LIMIT
1033	PDBI_MAX_DN_LIMIT
1034	PDBI_MAX_DNBLK_LIMIT
1035	PDBI_MAX_NE_LIMIT
1036	PDBI_REPLICATING
1037	PDBI_CHECK_DIGIT_ERROR
1038	PDBI_IMSI_NOT_FOUND
1039	PDBI_IMEI_IMSI_LIMIT
1040	PDBI_MAX_IMEI_LIMIT
1041	PDBI_MAX_IMEI_BLK_LIMIT
1042	PDBI_NO_LIST_FOR_IMEI

Index

A

Abort Transaction, 3-14
  command, 3-14
  PDBI sample session, 4-8
  request message, 3-15
  response message, 3-15
  response return codes, 3-15
abort_txn command, 3-14
ACK, 1-10
Acronyms, 1-10
admonishments, documentation, 1-8
ANSI, 1-10
API, 1-10
architecture, PDBI/PDBA/EPAP system, 2-6
ASCII, 1-10
ASM, 1-10
Asynchronous DSM Report, 4-14
asynchronous replication, 2-7, 3-48
atomic command, 2-12

B

Bad Arguments
  response message, 3-6
  return code, 3-6
bdn, 3-20, 3-25, 3-29, 3-31
Begin Transaction, 3-11
  command, 3-11
  request message, 3-11
  response message, 3-13
  response return codes, 3-13
begin_txn command, 3-11
bid, 3-46
bimsi, 3-34
block instance table, 2-5

C

CCGT, 1-10
ccgt, 3-38, 3-42
CD-ROM, 1-10
client connections, 2-10, 3-8
close, 3-8
command
  abort_txn, 3-14
  begin_txn, 3-11
  connect, 3-8
  disconnect, 3-10
  dlt_entity, 3-44
  dlt_sub, 3-28
  dump_conn, 3-52
  end_txn, 3-13
  ent_entityn, 3-36
  ent_subn, 3-15
  rtrv_entity, 3-46
  rtrv_subn, 3-30
  status, 3-51
  switchover, 3-48
  upd_entity, 3-40
  upd_subn, 3-22
Command Atomicity, 2-12
concurrent client connections, 2-10, 3-8
Connect, 3-8
  command, 3-8
  request message, 3-8
  response message, 3-10
  response return codes, 3-10
connect command, 3-8
connection
  between PDBA and PDBI client, 3-8
  idle time, 3-9
  limit of PDBI clients, 2-10
  PDBA, dump information for debug, 3-52
  security, PDBI, 2-11
  socket-based, 2-10
connections
  concurrent clients, 2-10, 3-8
CPS, 1-10
CPU, 1-10
Create a block entry of IMEIs, 3-55
Create a new IMSI and associate it with an existing IMEI, 3-56
Create a single entry IMEI, 3-53
Create IMEI Data, 3-53
Create Network Entity, 3-36
  command, 3-36
  PDBI sample session, 4-2
  request message, 3-36
  response message, 3-39

response return codes, 3-39
Create Subscription, 3-15
command, 3-15
PDBI sample session, 4-3
request message
one IMSI and one to eight DNs, 3-16
one or more DNs on same NE with no IMSI, 3-17
porting a block of DNs, 3-20
single IMSI with no DN, 3-15
response message, 3-22, 3-57, 3-60, 3-64
response return codes, 3-22, 3-57, 3-60, 3-64
Customer Support Center, 1-9

D

DA, 1-10
da, 3-38, 3-42
Data, 3-2
data, 3-31, 3-32, 3-33, 3-34, 3-46
data element, format in response messages, 3-3
Data Model on Platform, 2-4
Data Organization, 2-5
database
provisioning, 2-7
standby, 2-7
Database Services Module, 2-2
DB, 1-10
DCB, 1-10
DCM, 1-10
debug log, PDBA file format, 2-15
debug, dump PDBA information for, 3-52
default limit to connections, 2-10, 3-8
Delete a block of IMEIs, 3-62
Delete a single entry IMEI, 3-61
Delete IMSI(s) from the associated IMEI, 3-62
Delete Network Entity, 3-44
command, 3-44
request message, 3-44
response message, 3-45
response return codes, 3-45
Delete Subscription, 3-28
command, 3-28
request message
delete DN block, 3-29
delete IMSI, 3-28
delete single DN, 3-28
response message, 3-29
response return codes, 3-29
Delete the IMSI from all IMEIs, 3-63
Description of PDBI, 2-1
Disconnect, 3-10
command, 3-10
request message, 3-10
response message, 3-11
response return codes, 3-11
disconnect command, 3-10
dlt_entity command, 3-44
dlt_sub command, 3-28
DN, 1-10
dn, 3-17, 3-18, 3-24, 3-25, 3-28, 3-31
DN Blocks, 2-5, 2-21
DN range, 2-5
documentation
admonishments, 1-8
packaging, 1-8
part numbers, 1-7
updates, 1-7
documentation set, 1-2
DSM, 1-10
DSM List, 4-16
Dump Connections, 3-52
command, 3-52
request message, 3-53
response message, 3-53
response return codes, 3-53
dump_conn command, 3-52

E

Eagle documentation set, 1-2
edn, 3-20, 3-26, 3-29, 3-31
eid, 3-46
eimsi, 3-34
End Transaction, 3-13
command, 3-13
request message, 3-13
response message, 3-14
response return codes, 3-14
end_txn command, 3-13
endchar, 3-9

ent_entity command, 3-36
ent_sub command, 3-15
EPAP, 1-10
  description, 2-1
  status reporting and alarms, 2-9
ETSI, 1-10
ETSI NP standards, 2-4
Example EPAP/PDBA Network, 2-2

F

files, import/export and PDBA debug log formats, 2-15
force, 3-16, 3-17, 3-19, 3-21, 3-50
format
  common response message, 3-3
  optional parameters, 3-3
  PDBA debug log, 2-15
  PDBA import and export files, 2-15
  response message data element, 3-3
FSM, 1-10

G

gc, 3-37, 3-41
GDB, 1-10
G-Flex, 1-10
GMSC, 1-10
GPDB, 1-10
GPL, 1-10
G-Port, 1-10
GSM, 1-10
GTA, 1-11
GTT, 1-11

H

HLR, 1-11

I

ID, 1-11
id, 3-36, 3-40, 3-44, 3-46
idletimeout, 3-9
IMEI Blocks, 2-23
IMSI, 1-11, 2-3
imsi, 3-16, 3-17, 3-23, 3-25, 3-28, 3-33
IMSIs, 2-20
IN, 1-11
INAP, 1-11
INP, 1-11, 2-4
instance table, 2-5
integer identification (iid), 3-3
Introduction to Data Model, 2-4
Introduction to Platform Services, 2-3
IP, 1-11
IS-41 to GSM Migration feature, 2-4
IS-ANR, 1-11
ISDN, 1-11
IS-NR, 1-11
ITU, 1-11

K

KB, 1-11
KSR, 1-11

L

LAN, 1-11
LIM, 1-11
limit to PDBI connections, 2-10, 3-8
LNP, 1-11
log file format, PDBA debug, 2-15
LSMS, 1-11

M

MA, 1-11
manual, related publications, 1-2
MAP, 1-11
MB, 1-11
MDM, 1-11
MEI Block Data, 4-13
message relay, 2-4
messages
  Abort Transaction, 3-14
  Begin Transaction, 3-11
  Connect, 3-8
  Create Network Entity, 3-36
  Create Subscription, 3-15
  Delete Network Entity, 3-44
  Delete Subscription, 3-28
  Disconnect, 3-10
  Dump Connections, 3-52
  End Transaction, 3-13

PDBA Status Query, 3-51
Retrieve Network Entity, 3-46
Retrieve Subscription Data, 3-30
Switchover, 3-48
Update Network Entity, 3-40
Update Subscription, 3-22
messages, PDBI request/response
see also request message and response message
common response format, 3-3
common response messages, 3-5
data element format, 3-3
definitions, 3-2
descriptions, 3-8
integer identification (iid), 3-3
optional parameter format, 3-3
MIN, 1-11
MMI, 1-11
MNP, 1-11
MO_FSM, 1-11
MPS, 1-11
MSC, 1-11
MSISDN, 1-12, 2-3
MSU, 1-12
MT_FSM, 1-12
MTS, 1-12
MTSU, 1-12
MTT, 1-12
Multiple Session Connectivity, 2-13

N

NAK, 1-12
NE, 1-12
NEBS, 1-12
neonly, 3-31
network connections, 2-8
Network Entities, 2-17
newline, 3-9
nnai, 3-38, 3-42
nnp, 3-38, 3-42
none, 3-8
normal transaction mode, 3-9
NP HLR, 2-3
ntt, 3-38, 3-42
null, 3-9
num, 3-32, 3-34, 3-47

O

OAM, 1-12
OAP, 1-12
OOS-MT-DSBLD, 1-12
optional parameter format, 3-3
OS, 1-12

P

packaging, documentation, 1-8
Parse Failed
reasons for, 3-5
response message, 3-5
return code, 3-5
part numbers, documentation, 1-7
PC, 1-12
pc, 3-37, 3-41
pctype, 3-37, 3-40
PDB, 1-12
PDBA, 1-12
connection with PDBI client, 3-8
disconnect client from, 3-10
dump connection information for debug, 3-52
import/export and debug log file formats, 2-15
replication, 2-7
switch Active/Standby status, 3-48
well-known port, 3-2, 3-8
PDBA replication, 3-48
PDBA Status Query, 3-51
request message, 3-51
response message, 3-52, 3-68, 3-71
response return codes, 3-52
status command, 3-51
PDBI, 1-12
client connection with PDBA, 3-8
connection, 3-8
connection security, 2-11
description, 2-1, 2-10
limit, 2-10, 3-8
overview, 3-2
request/response messages
see also request message and response message
descriptions, 3-8
sample sessions
abort transaction, 4-8

create Network Entities, 4-2
create simple subscription data, 4-3
queries with multiple responses, 4-7
simple queries, 4-5
update request in read transaction, 4-9
update subscription data, 4-4
write transaction to Standby PDBA, 4-10
System Architecture, 2-7
PID, 1-12
Platform Services, 2-3
port, PDBA well-known, 3-2, 3-8
portability type, 3-18
PP SMS, 1-12, 3-20
PPP, 1-12
prepaid 1, 2-22
prepaid 2, 2-22
prepaid type, 3-18
PT, 1-12
pt, 3-18, 3-20, 3-24, 3-26, 3-31
pt parameter, 3-18, 3-20

Q

query/response, 2-4

R

read transaction
abort, 3-14
begin, 3-11
end, 3-13
replicated database, 2-7
Replication, 3-51
replication, 2-7, 3-48
request ID (iid), 3-3
request message
Abort Transaction, 3-15
Begin Transaction, 3-11
Connect, 3-8
Create Network Entity, 3-36
Create Subscription
one IMSI and one to eight DNs, 3-16
one or more DNs on same NE with no IMSI, 3-17
porting a block of DNs, 3-20
single IMSI with no DN, 3-15
Delete Network Entity, 3-44
Delete Subscription
delete DN block, 3-29
delete IMSI, 3-28
delete single DN, 3-28
Disconnect, 3-10
Dump Connections, 3-53
End Transaction, 3-13
integer identification (iid), 3-3
optional parameter format, 3-3
PDBA Status Query, 3-51
Retrieve Network Entity
for all NEs, 3-47
for NE range, 3-46
for specific IMSI, 3-46
Retrieve Subscription Data
retrieve data for DN range, 3-31
retrieve data for IMSI range, 3-34
retrieve data for specific DN, 3-30
retrieve data for specific IMSI, 3-33
Switchover, 3-50
Update Network Entity, 3-40
Update Subscription
modify data of single DN, 3-23
modify information for DN block, 3-25
modify SP for specific IMSI, 3-23
move existing DN to existing IMSI, 3-25
response message
Abort Transaction, 3-15
Bad Arguments, 3-6
Begin Transaction, 3-13
common format, 3-3
common responses, 3-5
Connect, 3-10
Create Network Entity, 3-39
Create Subscription, 3-15, 3-22, 3-57, 3-60, 3-64
data element format, 3-3
Delete Network Entity, 3-45
Delete Subscription, 3-29
Disconnect, 3-11
Dump Connections, 3-53
End Transaction, 3-14
integer identification (iid), 3-3

Parse Failed, 3-5
PDBA Status Query, 3-52, 3-68, 3-71
Retrieve Network Entity, 3-47
Retrieve Subscription Data, 3-34, 3-66
Switchover, 3-50
Update Network Entity, 3-44
Update Subscription, 3-27
Retrieve Network Entity, 3-46
command, 3-46
request message
for all NEs, 3-47
for NE range, 3-46
for specific IMSI, 3-46
response message, 3-47
response return codes, 3-47
Retrieve Subscription Data, 3-30
command, 3-30
PDBI sample session with multiple responses, 4-7
PDBI sample session, simple queries, 4-5
request message
retrieve data for DN range, 3-31
retrieve data for IMSI range, 3-34
retrieve data for specific DNI, 3-30
retrieve data for specific IMSI, 3-33
response message, 3-34, 3-66
response return codes, 3-36, 3-67, 3-70, 3-72
return codes
Abort Transaction, 3-15
Bad Arguments, 3-6
Begin Transaction, 3-13
Connect, 3-10
Create Network Entity, 3-39
Create Subscription, 3-22, 3-57, 3-60, 3-64
Delete Network Entity, 3-45
Delete Subscription, 3-29
Disconnect, 3-11
Dump Connections, 3-53
End Transaction, 3-14
Parse Failed, 3-5
PDBA Status Query, 3-52
Retrieve Network Entity, 3-47
Retrieve Subscription Data, 3-36, 3-67, 3-70, 3-72
Switchover response message, 3-50
Update Network Entity, 3-44
Update Subscription, 3-27
RFC, 1-12
RI, 1-12
ri, 3-37, 3-41
RMTP, 1-12
RN, 1-12
rn, 3-19, 3-21, 3-24, 3-26, 3-32
rspsize, 3-8
RTDB, 1-12, 2-1
rtrv_entity command, 3-46
rtrv_sub command, 3-30

S

sample PDBI sessions
see sessions, PDBI sample
SCCP, 1-12
SEAC, 1-12
security, PDBI connection, 2-11
sessions, PDBI sample
abort transaction, 4-8
create Network Entities, 4-2
create simple subscription data, 4-3
queries with multiple responses, 4-7
simple queries, 4-5
update request in read transaction, 4-9
update subscription data, 4-4
write transaction to Standby PDBA, 4-10
side, 3-50
Simple Subscription Data Creation with Single Txnmode, 4-11
Single DNs, 2-20
Single IMEI Data, 4-12
Single IMEIs, 2-22
single instance table, 2-5
single transaction mode, 3-9
SMS, 1-12
SMSC, 1-13
SNCC, 1-13
socket connection, 3-8
SP, 1-13, 2-3
sp, 3-16, 3-17, 3-19, 3-21, 3-23, 3-24, 3-26, 3-32, 3-34
SRF, 1-13
srfimsi, 3-39, 3-42
SRI, 1-13

SSN, 1-13
ssn, 3-38, 3-41
standby database, 2-7
status command, 3-51
status reporting and alarms, EPAP, 2-9
STP, 1-13
String-Based Messages, 2-10
switchactn, 3-8
Switchover, 3-48
  command, 3-48
  request message, 3-50
  response message, 3-50
  response return codes, 3-50
switchover command, 3-48
Synchronous DSM Report, 4-15
System Architecture, 2-6
system architecture, PDBI/PDBA/EPAP, 2-6

T

TCP, 1-13
TDM, 1-13
terminate connection, 3-9
timeout, 2-14, 3-12, 3-16, 3-17, 3-19, 3-21, 3-23, 3-24, 3-25, 3-26, 3-28, 3-29, 3-39, 3-43, 3-45, 3-49, 3-50, 3-54, 3-55, 3-58, 3-59, 3-61, 3-62
TKLC, 1-13
transaction
  abort read, 3-14
  abort write, 3-14
  begin read, 3-11
  begin write, 3-11
  end read, 3-13
  end write, 3-13
  write, not active for PDBA switchover, 3-48
transaction boundaries, 3-9
transaction modes, 3-9
TSM, 1-13
txnmode, 3-9, 3-16
type, 3-12, 3-31, 3-36, 3-40, 3-44, 3-46, 3-53

U

UAM, 1-13
UDP, 1-13
UIM, 1-13
upd_entity command, 3-40
upd_sub command, 3-22
Update a block entry of IMEIs, 3-59
Update a single entry IMEI, 3-58
Update IMEI Data, 3-58, 3-61
Update Network Entity, 3-40
  command, 3-40
  request message, 3-40
  response message, 3-44
  response return codes, 3-44
Update Subscription, 3-22
  command, 3-22
  PDBI sample session, 4-4
  request message
    modify data of single DN, 3-23
    modify information for DN block, 3-25
    modify SP for specific IMSI, 3-23
    move existing DN to existing IMSI, 3-25
  response message, 3-27
  response return codes, 3-27
updates, documentation, 1-7

V

version, 3-8
VMSC, 1-13
VSCCP, 1-13

W

well-known port, PDBA, 3-2, 3-8
write transaction
  abort, 3-14
  begin, 3-11
  end, 3-13
  not active for PDBA switchover, 3-48