Oracle® Communications

EAGLE Application Processor Provisioning Database Interface User's Guide

Release 16.4
F29904-02
August 2021

ORACLE

Oracle Communications EAGLE Application Processor Provisioning Database Interface User's Guide, Release 16.4

F29904-02

Copyright © 2000, 2021, Oracle and/or its affiliates.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or "commercial computer software documentation" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government’s use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

1 Introduction

Overview 1-1

2 Functional Description

General Description 2-1
System Architecture 2-7
Provisioning Database Interface Description 2-10
- Socket Based Connection 2-10
- String-Based Messages 2-10
- Security 2-11
  - Remote Port Forwarding 2-11
  - How to Configure the CPA for Connecting to the PDBA 2-13
- Transaction-Oriented API 2-13
- Batch-Oriented/Bulk Load 2-14
- Command Atomicity 2-15
- Provisioning Ranges of Subscriber Numbers 2-15
- Transparency of Redundant Systems 2-17
- Logs 2-17
- Crash Recovery 2-17
- Request IDs 2-17
- Multiple Session Connectivity 2-17
- Request Queue Management 2-18
- Interface Configuration and Installation 2-18

File Formats 2-18
- Debug Log 2-18
- Import/Export Files 2-19
  - Import File 2-19
  - Export File 2-21

3 PDBI Request/Response Messages

Overview 3-1

Messages 3-6
Connect 3-6
Disconnect 3-9
Begin Transaction 3-10
End Transaction 3-12
Abort Transaction 3-13
Create Subscription 3-14
Subscription Containing a Single IMSI with No DNs 3-14
Subscription Containing an IMSI and One to Eight DNs 3-15
One or More DNs on the Same NE with no IMSI 3-17
Subscription Porting a Block of DNs 3-21
Create Subscription Responses 3-25
Update Subscription 3-28
Modify the SP for a Specific IMSI 3-28
Modify the Subscription Data of a Single DN 3-29
Move an existing DN to an Existing IMSI 3-32
Modify the Subscription Information for a DN Block 3-33
Update Subscription Responses 3-37
Delete Subscription 3-39
Delete an IMSI 3-39
Delete a Single DN 3-40
Delete a DN block 3-40
Delete Subscription Responses 3-41
Retrieve Subscription Data 3-42
Retrieve Subscription Information About a Specific DN 3-42
Retrieve Subscription Information for a Range of DNs 3-46
Retrieve Subscription Information About a Specific IMSI 3-50
Retrieve Subscription Information for a Range of IMSIs 3-50
Retrieve Subscription Data Responses 3-51
Create Network Entity 3-53
Update Network Entity 3-57
Delete Network Entity 3-61
Retrieve Network Entity 3-62
Retrieve the Information for a Specific NE 3-62
Retrieve the Information for a Range of NEs 3-63
Retrieve the Information for All NEs 3-64
Retrieve Network Entity Responses 3-64
Switchover 3-65
PDBA Status Query 3-68
Dump Connections 3-69
Create IMEI Data 3-70

Create a Single Entry IMEI 3-70
Create a Block Entry of IMEIs 3-72
Create a New IMSI and Associate it with an Existing IMEI 3-73
Create IMEI Data Responses 3-74
Update IMEI Data 3-76
Update a Single Entry IMEI 3-76
Update a Block Entry of IMEIs 3-78
Update IMEI Data Responses 3-79
Delete IMEI Data 3-80
Delete a Single Entry IMEI 3-80
Delete a Block of IMEIs 3-81
Delete IMSI(s) from the Associated IMEI 3-81
Delete the IMSI from all IMEIs 3-82
Delete IMEI Data Responses 3-83
Retrieve IMEI Data 3-83
Retrieve All the Data Associated with a Single IMEI Entry 3-84
Retrieve IMEI Data: Retrieve a Range of IMEIs 3-84
Retrieve IMEI Data Responses 3-86
Request Service Module Card Report 3-87
Retrieve Service Module Card Report 3-89
Retrieve a List of the Service Module Cards 3-91

4 PDBI Sample Sessions

Introduction 4-1
Network Entity Creation 4-1
Simple Subscription Data Creation 4-2
Update Subscription Data 4-2
Simple Queries 4-6
Multiple Response Query 4-7
Abort Transaction 4-8
Update Request In Read Transaction 4-9
Write Transaction In Standby Connection 4-9
Simple Subscription Data Creation with Single Txnmode 4-10
Single IMEI Data 4-11
IMEI Block Data 4-11
Asynchronous Service Module Card Report 4-12
Synchronous Service Module Card Report 4-13
Service Module Card List 4-13

A PDBI Message Error Codes

PDBI Message Error Codes A-1

B TIF Number Substitution Relationships

TIF Number Substitution Relationships B-1

C TIF Linkset Based Blocklist Feature

D DN Block Self Healing

DN Block Self Healing D-1

My Oracle Support

My Oracle Support (https://support.oracle.com) is your initial point of contact for all product support and training needs. A representative at Customer Access Support can assist you with My Oracle Support registration.

Call the Customer Access Support main number at 1-800-223-1711 (toll-free in the US), or call the Oracle Support hotline for your local country from the list at http://www.oracle.com/us/support/contact/index.html. When calling, make the selections in the sequence shown below on the Support telephone menu:

- For Technical issues such as creating a new Service Request (SR), select **1**.
- For Non-technical issues such as registration or assistance with My Oracle Support, select **2**.
- For Hardware, Networking and Solaris Operating System Support, select **3**.

You are connected to a live agent who can assist you with My Oracle Support registration and opening a support ticket.

My Oracle Support is available 24 hours a day, 7 days a week, 365 days a year.

1 Introduction

This chapter contains general information about the PDBI documentation, the organization of this manual, and how to get technical assistance.

Overview

The *Provisioning Database Interface User's Guide* defines the interface that is used to populate the Provisioning Database (PDB) for the G-Flex, G-Port, EIR, INP, A-Port, AINPQ, **V-Flex**, and IS41GSM Migration features of the **EAGLE**.

2
Functional Description

This chapter provides an overview of PDBI, EPAP, PDBA, and DSM functions.

General Description

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

EPAP

As shown in Figure 2-1, the provisioning system contains two mated EPAPs. Of the two mated EPAPs, only one is the Active PDBA, while the other acts as a Standby PDBA.

Each EPAP maintains two copies of the RTDB in the B-Tree format. When a Service Module card needs a copy of the RTDB, the Active RTDB downloads the B-Tree file to the Service Module card. Each Service Module card uses the B-Tree file to create its own copy of the RTDB database. The primary purpose of an EPAP is to download the RTDB to the Service Module cards.

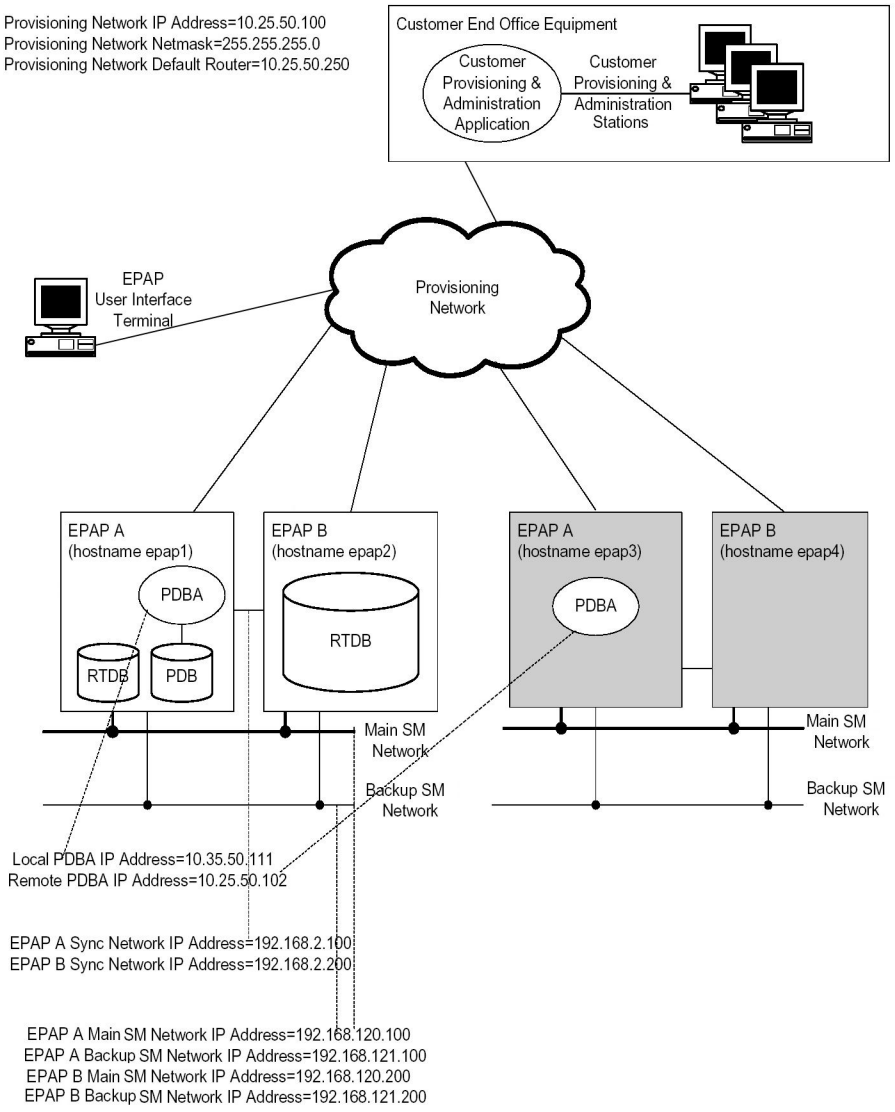
The Active PDBA interfaces with the customer database through the PDBI, which provides PDB updates. When the customer submits provisioning requests, the Active PDBA updates its PDB. After the updates are applied to the PDB of the Active PDBA, the updates are sent to the Standby PDBA.

Standalone PDB EPAP - This configuration supports only the Provisioning Database (PDB) and its associated processes. The Real Time Database (RTDB), EAGLE interfaces, Service Module cards, Mate Servers, and associated processes are not supported by this configuration. Standalone PDB EPAP is identified as an EPAP. Content in this chapter and subsequent chapters which is RTDB-related or requires a Mate server or EAGLE interface does not apply to this configuration. Refer to *Administration Guide* for specific information about the Standalone PDB EPAP configuration.

Service Module Card

As Figure 2-1 shows, the provisioning system uses up to 32 Service Module cards. Multiple Service Module cards 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 Service Module card contains an identical database. The RTDB on the Service Module cards must be identical to the RTDB maintained by the EPAPs. However, there are several reasons why the various databases might not be identical. When a Service Module 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 an update from its provisioning source. These updates are applied immediately to the Active EPAP PDB as they are received, but there is a delay before sending the updates to each EPAP RTDB and then subsequently to the Service Module cards.

Two possible scenarios lead to the condition when a Service Module card might not have enough memory to hold the entire database:

- The database is downloaded successfully to Service Module card, but subsequent updates eventually increase the size of the database beyond the Service Module

card memory capacity. In this situation, it is desirable to continue message processing, even though the database might not be as up-to-date as it could be.

- When a Service Module card is booted, 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 Service Module card is responsible for recognizing and reporting out-of-memory conditions. Under this condition, a Service Module card cannot process provisioning traffic.

Introduction to Platform Services

The 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, EIR, A-Port, IS41 GSM Migration, **V-Flex**, and AINPQ databases and for configuring the G-Flex, G-Port, INP, EIR, A-Port, IS41 GSM Migration, and AINPQ 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 ANSI-41 Mobile Number Portability (A-Port) feature supports mobile number portability in ANSI-based networks. The ANSI-41 Mobile Number Portability feature uses the EAGLE Application Processor (EPAP) provisioning database to retrieve the subscriber portability status and provision directory numbers for exported and imported IS41 subscribers. The A-Port feature supports both IS41 LOCREQ and SMSREQ messages for number portability handling. The A-Port feature uses the MNP SCCP Service Selector to process GTT-routed LOCREQ and SMSREQ SCCP messages.

The IS41 GSM Migration feature refers to the movement of the subscribers of an ANSI IS-41MAP protocol based network to a GSM MAP protocol based network while retaining their mobile telephone numbers.

After migration, subscribers are able to:

- Use **GPRS**-based data services that are provided only by GSM networks.
- Enhance their roaming capability to a larger number of countries because GSM networks are more widely deployed worldwide than IS41 networks.

The IS41 GSM Migration feature uses the G-Port MSISDN portability type (PT) field to identify subscribers that have migrated from IGM 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 IS41 number range. Since these subscribers are either migrated (PT=5) or not migrated (PT=0), the new PT values do not logically overlap the existing values. PT values are mutually exclusive of each other.

The ANSI-41 Number Portability Query (AINPQ) feature provides number portability in networks that support a mix of ITU and ANSI protocols by allowing ANSI-41 NPREQ queries on the EAGLE database. INP uses the INAP TCAP protocol and AINPQ uses the ANSI-41 TCAP protocol for Query functions.

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 GSM Mobile Number 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 **ETS I** 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 Block, Gray, and Allow. When an IMEI is placed on the block list, the carrier is able to prevent the handset from accessing their network. An Allow listed IMEI is allowed access to the network, while a Gray may require additional screening but is typically allowed access to the network.

EAGLE generates logs for Block list, Allow list, and Grey list IMEIs for all other possible scenarios. Following are a few examples:

- IMEI was block listed but blocklist was overridden because IMSI was present in the database
- IMEI was block listed and IMSI was not present in the database, therefore, blocklist continued.
- IMEI was not Block/Gray/Allow Listed, resulting in Allow List. The EIR log file is present at the location `/var/TKLC/epap/free` folder with the name `eirlog_<hostname>` where MPS server where the log file is stored.
  Along with the response code of EIR, each entry in the log file has the following information.

  ```
  < Time/Date stamp>, < Source Identifier>, <Source Sequence
  Number>, <IMSI>, <IMEI>, <response Code>, <Point Code Type>,
  <point Code or Hostname>
  ```

  For example,
  `20030715163600,192.168.61.1,1234,9195551212,12345678901234,0,ansi,3-3-1`

EIR logs are sent through UDP socket to the EPAP. The EPAP processes and stores these logs in the EIR List Log files.

The Prepaid Short Message Service Intercept (**PPSMS**) feature uses the G-Port DN portability type (PT) field to identify prepaid subscribers. These subscribers can be categorized as subscribers that are ported and subscribers that are not ported. The originated short messages (as part of SMS) of these subscribers need to be intercepted and forwarded to a corresponding intelligent network platform for verification. However, the new PT values of the subscribers that are ported in or not ported do not logically overlap with the existing values. Therefore, the PT values for these subscribers cannot be set when the DN associated with an SP is removed. In

order to minimize changes to the interface, the PT field is not added to the commands where an IMSI is provided as input. PPSMS is a part of G-Port that is activated separately.

The V-Flex feature is used to route calls to a specific VMS based on subscription (voice, multimedia) data provisioned via the EAGLE MMI port and EPAP PDBI . The V-Flex feature utilizes VMS and GRN Network Entity types. In addition, the Multiple Network Entities per Subscriber feature introduces the ability to associate DN Blocks and individual DNs with up to 2 NEs. SP and RN remain mutually exclusive, however any combination of 2 NEs per DN is allowed so long as there is only 1 of each Network Entity type.

The **ASD** (Additional Subscriber Data) feature enables generic data to be associated with DN and DN Block subscriber records.

The EPAP Provisioning blocklist feature helps prevent provisioning of protected E.164 address strings in the EPAP G-Flex database. Provisioning a protected E.164 address string as a DN, DN Block, or IMSI may result in unintended and incorrect routing of messages by the EAGLE Service Module card. The EPAP Provisioning blocklist feature allows the user to define a list of address strings that cannot be provisioned as DN, DN Block or IMSI address strings. The E.164 addresses of all HLRs must be provisioned in the provisioning blocklist.

The TIF Number Substitution feature is used to provision a new DN association for DNs and DN Blocks. All DN and DN Block records have a subscriber type to identify them as either public or private. Public DN and DN Block records may substitute to private DN. Likewise, private DN and DN Blocks may substitute to a public DN. Records are public by default; this default applies to pre-existing records and new records for which subscriber type is not explicitly defined.

The TIF Linkset Based blocklist functionality enables a misused user to make legitimate calls in case it is blocked on a particular linkset. The feature stores the blocklisted information for each number including the blocklisted SetID. Therefore, all the messages arriving on EAGLE are screened with the following combination:

- The blocklisted SetID referred in incoming linkset
- The blocklisted SetID configured in RTDB

The IDP A-Party blocklist feature provides subscriber blocklisting capability on the Calling Party (A-Party or CgPN) number in the IDP CAMEL message. The blocklisting function is achieved using either a query-based mode, or a relay-based mode in conjunction with IDP Relay feature processing. The blocklist data is used by the EAGLE to support IDP queries. If the calling party is associated with a blocklisted flag and a GRN has been provisioned against the associated DN or DN Block, then a connect message is sent back to the switch along with the GRN number. The GRN is then used to re-route the call to a predetermined destination. Pre-existing DN and DN Block records have blocklisting disabled by default.

EPAP-related features share the same Real Time Database (RTDB) database when operated together on a single node. EIR and INP/AINPQ are mutually exclusive on a node.

Introduction to the Data Model on the Platform

The PDBA 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:

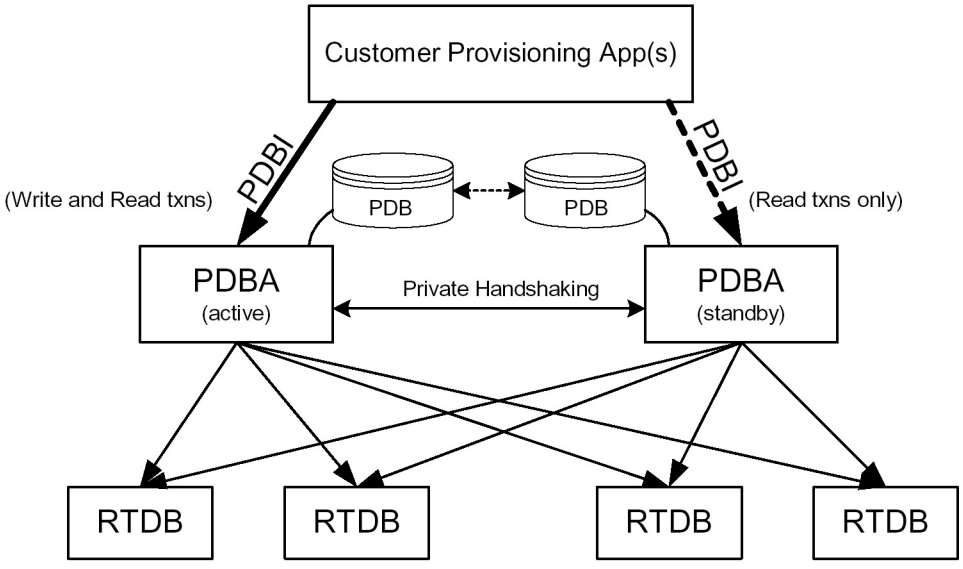
- DN range 10080-10600 encompasses the Single DN 10080 and DN Blocks 10000-10100 and 10500-10800
- DN range 10850-11500 encompasses the Single DN 10900 and DN Block 11000-12000
- DN range 10000-10040 encompasses no Single DNs and DN Block 10000-10100
- DN range 10400-13000 encompasses the Single DN 10900 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 PDBAs, one in **EPAP** A on each EAGLE. 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.

You can configure which PDBA forwards updates to an RTDB. 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 shows a block diagram of the MPS/EPAP platform. It also shows a mated pair of EAGLE. The EAGLE are the large blocks at the bottom. The MPSs, which are attached to the EAGLE, are above the EAGLE and contain EPAP A and EPAP B.

An MPS system consists of two MPS servers and associated hardware Each EAGLE 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 (the MPS system for one EAGLE), 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**, **INP**, **EIR**, **A-Port**, **AINPQ**, and **IGM** 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, INP, EIR, A-Port, AINPQ, and IS41 GSM Migration provisioning, the MPS upper and lower servers are called simply EPAP A and EPAP B or MPS A and MPS B.

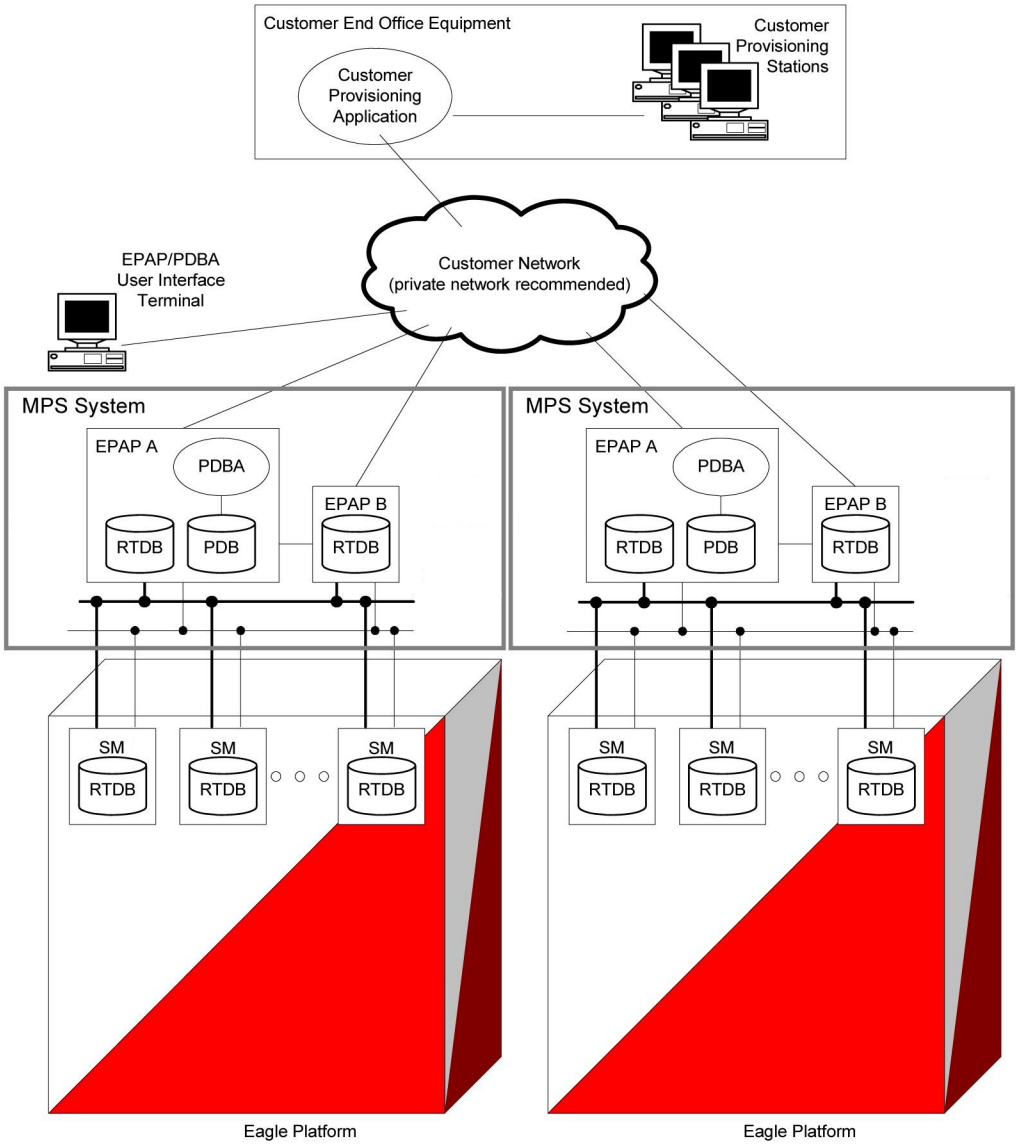
EPAP A and EPAP B are slightly different in their configuration. EPAP A runs the PDBA software and thus holds a copy of the PDB. This is the EPAP that is accessed using the PDBI. EPAP A also holds a copy of the RTDB for downloading to the Service Module cards. EPAP B contains a redundant copy of the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

The EPAPs are connected to the Service Module cards via a 10/100/1000 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, and the RTDB are described in detail in *EPAP Administration Guide*.

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 Service Module through EPAP Maintenance Blocks and Service Module Status Requests.

The status reporting, message format, and various alarm messages are explained in detail in *EPAP Administration Guide*.

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. Oracle 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 connections, contact Tekelec for information. concurrent client

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 will 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 will simply receive 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.
- It is easier for the PDBA to support any combination of the G-Flex, G-Port, INP, **EIR**, **A-Port**, **AINPQ** and **IGM** Migration 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 Guide*).

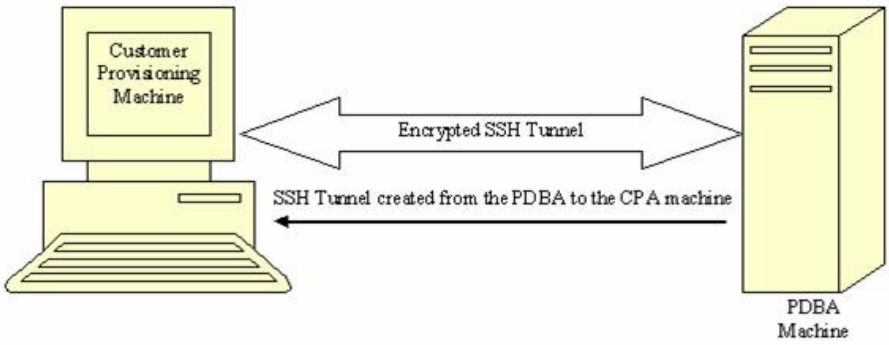
Remote Port Forwarding

Remote Port Forwarding refers to the SSH tunneling approach where the SSH tunnel is created from the client side of the tunnel towards the server side. The **CPA** machine is the server and the PDBA machine is the client.

> **Note:**
>
> SSH tunneling/Remote port forwarding can be simultaneously used with Normal (Transaction oriented) Provisioning mode.

Figure 2-4 shows an SSH tunnel on a connection between the Customer Provisioning machine and the PDBA machine.

Figure 2-4 SSH Tunnel Between the CPA and PDBA Machines



The PDBA machine user specifies a particular port number (configurable from GUI) to be opened on the CPA machine. Any data received on this port on the **CPA** machine is forwarded to the PDBA machine's IP address and the port number, 5873, through the secured SSH tunnel.

> **Note:**
>
> To implement Remote Port Forwarding to work, the CPA machine must have the OpenSSH suite (version 3.6.1 or later) installed and the SSH daemon must be running.

Request/Response Cycle in SSH Tunnel

When an SSH tunnel is in use, a complete request and response cycle takes place as follows:

1. The CPA sends a connect request to its local port number used for creating the tunnel.
2. The SSH encrypts the request message and sends it to the PDBA machine's SSH client port.
3. On the PDBA machine, the SSH client decrypts the message and forwards it to the PDBA port.
4. The PDBA gets the request message in unencrypted form and sends an unencrypted response to the SSH client.
5. The SSH client encrypts the response message and sends it to the SSH port on the **CPA** machine.

6. On the CPA machine, the SSH daemon decrypts the message and forwards it to the **CPA**. The **CPA** receives the message unencrypted.

How to Configure the CPA for Connecting to the PDBA

> **Note:**
>
> SSH tunneling does not work with the PDB proxy feature as SSH tunneling works with server IP and not with VIP.

This section describes the parameters that must be defined on a **CPA** (Customer Provisioning Application) machine to allow it to connect to the PDBA. The parameters used to configure a **CPA** machine for connecting to the PDBA depend on whether SSH tunneling is to be used.

How to Configure the CPA When SSH Tunneling Is Not Used

If SSH tunneling is not to be used, configure the CPA with the following parameters:

- IP address of the PDBA
- PDBA port number 5873

How to Configure the CPA When SSH Tunneling Is Used

If SSH tunneling is to be used, the OpenSSH suite (version 3.6.1 or later) must be installed on the **CPA**, the sshd daemon must be running, and the following parameters configured on the **CPA** machine:

- IP address: either 127.0.0.1 or localhost
- Port number: Use the same port number as is configured through the EPAP GUI (for more information, refer to *Administration Guide*.)

In addition, provide the following information to the EPAP administrator:

- Port number on the **CPA** to be used for tunneling (this port number should be the same as the port number specified on the **CPA** machine)
- Username and password of the **CPA** machine

> **Note:**
>
> The password is not stored by the EPAP software. It is used only one time for setting up the SSH tunnel.

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 (seer Command Atomicity).

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 and the `txnmode` parameter.

Batch-Oriented/Bulk Load

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

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 cannot be interrupted. 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.

Note that the EPAP **GUI** provides menus to provision IMSI Ranges, however these are not equivalent to MSISDN Blocks or IMEI Blocks. These IMSI Ranges are not provisioned via PDBI and are not downloaded to the EPAP RTDBs or Service Module RTDBs.

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. At the same time, the data is also passed, asynchronously, to the standby PDB and subsequently to the mated RTDBs on the mated EAGLE. 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 *Administration Guide*.

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 *Administration Guide*.

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. At the same time, the data is also passed, asynchronously, to the standby PDB and subsequently to the mated RTDBs on the mated EAGLE. 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 *Administration Guide*.

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 *Administration Guide*.

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 *Administration Guide*.

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 *Administration Guide*.

Import/Export Files

The Import and Export files use the **PDBI** `create` command format (see Create Subscription and Create Network Entity). 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 *Administration Guide*.) The format of the commands in the file is exactly the same as the PDBI commands specified in this document.

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

> **Caution:**
>
> Do not use tabs instead of spaces in the commands. Using tabs causes the command and replication to the standby PDB to fail.

The valid import file commands are:

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

> **Note:**
>
> Do not include `rtrv_sub`, `rtrv-entity`, or `rtrv_eir` commands in an import file. The inclusion of `rtrv` commands causes an import to take a very long time to complete. During an import, a write transaction lock is in place for the entire import for a manual import, and intermittently in place for an automatic import. While the write transaction lock is in place during an import, no other updates to the database can be made.

Data can be imported manually or automatically, as described in the following sections:

- Manual Import
- Automatic Import

The syntax of the imported file data is described in Import File Syntax .

Manual Import

The manual import mode is used to import data typically on a one-time basis or as needed and is configured by the Import File to PDB Screen. The selected file is processed immediately. A manual import locks the PDB write transaction; other users will not be able to obtain the write transaction until the import operation is complete.

Automatic Import

As long as the PDB is active, the automatic import searches the*/var/TKLC/epap/free/pdbi_import* directory for new files on a remote system for import every 5 minutes. If a file exists in the directory and it is not being modified or in the process of being transferred when it is polled, the import will run automatically at that time. If the file is being modified or is in the process of being transferred, the automatic import tries again after five minutes. Delaying when a file is being modified or in the process of being transferred prevents the import of incomplete files.

The automatic import option can import up to 16 files at a time. This is limited by the available number of PDBI connections. If more than 16 files exist in the directory, as soon as one file completes, another file is started until all files have completed. The results of the import are automatically exported to the remote system specified by the Configure File Transfer Screen (described in *Administration Guide*).

Once the import is complete, the data file is automatically removed and a results file is automatically transferred back to the remote system. An automatic import obtains the PDB write transaction and processes several of the import file commands. Then the write transaction is released, allowing other connections to provision data. An automatic import obtains the write transaction repeatedly until all the import file commands have been processed.

Automatic import is also called "Batch-oriented/bulk load" (see Batch-Oriented/Bulk Load).

Import File Syntax

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 *Administration Guide*.)

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), 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 *Administration Guide*.) The data can be formatted in two ways, either as **PDBI** commands or as raw delimited **ASCII**.

Three modes of export are supported in the **EPAP** software. Depending on the mode of export selected, the **EPAP** may be blocked from performing database updates or allowed to provision new data and data retrieve operations on the **EPAP** provisioning database **PDB**) during the export.

EPAP provides the following modes of operation:

- Blocking mode

  The Blocking mode blocks write requests to the **EPAP** database during a database export. Writes will not be allowed until the export completes.

- Snapshot mode

  > **Note:**
  >
  > This mode causes the server to run increasingly slower as updates are received on the other connections.

  The Snapshot mode allows write operations on the database during a database export. This mode provides the exported database as a complete snapshot of the database at the time the export started. This implies that changes to the database after the export started are not reflected in the exported database. This allows for a logically complete export file.

- Real Time mode

  The Real Time mode allows write operations during a database export, and provides the export file in real-time fashion rather than as a snapshot. Changes to the DB after the export has started may or may not be reflected in the export file, depending whether the changes are to an area of the DB that has already been exported. This mode also provides a file that could be imported back into the database later, but is less than ideal, since it is not a complete snapshot of a given time. As an additional point of data, the level of the database when the export finished is placed at the end of the export file.

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 and DN updates for individualized data)
3. Single **DNs** (that are not associated with any IMSI)
4. DN Blocks
5. **IMEIs** (with associated **IMSIs**)
6. IMEI blocks
7. Update DNs (with information like TIF Number Substitution)

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 six separate sections.

Each of the following sections corresponds to a data type, as follows:

- Network Entities
- IMSIs
- Single DNs
- DN Blocks
- Single IMEIs
- IMEI Blocks

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.

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
V - Voicemail Server
G - Generic 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 [(s-)z-aaa-i].

s - Optional spare point code indicator
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 [(s-)nnnnn].

s - Optional spare point code indicator
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
4 - Delete the country code
5 - Delete the country code and prepend with entity id
6 - Send a digit action of 6 to the EAGLE

7 - Send a digit action of 7 to the 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:

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

Example Network Entity Entry:

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

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.

Example IMSI Entry:

```
1234567890,101010
```

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

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>,<VMS>,<GRN> ,<ASD>,<ST>,<NSDN>,<CGBL>,<CDBL
>,<LSBLSET>
```

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, IS41 GSM Migration, A-Port, and PPSMS. For G-Port and A-Port, it controls number Portability Status encoding in SRI acks. For IS41 GSM Migration, it identifies whether a subscriber has or has not migrated from IS41 to GSM, (maintaining a single GSM handset). For PPSMS, it identifies a DN as one of thirty-two types needing PPSMS intercept.
Values:

none– no status (default = none)
0 – not known to be ported, migrated to **IS41** or non-migrated **IS41** sub (used for IS41 GSM Migration)
1 – own number ported out (used for G-Port and A-Port)
2 – foreign number ported to foreign network (used for G-Port and A-Port)
3 – prepaid 1 (used by PPSMS)
4 – prepaid 2 (used by PPSMS)
5 – migrated to GSM (used for IS41 GSM Migration)
6 – prepaid 3 (used by PPSMS)
7 – prepaid 4 (used by PPSMS)
8 – prepaid 5 (used by PPSMS)
9 – prepaid 6 (used by PPSMS)
10 – prepaid 7 (used by PPSMS)
11 – prepaid 8 (used by PPSMS)
12 – prepaid 9 (used by PPSMS)
13 – prepaid 10 (used by PPSMS)
14 – prepaid 11 (used by PPSMS)
15 – prepaid 12 (used by PPSMS)
16 – prepaid 13 (used by PPSMS)
17 – prepaid 14 (used by PPSMS)
18 – prepaid 15 (used by PPSMS)
19 – prepaid 16 (used by PPSMS)
20 – prepaid 17 (used by PPSMS)
21 – prepaid 18 (used by PPSMS)
22 – prepaid 19 (used by PPSMS)
23 – prepaid 20 (used by PPSMS)
24 – prepaid 21 (used by PPSMS)
25 – prepaid 22 (used by PPSMS)
26 – prepaid 23 (used by PPSMS)
27 – prepaid 24 (used by PPSMS)
28 – prepaid 25 (used by PPSMS)
29 – prepaid 26 (used by PPSMS)
30 – prepaid 27 (used by PPSMS)
31 – prepaid 28 (used by PPSMS)
32 – prepaid 29 (used by PPSMS)
33 – prepaid 30 (used by PPSMS)
34 – prepaid 31 (used by PPSMS)
35 – prepaid 32 (used by PPSMS)
36 – not identified to be ported

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.

VMS
(Optional) Specifies which Voicemail Server the DNs are on. Corresponds to the E.164 address of the voicemail server. The VMS must correspond to an existing VMS entity.
Values:

1 to 15 hexadecimal digits expressed using **ASCII** characters.

GRN
(Optional) Specifies which Generic Routing Number the DNs are on. Corresponds to the E.164 address used when the EAGLE “NE Query Only Option” has been turned on. The GRN must correspond to an existing GRN entity.
Values:

1 to 15 hexadecimal digits expressed using **ASCII** characters.

ASD
(Optional) Specifies the Additional Subscriber Data that is associated with this DN. Leading zeros are significant.

Values
1 to 10 hexadecimal digits expressed using ASCII characters. Leading zeros are significant.

ST
(Optional) The subscriber type for created DNs.

Values
A decimal number in the range
0 - public
1 - private

NSDN
A TIF Number Substitution DN (specified in international format).
Values:

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

CGBL
(Optional) IDP calling party blocklist.

Values
yes - IDP calling party blocklist is enabled

> **Note:**
>
> The cgbl parameter will only be listed in the export file if its value is **yes**.

CDBL
(Optional) IDP called party blocklist.

Values
yes - IDP called party blocklist is enabled

> **Note:**
>
> The cdbl parameter will only be listed in the export file if its value is **yes**.

lsblset
(Optional) A TIF Linkset based blocklist parameter that is used to decide if the DN needs to be blocklisted or not.

Values
a number from **1** to **255**.

By default, no value is selected.

Example DN Entry:

```
12345,1234567890,,101010,,202020,,,,,
```

DN Blocks

The fourth 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>
<VMS>,<GRN>,<ASD>,<ST>,<NSDN>,<CGBL>,<CDBL>,<LSBLSET>
```

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, IS41 GSM Migration, A-Port, and PPSMS. For G-Port and A-Port, it controls number Portability Status encoding in SRI acks. For IS41 GSM Migration, it identifies whether a subscriber has or has not migrated from IS41 to GSM, (maintaining a single GSM handset). For PPSMS, it identifies a DN as one of thirty-two types needing PPSMS intercept.
Values:

none– no status (default = none)
0 – not known to be ported
migrated to **IS41** or non-migrated **IS41** sub (used for IS41GSM Migration)
1 – own number ported out (used for -Port and A-Port)
2 – foreign number ported to foreign network (used for G-Port and A-Port)
3 – prepaid 1 (used by PPSMS)
4 – prepaid 2 (used by PPSMS)
5 – migrated to GSM (used for IS41 GSM Migration)
6 – prepaid 3 (used by PPSMS)
7 – prepaid 4 (used by PPSMS)
8 – prepaid 5 (used by PPSMS)
9 – prepaid 6 (used by PPSMS)
10 – prepaid 7 (used by PPSMS)
11 – prepaid 8 (used by PPSMS)
12 – prepaid 9 (used by PPSMS)
13 – prepaid 10 (used by PPSMS)
14 – prepaid 11 (used by PPSMS)
15 – prepaid 12 (used by PPSMS)
16 – prepaid 13 (used by PPSMS)
17 – prepaid 14 (used by PPSMS)
18 – prepaid 15 (used by PPSMS)
19 – prepaid 16 (used by PPSMS)
20 – prepaid 17 (used by PPSMS)
21 – prepaid 18 (used by PPSMS)
22 – prepaid 19 (used by PPSMS)
23 – prepaid 20 (used by PPSMS)
24 – prepaid 21 (used by PPSMS)
25 – prepaid 22 (used by PPSMS)
26 – prepaid 23 (used by PPSMS)
27 – prepaid 24 (used by PPSMS)
28 – prepaid 25 (used by PPSMS)
29 – prepaid 26 (used by PPSMS)
30 – prepaid 27 (used by PPSMS)
31 – prepaid 28 (used by PPSMS)
32 – prepaid 29 (used by PPSMS)
33 – prepaid 30 (used by PPSMS)
34 – prepaid 31 (used by PPSMS)
35 – prepaid 32 (used by PPSMS)
36 – not identified to be ported

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 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.

VMS
(Optional) Specifies which Voicemail Server the DNs are on. Corresponds to the E.164 address of the voicemail server. The VMS must correspond to an existing VMS entity.
Values:

1 to 15 hexadecimal digits expressed using **ASCII** characters.

GRN
(Optional) Specifies which Generic Routing Number the DNs are on. Corresponds to the E.164 address used when the EAGLE “NE Query Only Option” has been turned on. The GRN must correspond to an existing GRN entity.
Values:

1 to 15 hexadecimal digits expressed using **ASCII** characters.

ASD
(Optional) Specifies the Additional Subscriber Data that is associated with this DN. Leading zeros are significant.

Values
1 to 10 hexadecimal digits expressed using ASCII characters. Leading zeros are significant.

ST
(Optional) The subscriber type for created DNs.

Values
A decimal number in the range
0 - public
1 - private

NSDN
A TIF Number Substitution DN (specified in international format).
Values:

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

CGBL
(Optional) IDP calling party blocklist.

Values
yes - IDP calling party blocklist is enabled

> **Note:**
>
> The cgbl parameter wil only be listed in the export file if its value is **yes**.

CDBL
(Optional) IDP called party blocklist.

Values
yes - IDP called party blocklist is enabled

> **Note:**
>
> The cdbl parameter wil only be listed in the export file if its value is **yes**.

lsblset
(Optional) A TIF Linkset based blocklist parameter that is used to decide if the DN needs to be blocklisted or not.

Values
a number from **1** to **255**.

By default, no value is selected.

SPLIT
(Optional) DN Block splitting ability

Values
yes - DN Block splitting is enabled

no - DN Block splitting is disabled

> **Note:**
>
> The split parameter will be listed in the export file only if its value is **no**

Example DN Block Entry:

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

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>,<Allow>,<GRAY>,<block>,<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)

Allow
(Optional) Specifies a List Type of Allow
Values:

yes or **no**

GRAY
(Optional) Specifies a List Type of Gray
Values:

yes or **no**

Block
(Optional) Specifies a List Type of block
Values:

yes or **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**.

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>,<ALLOW>,<GRAY>,<block>`

Where:

BIMEI
Specifies the beginning of the MEI 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**.

ALLOW
(Optional) Specifies a List Type of Allow
Values:

yes or **no**

GRAY
(Optional) Specifies a List Type of Gray
Values:

yes or **no**

Block
(Optional) Specifies a List Type of block
Values:

yes or **no**

3
PDBI Request/Response Messages

This chapter describes available requests and the possible responses for PDBI request/response messages.

Overview

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

Provisioning

Provisioning clients connect to the **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**, **INP**, **EIR**, **A-Port**, and/or **IS41 GSM** Migration 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 *Commands User's Guide*.

Number Prefixes

The PDBA has the concept of default number prefixes. These are PDBA parameters that are configurable from *Administration Guide*. 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 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.

The Number Prefix must conform to syntax rules for DN, NSDN, and DN Block values. Prefix lengths must not exceed 10 digits.

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.
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 RTDB imposes a transaction size limit on the PDBA. In order to ensure that the PDBA and the EAGLE 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 database. The limit is 200 EAGLE 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 database commands, one for the IMSI and one for the DNs. The worst case number of EAGLE 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 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.

Service Module Card Report

The PDBA keeps track of the status of the **Service Module** 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 Service Module card Report. The Service Module card 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 Service Module card Report is the same. The intent of the Service Module card Report is to inform the receiver what percentage of Service Module cards are at a specific database level. This information can be used by the client to determine when enough Service Module cards have a specific update to consider it safe for traffic.

The report includes the database level being reported on, the percentage of Service Module cards that have that level, and the total number of known Service Module cards. Also included is a list of all Service Module 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, Oracle is able to configure and support up to 128 connections. If more than 16 concurrent client connections are required, contact My Oracle Support for more information.

Request

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

Parameters :

version
(Optional) 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 amount of time 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.
[00-44640]:[00-59] – Terminate this connection after this many idle minutes and seconds. Limit is 44640:00.

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 Service Module card Report messages.

Values:
no – The Service Module card Reports are not wanted (default).
yes – The Service Module card Reports are wanted.

dsmrptperc
(Optional) Allows the client to specify what percent to use in the Service Module card Report. This overrides the system wide default Service Module card 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 SM Report. This overrides the system wide default SM Report frequency value for this one connection. This parameter is only meaningful if dsmrpt has been specified as yes.

> **Note:**
>
> If a connection is configured to send asynchronously SM Reports and processes a command that takes a long time, SM Reports will be suspended until after the long running command is completed. For example, if a client connects with a dsmrptfreq of 1 and then requests a retrieve of millions of DNs, the SM Reports will be suspended until the retrieve command has completed. Once the SM Reports resume they will continue at the configured frequency..

Values:
1 – 86400 – Send the Service Module card 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/[00-44640]:[00-59]>], [txnmode <normal/single>],
[dsmrpt <no/yes>], [dsmrptperc <1..100>], [dsmrptfreq <1..86400>])
```

> **Note:**
>
> The limit is 44640:00.

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 this topic indicate the result of the Connect request. See PDBI Message Error Codes for the recommended actions to help resolve the error related return codes.

Table 3-2 Connect Response Return Codes

Return Code	Text	Description	Data Section Contents
0	SUCCESS	Everything worked.	The assigned connection id and Active status are returned. `data (connectId ########, side active/standby)`
1003	ALREADY_CONNECTED	A connect request was sent on a socket that already had an established connection.	NONE

Table 3-2 (Cont.) Connect Response Return Codes

Return Code	Text	Description	Data Section Contents
1012	INVALID_VALUE	One of the fields specified had an invalid value.	The offending field is returned in the data section: `data (param <field label>)`
1023	UNKNOWN_VERSION	The specified version is not known or not supported.	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 Disconnectrequests 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. See PDBI Message Error Codes for the recommended actions to help resolve the error related return codes.

Table 3-3 Disconnect Response Return Codes

Return Code	Text	Description	Data Section Contents
0	**SUCCESS**	Everything worked.	**NONE**
1010	**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`.

Begin Transaction 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>])
```

Begin Transaction Response

The return codes in Table 3-4 indicate the result of the Begin Transaction request. See PDBI Message Error Codes for the recommended actions to help resolve the error related return codes.

Table 3-4 Begin Transaction Response Return Codes

Return Code	Text	Description	Data Section Contents
0	**SUCCESS**	Everything worked.	**NONE**
1005	**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>)`

Table 3-4 (Cont.) Begin Transaction Response Return Codes

Return Code	Text	Description	Data Section Contents
1006	**NO_WRITE_PERMISSION**	The **PDBI** client making the connection does not have **WRITE** access permissions.	**NONE**
1008	**STANDBY_SIDE**	An attempt to open a **write** transaction occurred on the Standby **PDBA**.	**NONE**
1010	**ACTIVE_TXN**	A **read** or **write** transaction is already open on this connection.	**NONE**
1012	**INVALID_VALUE**	One of the fields specified had an invalid value.	The offending field is returned in the data section: `data (param <field label>)`

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.

End Transaction 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 `creates` or `deletes`. This value is used by the Service Module cards to check consistency with the RTDB.

Request syntax :

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

End Transaction 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. See PDBI Message Error Codes for the recommended actions to help resolve the error related return codes.

Table 3-5 End Transaction Response Return Codes

Return Code	Text	Description	Data Section Contents
0	**SUCCESS**	**Database** update was successful.	If the transaction type was **write**, the new database level is returned. `data (dblevel #######)`
1009	**NO_ACTIVE_TXN**	There was no currently active transaction for this connection.	**NONE**
1017	**NO_UPDATES**	The **write** transaction had no successful updates. No database change occurs, and no new database level is returned.	**NONE**
1031	**DB_EXCEPTION**	An unexpected exception was thrown during the database commit. The entire transaction was rolled back to ensure predicable behavior. Contact Oracle.	**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. See PDBI Message Error Codes for the recommended actions to help resolve the error related return codes.

Table 3-6 Abort Transaction Response Return Code

Return Code	Text	Description	Data Section Contents
0	SUCCESS	Abort successful.	NONE
1009	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**

When a request to create a subscription fails due to provisioning checks added for the EPAP Provisioning Blocklist feature, the beginning (bprovbl) and ending (eprovbl) address string of the conflicting EPAP Provisioning Blocklist will be returned in the response.

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. If the IMSI conflicts with an entry in the EPAP Provisioning Blocklist table, the command will be rejected and use of the optional force parameter will not override this function.

> **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. Using the force parameter cannot override conflicts with a provisioning blocklist entry.

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 any of the DNs in the request conflict with a DN Block marked ineligible for splitting, the request is rejected. The use of optional **force** parameter would not change this behavior. 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 table. If removing the DNs results in the previous IMSI having no DNs, the IMSI with no DN is not deleted. If the IMSI or any of the DNs specified conflicts with an entry in the EPAP Provisioning Blocklist table, the command will be rejected and use of the optional **force** parameter will not override this function.

> **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. Using the force parameter cannot override conflicts with a provisioning blocklist entry.

2. When the IMSI does not already exist, the **sp** parameter is required. If the command specifies new DNs for an existing, IMSI, the **sp** parameter is not required and the existing **sp** is not changed.

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.

If any of the DNs specified conflicts with an entry in the EPAP Provisioning Blocklist table or with a DN Block marked ineligible for splitting, the command will be rejected and use of the optional force parameter will not override this function.

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, and the enter will succeed.

Guidelines for using the TIF Number Substitution parameter nsdn and st can be found in TIF Number Substitution Relationships . The default value for subscriber type st is Public. The rtrv_sub() command will not list st in its response for records that have the default value Public.

The **IDP** calling and called party blocklist parameters (`cgbl/cdbl`) are optional and have a default value of `no`. The GRN entity can optionally be associated with the DN. The **GRN** is used as redirection digits if either the calling or called party blocklist parameters are set to `yes`. The IDP blocklist feature is not related to the EPAP Provisioning Blocklist feature.

The (**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.portability type (

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 a value `1` (ported out) and also 3 (prepaid 1). However, there is no effect on the G-Port function if `pt` type 3 through 35 is specified. In these cases, if a message is being processed for G-Port and the DN block matches with a `pt` type 3 through 35, G-Port considers it the same as if the `pt` type = `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, A-Port, IS41 GSM Migration, and PPSMS. For G-Port and A-Port, it controls number Portability Status encoding in SRI acks. For IS41 GSM Migration, it identifies whether a subscriber has or has not migrated from IS41 to GSM, (maintaining a single GSM handset). For PPSMS, it identifies a DN as one of two types needing PPSMS intercept.

Values:
none – no status (default = none)
0 – not known to be ported, migrated to IS41 or non-migrated IS41 sub (used for IS41 GSM Migration)
1 – own number ported out (used for G-Port and A-Port)
2 – foreign number ported to foreign network (used for G-Port and A-Port)
3 – prepaid 1 (used by PPSMS)
4 – prepaid 2 (used by PPSMS)
5 – migrated to GSM (used for IS41 GSM Migration)
6 – prepaid 3 (used by PPSMS)
7 – prepaid 4 (used by PPSMS)
8 – prepaid 5 (used by PPSMS)
9 – prepaid 6 (used by PPSMS)
10 – prepaid 7 (used by PPSMS)
11 – prepaid 8 (used by PPSMS)
12 – prepaid 9 (used by PPSMS)
13 – prepaid 10 (used by PPSMS)
14 – prepaid 11 (used by PPSMS)
15 – prepaid 12 (used by PPSMS)
16 – prepaid 13 (used by PPSMS)
17 – prepaid 14 (used by PPSMS)
18 – prepaid 15 (used by PPSMS)
19 – prepaid 16 (used by PPSMS)
20 – prepaid 17 (used by PPSMS)
21 – prepaid 18 (used by PPSMS)
22 – prepaid 19 (used by PPSMS)
23 – prepaid 20 (used by PPSMS)
24 – prepaid 21 (used by PPSMS)
25 – prepaid 22 (used by PPSMS)
26 – prepaid 23 (used by PPSMS)
27 – prepaid 24 (used by PPSMS)
28 – prepaid 25 (used by PPSMS)
29 – prepaid 26 (used by PPSMS)
30 – prepaid 27 (used by PPSMS)
31 – prepaid 28 (used by PPSMS)
32 – prepaid 29 (used by PPSMS)
33 – prepaid 30 (used by PPSMS)
34 – prepaid 31 (used by PPSMS)

35 – prepaid 32 (used by PPSMS)
36 – not identified to be ported

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.

vms
(Optional) Specifies which Voicemail Server the DN(s) are on and corresponds to the E.164 address of the voicemail server. The VMS must correspond to an existing VMS entity.

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

grn
(Optional) Specifies which Generic Routing Number the DN(s) are on and corresponds to the E.164 address used when the EAGLE “NE Query Only Option” has been turned on. The GRN must correspond to an existing GRN entity.

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

asd
(Optional) Additional Subscriber Data to be associated with the DN.

Values
1 to 10 hexadecimal digits expressed using ASCII characters.

st
(Optional) The subscriber type for created DNs.

Values
A decimal number in the range
0 - public
1 - private

nsdn
A TIF Number Substitution DN (specified in international format).
Values:

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

cgbl
(Optional) IDP calling party blocklist.

Values
no - IDP calling party blocklist is disabled.

yes - IDP calling party blocklist is enabled

cdbl
(Optional) IDP called party blocklist.

Values
no - IDP called party blocklist is disabled.

yes - IDP called party blocklist is enabled.

lsblset
(Optional) A TIF Linkset based blocklist parameter that is used to decide if the DN needs to be blocklisted or not.

Values
a number from **1** to **255**.

By default, no value is selected.

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**.
2. Conflicts with an EPAP Provisioning blocklist entry and cannot be circumvented using the force parameter.
3. It is not valid to specify more than 2 Network Entity types (**sp**,**rn**,**vms**,**grn**).
4. Total command length must not exceed 247 characters.

> **Note:**
>
> Entering commands that exceed this length will result in the PDBI_CMD_LENGTH_EXCEEDED error (value 1045). In order to avoid this, remove unnecessary characters (including white space and parameters that are specified as the default value). If necessary, consider performing this provisioning in two steps by using an enter command followed by an update command.

5. The DN given by**nsdn** must exist.
6. Subscriber type**st** must be defined when using the**nsdn** parameter
7. It is not legal to specify the 2 Network Entities**asd**, and**nsdn**.

Request syntax:

```
Ent_sub([iid XXXXX,] dn XXXXX, . . ., dn XXXXX, [pt <none/0/1/2/3/4……35>,]
[st <0/1>,]
[nsdn XXXXX,] [sp XXXXX,] [rn XXXXX,] [vms XXXXX,] [grn XXXXX,] [asd XXXXX,]
[cgbl < no/yes>,] [cdbl <no/yes>,] [lsblset <1,2……255>,] [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. When the DN Block Self Healing feature is on, if the new DN Block to be added is a subset of the existing block with different properties and the existing DN Block has the splitting ability set to `yes`. The original DN Block will be split to accommodate the new DN Block. A new parameter `split` is added for this purpose. The default value for split parameter is `yes`.The **force** parameter is not supported for this command.

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.

Guidelines for using the TIF Number Substitution (TIF NS) parameter nsdn and st can be found in TIF Number Substitution Relationships . The default value for subscriber type st is Public. The `rtrv_sub()` command will not list st in its response for records that have the default value Public.

Guidelines for using the DN Block Self Healing parameter `split` can be found in DN Block Self Healing. The default value for the `split` parameter is `yes`. The `rtrv_sub()` command does not list `split` in its response for records that have the default value `yes`.

The IDP calling and called party blocklist parameters (`cgbl/cdbl`) are optional and have a default value of `no`. The GRN entity can optionally be associated with the DN Block. The GRN is used as redirection digits if either the calling or called party blocklist parameters are set to`yes`. The IDP A-Party Blocklist feature is not related to the EPAP Provisioning Blocklist feature.

The `pt` parameter is used to define the prepaid type. The prepaid type (value of 3 through 35) 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 function if `pt` type 3 through 35 is specified. In these cases, if a message is being processed for G-Port and the DN block matches with a `pt` type 3 through 35, 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**), **V-Flex**, and INP features; the portability type parameter applies only to G-Port and PPSMS.

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, A-Port, IS41 GSM Migration, and PPSMS. For G-Port and A-Port, it controls number Portability Status encoding in SRI acks. For IS41 GSM Migration, it identifies whether a subscriber has or has not migrated from IS41 to GSM, (maintaining a single GSM handset). For PPSMS, it identifies a DN as one of two types needing PPSMS intercept.

Values:
none – no status (default = none)
0 – not known to be ported, migrated to IS41 or non-migrated IS41 sub (used for IS41 GSM Migration)
1 – own number ported out (used for G-Port and A-Port)
2 – foreign number ported to foreign network (used for G-Port and A-Port)
3 – prepaid 1 (used by PPSMS)
4 – prepaid 2 (used by PPSMS)
5 – migrated to GSM (used for IS41 GSM Migration)
6 – prepaid 3 (used by PPSMS)
7 – prepaid 4 (used by PPSMS)
8 – prepaid 5 (used by PPSMS)
9 – prepaid 6 (used by PPSMS)
10 – prepaid 7 (used by PPSMS)
11 – prepaid 8 (used by PPSMS)
12 – prepaid 9 (used by PPSMS)
13 – prepaid 10 (used byPPSMS)
14 – prepaid 11 (used by PPSMS)
15 – prepaid 12 (used by PPSMS)
16 – prepaid 13 (used by PPSMS)
17 – prepaid 14 (used by PPSMS)
18 – prepaid 15 (used by PPSMS)
19 – prepaid 16 (used by PPSMS)
20 – prepaid 17 (used by PPSMS)
21 – prepaid 18 (used by PPSMS)
22 – prepaid 19 (used by PPSMS)
23 – prepaid 20 (used by PPSMS)
24 – prepaid 21 (used by PPSMS)
25 – prepaid 22 (used by PPSMS)
26 – prepaid 23 (used by PPSMS)
27 – prepaid 24 (used by PPSMS)
28 – prepaid 25 (used by PPSMS)
29 – prepaid 26 (used by PPSMS)
30 – prepaid 27 (used by PPSMS)
31 – prepaid 28 (used by PPSMS)
32 – prepaid 29 (used by PPSMS)
33 – prepaid 30 (used by PPSMS)

34 – prepaid 31 (used by PPSMS)
35 – prepaid 32 (used by PPSMS)
36 – not identified to be ported

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. G-Flex-only customers should not use **RNs**.

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

vms
(Optional) Specifies which Voicemail Server the **DN**(s) are on and corresponds to the E.164 address of the voicemail server. The VMS must correspond to an existing VMS entity.

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

grn
(Optional) Specifies which Generic Routing Number the DN(s) are on and corresponds to the E.164 address used when the EAGLE “NE Query Only Option” has been turned on. The GRN must correspond to an existing GRN entity.

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

asd
(Optional) Additional Subscriber Data to be associated with the DN Block.

Values
1 to 10 hexadecimal digits expressed using ASCII characters.

st
(Optional) The subscriber type for created DNs.

Values
A decimal number in the range
0 - public
1 - private

nsdn
A TIF NS DN (specified in international format).
Values:

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

cgbl
(Optional) IDP calling party blocklist.

Values
no - IDP calling party blocklist is disabled.

yes - IDP calling party blocklist is enabled

cdbl
(Optional) IDP called party blocklist.

Values
no - IDP called party blocklist is disabled.

yes - IDP called party blocklist is enabled.

lsblset
(Optional) A TIF Linkset based Blocklist parameter that is used to decide if the DN needs to be blocklisted or not.

Values
a number from **1** to **255**.

By default, no value is selected.

split
(Optional) Specifies whether a DN Block is eligible for splitting.

Values
yes – splitting is enabled for DN Block self heal (default= yes).

no – splitting is disabled for DN Block self heal.

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**.
3. It is not valid to specify more than 2 Network Entity types (**sp**,**rn**,**vms**,**grn**).
4. The DN given by **nsdn** must exist.
5. Subscriber type **st** must be defined when using the **nsdn** parameter
6. It is not legal to specify the 2 Network Entities**asd**, and**nsdn**.

The DN given by **nsdn** must exist. The DN given by **nsdn** must exist.

Request syntax :

```
Ent_sub([iid XXXXX,] bdn XXXXX, edn XXXXX, [pt <none/0/1/2/3/4/….35>,] [st
<0/1>,]
[nsdn XXXXX,] [sp XXXXX,] [rn XXXXX,] [vms XXXXX,] [grn XXXXX,] [asd XXXXX,]
[cgbl < no/yes>,] [cdbl <no/yes>,][lsblset <1,2……255>,]  [split yes/no,]
[force yes/no,] [timeout <0..3600>])
```

Create Subscription Responses

The return codes in Table 3-7 might result from the Create Subscription request. See PDBI Message Error Codes for the recommended actions to help resolve the error related return codes.

Table 3-7 Create Subscription Response Return Codes

Return Code	Text	Description	Data Section Contents
0	SUCCESS	Everything worked.	NONE
1005	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>)`
1006	NO_WRITE_PERMISSION	The PDBI client making request does not have write access permissions.	NONE
1009	NO_ACTIVE_TXN	There was no currently active transaction for this connection.	NONE
1011	WRITE_IN_READ_TXN	The `create` command was sent on a **read** transaction.	**NONE**
1012	INVALID_VALUE	One of the fields specified had an invalid value.	The offending field is returned in the data section: `data (param <field label>)`

1014	CONFLICT_FOUND	An entry was found already in database matching an element of this request. If `force yes` parameter is used and this option is supported for this dta type, 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)`
			data (bprovbl XXXX, eprovBL XXXX)

1017	NO_UPDATES	The database already contains data in request. No update was necessary.	NONE

Table 3-7 (Cont.) Create Subscription Response Return Codes

Return Code	Text	Description	Data Section Contents
1021	NE_NOT_FOUND	The specified **NE** does not exist.	NONE
1027	IMSI_DN_LIMIT	The addition of DNs specified in request would cause IMSI to have more than eight DNs	NONE
1029	TXN_TOO_BIG	This request would cause current transaction to be larger than limit.	NONE
1032	MAX_IMSI_LIMIT	Could not add new IMSI to the database. Adding a new IMSI would exceed the max allowed IMSIs.	NONE
1033	MAX_DN_LIMIT	Could not add new DN(s) to the database. Adding a new DN(s) would exceed the max allowed DNs.	NONE
1034	MAX_DNBLK_LIMIT	Could not add new DN Block to the database. Adding a new DN Block would exceed the max allowed DN Blocks.	NONE
1044	SUB_NE_LIMIT	There are too many network entities for a DN or DN Block.	NONE
1046	MAX_ASD_LIMIT	The database to exceeds the global limit of unique ASD records.	NONE
1047	DN_NOT_FOUND	The DN specified by nsdn does not exist.	The offending existing element will be returned. data (nsdn XXXXX)
1048	MAX_ASSOCIATIONS	This request contains too many associations for a DN or DN Block. The error is raised when TIF Number Substitution is combined with other associations and distinct from SUB_NE_LIMIT.	NONE
1049	UNRESOLVED_DEPENDENCY	This record is referred to by other records.	When using force to enter a pre-existing IMSI that has a DN with unresolved dependencies, the DN and the count of each dependency are returned. If more than one DN has unresolved dependencies, only one will be returned in this error. Data (dn XXXXX, counts([nsdn #####]))
1050	INCOMPATIBLE_ST	The specified ST value conflicts with the ST value of NSDN.	The ST value of NSDN will be returned in the data section data (st <0/1>)

Table 3-7 (Cont.) Create Subscription Response Return Codes

Return Code	Text	Description	Data Section Contents
1051	INCOMPATIBLE_ROP	The GRN ROP value conflicts with the GMT value for the specified ASD.	The GRN value for the ASD will be returned in the data section. data (grn #####)
1052	DNB_SAME_PROPERTIES	The new DN Block requested by the operator is a subset of an existing block with same properties.	The bdn and edn of conflicting block will be returned.
1053	MULTI_DNB_CONFLICT	The new DN Block could not be added to the database as multiple conflicting DN Blocks were found within given bdn-edn.	NONE
1054	DNB_SPLIT_NOT_ALLOWED	The new DN Block tries to split an existing DN Block that is not allowed to be split or a new individual DN is entered within an existing DN Block that is not allowed to be split.	The bdn and edn of conflicting block will be returned.
1058	IMSI_FULL	Could not add new IMSI to the database. Adding new IMSI would exceed the supported SMxG card size. The other supported subscriptions are allowed if space is available on the corresponding SMxG card.	NONE
1060	DN_FULL	Could not add new DN to the database. Adding new DN would exceed the supported SMxG card size. The other supported subscriptions are allowed if space is available on the corresponding SMxG card.	NONE
1061	ASD_FULL	Could not add new ASD to the database. Adding new ASD would exceed the supported SMxG card size. The other supported subscriptions are allowed if space is available on the corresponding SMxG card.	NONE
1062	IMSI_ENT_NOT_ALLOWED	Could not add new IMSI to the database. The RTDB is either down or incoherent. Adding new IMSI requires creation of new IMSI table which may lead to over-allocation.	NONE

Update Subscription

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.

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 Network Entity (SP, RN, VMS,or GRN) 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 will result in a DN that is not associated with a Network Entity. If a DN was already associated with a sp or rn, specifying a new sp or rn will result in a replacement. Any update in which a Network Entity type is specified will only be allowed if the DN has not reached its maximum of 2 NEs per **DN**.

If a DN has a NE association and an update command specifies a new NE of the same NE type, a replacement is attempted. Updating a DN by setting a NE type (sp/rn/vms/grn) to the value none, results in a DN that is no longer associated with that NE. If a DN has an existing NE association and an update command specifies a NE of a different NE type, will attempt to add a new NE association. To replace a NE association with a new NE association of a different type, remove the old NE (specifying value none) and enter the new NE. This is done in a single request or as two separate requests (one command to remove the old NE and one command to add the new NE). Any update in which a Network Entity type is specified is only allowed if the DN has not reached its maximum of 2 NEs.

Guidelines for using the TIF Number Substitution parameter nsdn and st can be found in TIF Number Substitution Relationships . The default value for subscriber type st is Public. The rtrv_sub() command will not list st in its response for records that have the default value Public.

The **IDP** calling and called party blocklist parameters (cgbl/cdbl) are optional and have a default value of no. The GRN entity can optionally be associated with the DN. The **GRN** is used as redirection digits if either the calling or called party blocklist parameters are set to yes. The IDP A-Party Blocklist feature is not related to the EPAP Provisioning Blocklist feature.

If an update requires a replacement other than sp/rn for sp/rn, the user may specify the new Network Entity and the Network entity to be removed should be specified as a value of none. This may be done in a single request or as two separate requests (1 to remove the old **NE** and 1 to add the new **NE**).

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, A-Port, IS41 GSM Migration, and PPSMS. For G-Port and A-Port, it controls number Portability Status encoding in SRI acks. For IS41 GSM Migration, it identifies whether a subscriber has or has not migrated from IS41 to GSM, (maintaining a single GSM handset). For PPSMS, it identifies a DN as one of two types needing PPSMS intercept.

Values:
none – no status (default = none)
0 – not known to be ported, migrated to IS41 or non-migrated IS41 sub (used for IS41 GSM Migration)
1 – own number ported out (used for G-Port and A-Port)
2 – foreign number ported to foreign network (used for G-Port and A-Port)
3 – prepaid 1 (used by PPSMS)
4 – prepaid 2 (used by PPSMS)
5 – migrated to GSM (used for IS41 GSM Migration)
6 – prepaid 3 (used by PPSMS)
7 – prepaid 4 (used by PPSMS)
8 – prepaid 5 (used by PPSMS)
9 – prepaid 6 (used by PPSMS)
10 – prepaid 7 (used by PPSMS)
11 – prepaid 8 (used by PPSMS)
12 – prepaid 9 (used by PPSMS)
13 – prepaid 10 (used by PPSMS)
14 – prepaid 11 (used by PPSMS)
15 – prepaid 12 (used by PPSMS)
16 – prepaid 13 (used by PPSMS)
17 – prepaid 14 (used by PPSMS)
18 – prepaid 15 (used by PPSMS)
19 – prepaid 16 (used by PPSMS)
20 – prepaid 17 (used by PPSMS)
21 – prepaid 18 (used by PPSMS)
22 – prepaid 19 (used by PPSMS)
23 – prepaid 20 (used by PPSMS)
24 – prepaid 21 (used by PPSMS)
25 – prepaid 22 (used by PPSMS)
26 – prepaid 23 (used by PPSMS)
27 – prepaid 24 (used by PPSMS)
28 – prepaid 25 (used by PPSMS)
29 – prepaid 26 (used by PPSMS)
30 – prepaid 27 (used by PPSMS)
31 – prepaid 28 (used by PPSMS)
32 – prepaid 29 (used by PPSMS)
33 – prepaid 30 (used by PPSMS)
34 – prepaid 31 (used by PPSMS)
35 – prepaid 32 (used by PPSMS)
36 – not identified to be ported

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.

vms
(Optional) Specifies which Voicemail Server the **DN**(s) are on and corresponds to the E.164 address of the voicemail server. The VMS must correspond to an existing VMS entity.

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

grn
(Optional) Specifies which Generic Routing Number the DN(s) are on and corresponds to the E.164 address used when the EAGLE “NE Query Only Option” has been turned on. The GRN must correspond to an existing GRN entity.

Values:
1 to 15 hexadecimal digits expressed using **ASCII** characters.
none – Sets the rms to not point to any GRN 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.

asd
(Optional) Additional Subscriber Data to be associated with the DN Block. Leading zeros are significant.

Values
1 to 10 hexadecimal digits expressed using ASCII characters. Leading zeros are significant.
none – Removes additional subscriber data from the DN.

st
(Optional) The subscriber type for created DNs.

Values
A decimal number in the range
0 - public
1 - private

nsdn
A TIF NS DN (specified in international format).
Values:

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

cgbl
(Optional) IDP calling party blocklist.

Values
no - IDP calling party blocklist is disabled.

yes - IDP calling party blocklist is enabled

cdbl
(Optional) IDP called party blocklist.

Values
no - IDP called party blocklist is disabled.

yes - IDP called party blocklist is enabled.

lsblset
(Optional) A TIF Linkset based Blocklist parameter that is used to decide if the DN needs to be blocklisted or not.

Values
a number from **1** to **255**.

By default, no value is selected.

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 create more than 2 Network Entity associations.
2. The DN given by **nsdn** must exist.
3. Cannot update a DN to have the 2 Network Entities **asd** and **nsdn** together.
4. 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 <none/0/1/2/3//….35>,] [st <0/1>,]
[nsdn XXXXX,]
[sp XXXXX,] [rn XXXXX,] [vms XXXXX,] [grn XXXXX,] [asd XXXXX,] [cgbl
<no/yes>,]
[cdbl < no/yes>,][lsblset <1,2……255>,]  [force yes/no,] [timeout
<0..3600>])
```

Move an existing DN to an Existing IMSI

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

If the **DN** is already associated with an **IMSI**, it will be removed from that **IMSI** and associated with the new **IMSI**. The original **IMSI** will not be removed, even if this results in it having no **DNs** associated with it.

If the DN is already associated with a vms or grn, those NE associations will be removed when the DN is moved to the specified IMSI. IMSIs and DNs on IMSIs can only be associated with 1 NE.

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 modifies 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. If a DN block has a NE association and the update command specifies a new NE of the same NE type, a replacement is attempted. Updating a `DN` block by setting a NE type (`sp`/`rn`/`vms`/`grn`) to the value `none`, results in a `DN` that is no longer associated with that NE. If a `DN` block has an existing NE association and the update command specifies a NE of a different NE type, will attempt to add a new NE association. To replace a NE association with a new NE association of a different type, remove the old NE (specifying value `none`) and enter the new NE. This is done in a single request or as two separate requests (one command to remove the old NE and one command to add the new NE). Any update in which a Network Entity type is specified is only allowed if the `DN` block has not reached its maximum of 2 NEs.

Guidelines for using the Number Substitution parameter nsdn and st can be found in TIF Number Substitution Relationships . The default value for subscriber type st is Public. The `rtrv_sub()` command will not list st in its response for records that have the default value Public.

The **IDP** calling and called party blocklist parameters (`cgbl`/`cdbl`) are optional and have a default value of `no`. The GRN entity can optionally be associated with the DN. The **GRN** is used as redirection digits if either the calling or called party blocklist parameters are set to `yes`. The IDP A-Party blocklist feature is not related to the EPAP Provisioning Blocklist feature.

The DN Block Self Healing parameter (`split`) is optional and has a default value of `yes`. This parameter can be modified by the user at any time without affecting any other attribute of the DN Block. Guidelines for the DN Block Self Healing feature can be found in DN Block Self Healing. The `rtrv_sub()` command will not list `split` in its response for records that have the default value `yes`.

> **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, A-Port, IS41 GSM Migration, and PPSMS. For G-Port and A-Port, it controls number Portability Status encoding in SRI acks. For IS41 GSM Migration, it identifies whether a subscriber has or has not migrated from IS41 to GSM, (maintaining a single GSM handset). For PPSMS, it identifies a DN as one of two types needing PPSMS intercept.

Values:
none – no status (default = none)
0 – not known to be ported, migrated to IS41 or non-migrated IS41 sub (used for IS41 GSM Migration)
1 – own number ported out (used for G-Port and A-Port)
2 – foreign number ported to foreign network (used for G-Port and A-Port)
3 – prepaid 1 (used by PPSMS)
4 – prepaid 2 (used by PPSMS)
5 – migrated to GSM (used for IS41 GSM Migration)
6 – prepaid 3 (used by PPSMS)
7 – prepaid 4 (used by PPSMS)
8 – prepaid 5 (used by PPSMS)
9 – prepaid 6 (used by PPSMS)
10 – prepaid 7 (used by PPSMS)
11 – prepaid 8 (used by PPSMS)
12 – prepaid 9 (used by PPSMS)
13 – prepaid 10 (used by PPSMS)
14 – prepaid 11 (used by PPSMS)
15 – prepaid 12 (used by PPSMS)

16 – prepaid 13 (used by PPSMS)
17 – prepaid 14 (used by PPSMS)
18 – prepaid 15 (used by PPSMS)
19 – prepaid 16 (used by PPSMS)
20 – prepaid 17 (used by PPSMS)
21 – prepaid 18 (used by PPSMS)
22 – prepaid 19 (used by PPSMS)
23 – prepaid 20 (used by PPSMS)
24 – prepaid 21 (used by PPSMS)
25 – prepaid 22 (used by PPSMS)
26 – prepaid 23 (used by PPSMS)
27 – prepaid 24 (used by PPSMS)
28 – prepaid 25 (used by PPSMS)
29 – prepaid 26 (used by PPSMS)
30 – prepaid 27 (used by PPSMS)
31 – prepaid 28 (used by PPSMS)
32 – prepaid 29 (used by PPSMS)
33 – prepaid 30 (used by PPSMS)
34 – prepaid 31 (used by PPSMS)
35 – prepaid 32 (used by PPSMS)
36 – not identified to be ported

sp
(Optional) Specifies which **SP** the DN Block 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 SP 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 RN entity.

vms
(Optional) Specifies which Voicemail Server the **DN**(s) are on and corresponds to the E.164 address of the voicemail server. The VMS must correspond to an existing VMS entity.

Values:
1 to 15 hexadecimal digits expressed using **ASCII** characters.
none – Sets the **vms** to not point to any VMS entity.

grn
(Optional) Specifies which Generic Routing Number the DN(s) are on and corresponds to the E.164 address used when the EAGLE “NE Query Only Option” has been turned on. The GRN must correspond to an existing GRN entity.

Values:
1 to 15 hexadecimal digits expressed using **ASCII** characters.
none – Sets the **grn** to not point to any GRN entity.

asd
(Optional) Additional Subscriber Data to be associated with the DN Block.

Values
1 to 10 hexadecimal digits expressed using ASCII characters. Leading zeros are significant.
none – Removes additional subscriber data from the DN.

st
(Optional) The subscriber type for created DNs.

Values
A decimal number in the range
0 - public
1 - private

nsdn
A TIF Number Substitution DN (specified in international format).
Values:

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

cgbl
(Optional) IDP calling party blocklist.

Values
no - IDP calling party blocklist is disabled.

yes - IDP calling party blocklist is enabled

cdbl
(Optional) IDP called party blocklist.

Values
no - IDP called party blocklist is disabled.

yes - IDP called party blocklist is enabled.

lsblset
(Optional) A TIF Linkset based blocklist parameter that is used to decide if the DN needs to be blocklisted or not.

Values
a number from **1** to **255**.

By default, no value is selected.

split
(Optional) Specifies whether a DN Block is eligible for splitting.

Values
yes – splitting is enabled for DN Block self heal (default = yes).

no – splitting is disabled for DN Block self heal.

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 create more than 2 Network Entity associations.
3. The DN given by `nsdn` must exist.
4. A DN Block cannot be updated to have two Network Entities, `asd` and `nsdn`, together.
5. See TIF Number Substitution Relationships for complete rules.
6. See DN Block Self Healing for complete rules.

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

Request syntax :

```
Upd_sub([iid XXXXX,] bdn XXXXX, edn XXXXX, [pt <none/0/1/2/3//….35>,] [st
<0/1>,]
[nsdn XXXXX,] [sp XXXXX,] [rn XXXXX,] [vms XXXXX,] [grn XXXXX,] [asd XXXXX,]
[cgbl < no/yes>,] [cdbl <no/yes>,] [lsblset <1,2……255>,] [split <no/yes>,]
[timeout <0..3600>])
```

Update Subscription Responses

The return codes listed in Table 3-8 indicate the result of the Update Subscription request. See PDBI Message Error Codes for the recommended actions to help resolve the error related return codes.

Table 3-8 Update Subscription Response Return Codes

Return Code	Text	Description	Data Section Contents
0	SUCCESS	Everything worked.	**NONE**
1012	INVALID_VALUE	One of the fields specified had an invalid value.	The offending field is returned in data section: **data (param <field label>)**
1011	WRITE_IN_ READ_TXN	The command was sent on a **read only** transaction.	NONE
1013	NOT_FOUND	The requested **DN**, **DN** block, or **IMSI** was not found.	NONE
1027	IMSI_DN_LIMIT	The **IMSI** already has maximum number of **DNs**.	NONE

Table 3-8 (Cont.) Update Subscription Response Return Codes

Return Code	Text	Description	Data Section Contents
1029	TXN_TOO_BIG	This request would cause current transaction to be larger than limit.	NONE
1017	NO_UPDATES	**Database** already contains date in request. No update is necessary.	NONE
1009	NO_ACTIVE_TXN	There is no currently active transaction for this connection.	NONE
1021	NE_NOT_FOUND	**NE** specified does not exist.	NONE
1006	NO_WRITE_PERMISSION	**PDBI** client making request does not have write access permissions.	NONE
1005	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>)`
1044	SUB_NE_LIMIT	This request would cause the DN or DN Block to have more than the maximum of 2 NE associations	NONE
1046	MAX_ASD_LIMIT	This request would cause the database to exceed the global limit of unique ASD records.	NONE
1047	DN_NOT_FOUND	The DN specified by nsdn does not exist.	The offending existing element will be returned. data (nsdn XXXXX)
1048	MAX_ASSOCIATIONS	This request contains too many associations for a DN or DN Block. The error is raised when Number Substitution is combined with other associations and distinct from SUB_NE_LIMIT.	NONE

Table 3-8 (Cont.) Update Subscription Response Return Codes

Return Code	Text	Description	Data Section Contents
1049	UNRESOLVED_DEPENDENCY	This record is referred to by other records.	When using force to enter a pre-existing IMSI that has a DN with unresolved dependencies, the DN and the count of each dependency are returned. If more than one DN has unresolved dependencies, only one will be returned in this error. Data (dn XXXXX, counts([ns dn #####]))
1050	INCOMPATIBLE_ST	The specified ST value conflicts with the ST value of NSDN.	The ST value of NSDN will be returned in the data section data (st <0/1>)

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.

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.

If the DN Block Self Healing feature is on, an attempt to delete the subrange when master range properties mismatch or an attempt to delete the master range itself will be rejected. Guidelines for the DN Block Self Healing feature can be found in DN Block Self Healing.

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>])
```

Delete Subscription Responses

The return codes listed in Table 3-9 indicate the result of the Delete Subscription request. See PDBI Message Error Codes for the recommended actions to help resolve the error related return codes.

Table 3-9 Delete Subscription Response Return Codes

Return Code	Text	Description	Data Section Contents
0	**SUCCESS**	Everything worked.	**NONE**
1011	**WRITE_IN**_READ_TXN	The command was sent on a **read only** transaction.	**NONE**
1013	**NOT**_FOUND	The requested **DN**, **DN** block, or **IMSI** was not found.	NONE
1029	**TXN_TOO**_BIG	This request would cause current transaction to be larger than the limit.	NONE
1009	**NO_ACTIVE_TXN**	There is no currently active transaction for this connection.	NONE
1006	**NO_WRITE**_PERMISSION	**PDBI** client making request does not have write access permissions.	NONE

Table 3-9 (Cont.) Delete Subscription Response Return Codes

Return Code	Text	Description	Data Section Contents
1005	**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>)`
1049	UNRESOLVED_DEPENDENCY	This record is referred to by other records.	When using force to enter a pre-existing IMSI that has a DN with unresolved dependencies, the DN and the count of each dependency are returned. If more than one DN has unresolved dependencies, only one will be returned in this error. Data `(dn XXXXX, counts([nsdn #####]))`
1055	DNB_PARENT_PROPERTY_MISMATCH	Fragments of a master range have differing attributes. No automated resolution is possible to coalesce these records to satisfy a delete command.	The bdn and edn of two differing master range fragments are returned.
1056	DNB_DLT_NOT_ALLOWED	Fragments of a master range cannot be deleted while subranges are present	The bdn and edn of a subrange are returned.

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.

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.

pt
(Optional) The portability type for the created DN. This field is only used by G-Port, A-Port, IS41 GSM Migration, and PPSMS. For G-Port and A-Port, it controls number Portability Status encoding in SRI acks. For IS41 GSM Migration, it identifies whether a subscriber has or has not migrated from IS41 to GSM, (maintaining a single GSM handset). For PPSMS, it identifies a DN as one of two types needing PPSMS intercept.

Values:
none – no status (default = none)
0 – not known to be ported, migrated to IS41 or non-migrated IS41 sub (used for IS41 GSM Migration)
1 – own number ported out (used for G-Port and A-Port)
2 – foreign number ported to foreign network (used for G-Port and A-Port)
3 – prepaid 1 (used by PPSMS)
4 – prepaid 2 (used by PPSMS)
5 – migrated to GSM (used for IS41 GSM Migration)
6 – prepaid 3 (used by PPSMS)
7 – prepaid 4 (used by PPSMS)
8 – prepaid 5 (used by PPSMS)
9 – prepaid 6 (used by PPSMS)
10 – prepaid 7 (used by PPSMS)
11 – prepaid 8 (used by PPSMS)
12 – prepaid 9 (used by PPSMS)
13 – prepaid 10 (used by PPSMS)
14 – prepaid 11 (used by PPSMS)
15 – prepaid 12 (used by PPSMS)
16 – prepaid 13 (used by PPSMS)
17 – prepaid 14 (used by PPSMS)
18 – prepaid 15 (used by PPSMS)
19 – prepaid 16 (used by PPSMS)
20 – prepaid 17 (used by PPSMS)
21 – prepaid 18 (used by PPSMS)
22 – prepaid 19 (used by PPSMS)
23 – prepaid 20 (used by PPSMS)
24 – prepaid 21 (used by PPSMS)
25 – prepaid 22 (used by PPSMS)
26 – prepaid 23 (used by PPSMS)
27 – prepaid 24 (used by PPSMS)
28 – prepaid 25 (used by PPSMS)
29 – prepaid 26 (used by PPSMS)
30 – prepaid 27 (used by PPSMS)
31 – prepaid 28 (used by PPSMS)
32 – prepaid 29 (used by PPSMS)
33 – prepaid 30 (used by PPSMS)
34 – prepaid 31 (used by PPSMS)
35 – prepaid 32 (used by PPSMS)
36 – not identified to be ported

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.

vms
(Optional) Specifies which Voicemail Server the **DN**(s) are on and corresponds to the E.164 address of the voicemail server. The VMS must correspond to an existing VMS entity.

Values:
1 to 15 hexadecimal digits expressed using **ASCII** characters.
none – Sets the **vms** to not point to any network entity.

grn
(Optional) Specifies which Generic Routing Number the DN(s) are on and corresponds to the E.164 address used when the EAGLE “NE Query Only Option” has been turned on. The GRN must correspond to an existing GRN entity.

Values:
1 to 15 hexadecimal digits expressed using **ASCII** characters.
none – Sets the **grn** to not point to any network entity.

asd
(Optional) Additional Subscriber Data to be associated with the DN Block.

Values
1 to 10 hexadecimal digits expressed using ASCII characters. Leading zeros are significant.
none – Removes additional subscriber data from the DN.

st
(Optional) The subscriber type for created DNs.

Values
A decimal number in the range
0 - public
1 - private

nsdn
A Number Substitution DN (specified in international format).
Values:

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

cgbl
(Optional) IDP calling party blocklist.

Values
no - IDP calling party blocklist is disabled.

yes - IDP calling party blocklist is enabled

cdbl
(Optional) IDP called party blocklist.

Values
no - IDP called party blocklist is disabled.

yes - IDP called party blocklist is enabled.

lsblset
(Optional) A TIF Linkset based Blocklist parameter that is used to decide if the DN needs to be blocklisted or not.

Values
a number from **1** to **255**.

By default, no value is selected.

split
(Optional) Specifies the DN Block splitting ability. This parameter filters the request to retrieve only those DN Blocks that have the specified split value. This parameter must be specified in order to be used as a filter. If this parameter is not specified, filtering on the split value is not performed.

Values
no – DN Block splitting is disabled.

yes – DN Block splitting is enabled.

data
(Optional) Lets the requester specify the type of output data to be returned. See Retrieve Subscription Data Responses 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**

Request syntax :

```
Rtrv_sub([iid XXXXX,] dn XXXXX, [type <block/single>,] [pt <none/0/1/2/3/4/
….35>],
[st <0/1>,] [nsdn XXXXX,] [sp XXXXX,] [rn XXXXX,] [vms XXXXX,] [grn XXXXX,]
```

```
[asd XXXXX,]
[cgbl <no/yes>,] [cdbl <no/yes>,] [lsblset <1,2……255>,] [split <no/
yes>,] [data <all/neonly/count>,] [num <1..40000000>])
```

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, A-Port, IS41 GSM Migration, and PPSMS. For G-Port and A-Port, it controls number Portability Status encoding in SRI acks. For IS41 GSM Migration, it identifies whether a subscriber has or has not migrated from IS41 to GSM, (maintaining a single GSM handset). For PPSMS, it identifies a DN as one of two types needing PPSMS intercept.

Values:
none – no status (default = none)
0 – not known to be ported, migrated to IS41 or non-migrated IS41 sub (used for IS41 GSM Migration)
1 – own number ported out (used for G-Port and A-Port)
2 – foreign number ported to foreign network (used for G-Port and A-Port)
3 – prepaid 1 (used by PPSMS)
4 – prepaid 2 (used by PPSMS)
5 – migrated to GSM (used for IS41 GSM Migration)
6 – prepaid 3 (used by PPSMS)
7 – prepaid 4 (used by PPSMS)
8 – prepaid 5 (used by PPSMS)
9 – prepaid 6 (used by PPSMS)
10 – prepaid 7 (used by PPSMS)

11 – prepaid 8 (used by PPSMS)
12 – prepaid 9 (used by PPSMS)
13 – prepaid 10 (used by PPSMS)
14 – prepaid 11 (used by PPSMS)
15 – prepaid 12 (used by PPSMS)
16 – prepaid 13 (used by PPSMS)
17 – prepaid 14 (used by PPSMS)
18 – prepaid 15 (used by PPSMS)
19 – prepaid 16 (used by PPSMS)
20 – prepaid 17 (used by PPSMS)
21 – prepaid 18 (used by PPSMS)
22 – prepaid 19 (used by PPSMS)
23 – prepaid 20 (used by PPSMS)
24 – prepaid 21 (used by PPSMS)
25 – prepaid 22 (used by PPSMS)
26 – prepaid 23 (used by PPSMS)
27 – prepaid 24 (used by PPSMS)
28 – prepaid 25 (used by PPSMS)
29 – prepaid 26 (used by PPSMS)
30 – prepaid 27 (used by PPSMS)
31 – prepaid 28 (used by PPSMS)
32 – prepaid 29 (used by PPSMS)
33 – prepaid 30 (used by PPSMS)
34 – prepaid 31 (used by PPSMS)
35 – prepaid 32 (used by PPSMS)
36 – not identified to be ported

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.

vms
(Optional) Specifies which Voicemail Server the **DN**(s) are on and corresponds to the E.164 address of the voicemail server. The VMS must correspond to an existing VMS entity.

Values:
1 to 15 hexadecimal digits expressed using **ASCII** characters.
none – Sets the **vms** to not point to any network entity.

grn
(Optional) Specifies which Generic Routing Number the DN(s) are on and corresponds to the E.164 address used when the EAGLE “NE Query Only Option” has been turned on. The GRN must correspond to an existing GRN entity.

Values:
1 to 15 hexadecimal digits expressed using **ASCII** characters.
none – Sets the **grn** to not point to any network entity.

asd
(Optional) Additional Subscriber Data to be associated with the DN Block.

Values
1 to 10 hexadecimal digits expressed using ASCII characters. Leading zeros are significant.
none – Removes additional subscriber data from the DN.

st
(Optional) The subscriber type for created DNs.

Values
A decimal number in the range
0 - public
1 - private

nsdn
A Number Substitution DN (specified in international format).
Values:

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

cgbl
(Optional) IDP calling party blocklist.

Values
no - IDP calling party blocklist is disabled.

yes - IDP calling party blocklist is enabled

cdbl
(Optional) IDP called party blocklist.

Values
no - IDP called party blocklist is disabled.

yes - IDP called party blocklist is enabled.

lsblset
(Optional) A TIF Linkset based Blocklist parameter that is used to decide if the DN needs to be blocklisted or not.

Values
a number from **1** to **255**.

By default, no value is selected.

split
(Optional) Specifies the DN Block splitting ability. This parameter filters the request to retrieve only those DN Blocks that have the specified split value. This parameter must be specified in order to be used as a filter. If this parameter is not specified, filtering on the split value is not performed.

Values
no – DN Block splitting is disabled.

yes – DN Block splitting is enabled.

data
(Optional) Lets the requester specify the type of output data to be returned. See Retrieve Subscription Data Responses 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.

3. Specifying a combination of 2 Network Entity types (sp, rn, vms, grn) other than sp and rn is allowed and will result in instances matching both associations. Only specifying 1 NE will return all instances associated with the NE, even if instances are associated with other NEs.

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
<none/0/1/2/3/4/….35>],
[st <0/1>,] [nsdn XXXXX,] [sp XXXXX,] [rn XXXXX,] [vms XXXXX,] [grn XXXXX,]
```

```
[asd XXXXX,] [cgbl <no/yes>,] [cdbl <no/yes>,] [lsblset <1,2……255>,]
[split <no/yes>,] [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 requester specify the type of output data to be returned. See Retrieve Subscription Data Responses 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 requester specify the type of output data to be returned. See Retrieve Subscription Data Responses 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>])
```

Retrieve Subscription Data Responses

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 requester 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.

Note:

1. The `st` parameter will appear in the response only if its value differs from the system default (Public). See TIF Number Substitution Relationships for more details about number substitution.
2. The `cdbl` parameter will be listed in the `rtrv_sub()` command response only when it's value is `yes`.
3. The `cgbl` parameter will be listed in the `rtrv_sub()` command response only when it's value is `yes`.
4. In case of DN Block, the `split` parameter will be listed in the `rtrv_sub()` command response only when its value is `no`.

- 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
<none/0/1/2/3/4/….35>], [sp XXXXX,] [rn XXXXX] [vms XXXXX,] [grn
XXXXX,] [asd XXXXX,] [st <1>,] [nsdn XXXXX]), [cgbl <yes>,] [cdbl
<yes>,]
. . .,
( . . . ) ) )
```

```
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
<none/0/1/2/3/4/….35>], [sp XXXXX,] [rn XXXXX,] [vms XXXXX,] [grn
XXXXX,] [asd XXXXX,] [st <1>,] [nsdn XXXXX]), [cgbl <yes>,] [cdbl
<yes>,] [split <no>,]
. . .,
( . . . ) ) )
```

- 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. See PDBI Message Error Codes for the recommended actions to help resolve the error related return codes.

Table 3-10 Retrieve Subscription Data Response Return Codes

Return Code	Text	Description	Data Section Contents
0	**SUCCESS**	The request succeeded and this is the last (or only) response.	Depends on the request type, etc.
1012	**INVALID_VALUE**	One of the fields specified had an invalid value.	Offending field is returned in data section: **data (param <field label>)**
1016	**PARTIAL_SUCCESS**	The request has succeeded, but this is only one of many responses.	Depends on the request type, etc.
1013	**NOT_FOUND**	The requested **DN**, **DN** block, or **IMSI** was not found.	**NONE**
1009	**NO_ACTIVE_TXN**	There is no currently active transaction for this connection.	**NONE**
1021	**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**. Spare point codes, indicated by an optional **s-** prefix, are supported for Network Entities with a **pctype** of **intl** and **natl**.

Create Network Entity 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 do not use **RNs**. Used for **A-Port** and **IS41 GSM** Migration features.)
VMS – Voicemail Server. (Used for **V-Flex** customers.)
GRN – Generic Routing Number

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**, **VMS**s, and **GRN**s.)

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

Values:
For **pctype** of **intl** the format is zone-area-id [(s-)z-aaa-i].
s - Optional spare point code indicator
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 [(s-)nnnnn].
nnnnn= **1** – **16383**
For **pctype** of **ansi**, the format is network-cluster-member (nnn-ccc-mmm).
s - Optional spare point code indicator
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/VMS/GRN>, 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>])
```

Create Network Entity Response

The return codes listed in Table 3-11 indicate the result of the Create network entity request. See PDBI Message Error Codesfor the recommended actions to help resolve the error related return codes.

Table 3-11 Create Network Entity Response Return Codes

Return Code	Text	Description	Data Section Contents
0	SUCCESS	Everything worked.	NONE
1005	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>)`
1006	NO_WRITE_PERMISSION	**PDBI** client making request does not have write access permissions.	NONE
1009	NO_ACTIVE_TXN	There is no currently active transaction for this connection.	NONE
1011	WRITE_IN_READ_TXN	The command was sent on a **read only** transaction.	NONE

Table 3-11 (Cont.) Create Network Entity Response Return Codes

Return Code	Text	Description	Data Section Contents
1012	INVALID_VALUE	One of the fields specified had an invalid value.	Offending field is returned in data section: **data (param <field label>)**
1015	ITEM_EXISTS	The network entity already exists.	
1029	TXN_TOO_BIG	This request would cause current transaction to be larger than the limit.	NONE
1035	MAX_NE_LIMIT	Could not add new Network Entity to the database. Adding a new Network Entity would exceed the max allowed Network Entities.	NONE

Update Network Entity

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

Spare point codes, indicated by an optional **s-** prefix, are supported for Network Entities with a **pctype** of **intl** and **natl**.

Update Network Entity 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**. Used for **A-Port** and **IS41 GSM** Migration features.)
VMS – Voicemail Server. (Used for **V-Flex** customers.)
GRN – Generic Routing Number

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. (Only valid for **RNs**, **VMS**s, and **GRN**s.)

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 [(s-)z-aaa-i].
s - Optional spare point code indicator
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 [(s-)nnnnn].
s - Optional spare point code indicator
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`.
6. If the pctype is changed to none, all of the other parameters are cleared out, including the srfimsi.

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>])
```

Update Network Entity Response

The return codes listed in Table 3-12 indicate the result of the Update Network Entity request. See PDBI Message Error Codes for the recommended actions to help resolve the error related return codes.

Table 3-12 Update Network Entity Response Return Codes

Return Code	Text	Description	Data Section Contents
0	**SUCCESS**	Everything worked.	**NONE**
1012	**INVALID_VALUE**	One of the fields specified had an invalid value.	Offending field is returned in data section: **data (param <field label>)**
1011	**WRITE_IN_READ**_TXN	The command was sent on a **read only** transaction.	**NONE**
1013	**NOT_FOUND**	The requested **SP** was not found.	**NONE**
1029	**TXN_TOO_BIG**	This request would cause current transaction to be larger than the limit.	**NONE**
1009	**NO_ACTIVE_TXN**	There is no currently active transaction for this connection.	**NONE**
1017	**NO_UPDATES**	**Database** already contains the data in this request. No update necessary.	**NONE**
1006	**NO_WRITE**_PERMISSION	**PDBI** client making request does not have write access permissions.	**NONE**

Table 3-12 (Cont.) Update Network Entity Response Return Codes

Return Code	Text	Description	Data Section Contents
1005	**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.

Delete Network Entity 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 (only available with **G-Port**, **INP**, **A-Port**, and **IS41 GSM** Migration features)
VMS – Voicemail Server. (Used for **V-Flex** customers.)
GRN – Generic Routing Number

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/VMS/GRN>
[, timeout <0..3600>])
```

Delete Network Entity Response

The return codes listed in Table 3-13 indicates the result of the Delete Network Entity request. See PDBI Message Error Codes for the recommended actions to help resolve the error related return codes.

Table 3-13 Delete Network Entity Response Return Codes

Return Code	Text	Description	Data Section Contents
0	**SUCCESS**	Everything worked.	**NONE**
1012	**INVALID**_VALUE	One of the fields specified had an invalid value.	Offending field is returned in data section: **data (param <field label>)**
1011	**WRITE_IN**_READ_TXN	The command was sent on a **read only** transaction.	**NONE**
1022	**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 #####,]))`
1013	**NOT**_FOUND	The requested **SP** was not found.	**NONE**
1029	**TXN_TOO_BIG**	The request would cause the current transaction t be larger than the limit.	**NONE**
1009	**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.

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 (only available with **G-Port**, **INP**, **A-Port**, and **IS41 GSM** Migration features)
VMS – Voicemail Server. (Used for **V-Flex** customers.)
GRN – Generic Routing Number

Request syntax :

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

Retrieve the Information for a Range of NEs

Parameters :

id
Global title address for this network entity.
Values:
1 to 15 hexadecimal digits expressed using ASCII characters

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 (only available with **G-Port**, **INP**, **A-Port**, and **IS-41 GSM** Migration features)
VMS – Voicemail Server. (Used for **V-Flex** customers.)
GRN – Generic Routing Number

data
(Optional) Lets the requester 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/VMS/GRN>,]
[[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>])
```

Retrieve Network Entity Responses

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. See PDBI Message Error Codes for the recommended actions to help resolve the error related return codes.

Table 3-14 Retrieve Network Entity Response Return Codes

Return Code	Text	Description	Data Section Contents
0	**SUCCESS**	The request succeeded and this is the last (or only) response.	See data description above.
1005	**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>)`
1006	**NO_WRITE**_PERMISSION	**PDBI** client making request does not have write access permissions.	**NONE**
1009	**NO_ACTIVE_TXN**	There is no currently active transaction for this connection.	**NONE**
1012	**INVALID**_VALUE	One of the fields specified had an invalid value.	Offending field is returned in data section: **data (param <field label>)**
1013	**NOT_FOUND**	The requested `id` was not found.	**NONE**
1016	**PARTIAL**_SUCCESS	The request has succeeded, but this is only one response in many.	See data description above.

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 EAGLEs 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.

Switchover 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>])
```

Switchover Response

The return codes listed in Table 3-15 indicate the result of the Switchover request. See PDBI Message Error Codes for the recommended actions to help resolve the error related return codes.

Table 3-15 Switchover Response Return Codes

Return Code	Text	Description	Data Section Contents
0	**SUCCESS**	Switchover worked.	**NONE**
1012	**INVALID**_VALUE	One of the fields specified had an invalid value.	Offending field is returned in data section: **data (param <field label>)**
1007	**NO_MATE**	The **PDBA** could not negotiate a switchover with its mate. The switchover was denied.	**NONE**

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

Return Code	Text	Description	Data Section Contents
1005	**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>)`
1006	**NO_WRITE**_PERMISSION	The connection requesting the switchover does not have **WRITE** permission.	**NONE**
1010	**ACTIVE_TXN**	The command was issued from within a transaction.	**NONE**
1026	**MATE_BUSY**	The mate **PDBA** currently has a **write** transaction open.	**NONE**
1030	**DB_MAINT**_REQD	Replication is unable to get the two databases in synch. Call Oracle.	**NONE**
1036	REPLICATING	Asynchronous replication is still in progress.	The number of levels remaining to replicate. data (levels <num>)
1043	BAD_SWITCH_IN_ABP	The PDBA status is controlled by the PDBA Proxy feature and cannot be changed manually. Call Oracle for more information.	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.

PDBA Status Query Request

Parameters : None

Request syntax :

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

PDBA Status Query 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	Text	Description	Data Section Contents
0	**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.

Dump Connections 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>])
```

Dump Connections Responses

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. See PDBI Message Error Codes for the recommended actions to help resolve the error related return codes.

Table 3-17 Dump Connections Response Return Code

Return Code	Text	Description	Data Section Contents
0	**SUCCESS**	Connection list returned.	See above.
1012	**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**.

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**).

allow
(Optional) Select list type of Allow.

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

gray
(Optional) Select list type of Gray.

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

block
(Optional) Select list type of block.

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 (**allow**, **gray**, or **block**) 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.
4. Total command length must not exceed 247 characters.

> **Note:**
>
> Entering commands that exceed this length will result in the PDBI_CMD_LENGTH_EXCEEDED error (value 1045). In order to avoid this, remove unnecessary characters (including white space and parameters that are specified as the default value). If necessary, consider performing this provisioning in two steps by using an enter command followed by an update command.

Request syntax:

```
ent_eir( [iid XXXXX] imei XXXXX, [svn 0..99,] [allow yes/no,]
[grey yes/no,] [block 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.

allow
(Optional) Select list type of Allow.

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

gray
(Optional) Select list type of Gray.

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

block
(Optional) Select list type of block.

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 (**allow**, **gray**, or **block**) 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 syntax:

```
ent_eir([[iid XXXXX] bimei XXXXX, eimei XXXXX, [allow yes/no,]
[grey yes/no,] [allow 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**.
2. Total command length must not exceed 247 characters.

> **Note:**
>
> Entering commands that exceed this length will result in the PDBI_CMD_LENGTH_EXCEEDED error (value 1045). In order to avoid this, remove unnecessary characters (including white space and parameters that are specified as the default value). If necessary, consider performing this provisioning in two steps by using an enter command followed by an update command.

Request syntax:

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

Create IMEI Data Responses

The return codes in Table 3-18 may result from the Create **IMEI** request. See PDBI Message Error Codes for the recommended actions to help resolve the error related return codes.

Table 3-18 Create IMEI Response Return Codes

Return Code	Text	Description	Data Section Contents
0	SUCCESS	Everything worked.	NONE
1005	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>)`

Table 3-18 (Cont.) Create IMEI Response Return Codes

Return Code	Text	Description	Data Section Contents
1006	NO_WRITE_PERMISSION	The PDBI client making request does not have write access permissions.	NONE
1009	NO_ACTIVE_TXN	There was no currently active transaction for this connection.	NONE
1011	WRITE_IN_READ_TXN	The `create` command was sent on a **read** only transaction.	NONE
1012	INVALID_VALUE	One of the fields specified had an invalid value.	The offending field is returned in the data section: `data (param <field label>)`
1014	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)`
1017	NO_UPDATES	The database already contains data in request. No update was necessary.	NONE
1029	TXN_TOO_BIG	This request would cause current transaction to be larger than limit.	NONE
1037	CHECK_ DIGIT_ ERROR	The check digit provisioned did not match the calculated check digit.	imei, bimei, or eimei data (param imei xxxxx)
1039	IMEI_IMSI_LIMIT	Would cause more than 8 IMSIs to be provisioned on an IMEI.	NONE
1040	MAX_IMEI_LIMIT	Exceeded the maximum number of individual IMEIs.	NONE
1041	MAX_IMEI_BLK_LIMIT	Exceeded the maximum number of IMEI blocks.	NONE

Table 3-18 (Cont.) Create IMEI Response Return Codes

Return Code	Text	Description	Data Section Contents
1058	IMSI_FULL	Could not add new IMSI to the database. Adding new IMSI would exceed the supported SMxG card size. The other supported subscriptions are allowed if space is available on the corresponding SMxG card.	NONE
1059	IMEI_FULL	Could not add new IMEI to the database. Adding new IMEI would exceed the supported SMxG card size. The other supported subscriptions are allowed if space is available on the corresponding SMxG card.	NONE
1062	IMSI_ENT_NOT_ALLOWED	Could not add new IMSI to the database. The RTDB is either down or incoherent. Adding new IMSI requires creation of new IMSI table which may lead to over-allocation.	NONE
1063	IMEI_ENT_NOT_ALLOWED	Could not add new IMEI to the database. The RTDB is either down or incoherent. Adding new IMEI requires creation of new IMEI table which may lead to over-allocation.	NONE

Update IMEI Data

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

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).

allow
(Optional) Select list type of Allow.

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

gray
(Optional) Select list type of Gray.

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

block
(Optional) Select list type of block.

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

imsi (optional)

Rules

1. The resulting **imei** must have at least 1 list type (allow, gray or block) 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 **EPAP**s (via calculated algorithm). The check digit is not stored; it is only used to verify the **imei**.
3. Limit **imsi** to 400 per **imei**.

Request syntax:

```
upd_eir([iid XXXXX,] imei xxxxx, [svn 0..99,]
```

```
[allow yes/no,] [gray yes/no,] [block yes/no,]
imsi xxxxx, …, imsi xxxxx,] [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.

allow
(Optional) Select list type of Allow.

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

gray
(Optional) Select list type of Gray.

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

block
(Optional) Select list type of block.

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 (allow, gray or block) 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 syntax:

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

Update IMEI Data Responses

The return codes in Table 3-19 may result from the Update **IMEI** request. See PDBI Message Error Codes for the recommended actions to help resolve the error related return codes.

Table 3-19 Update IMEI Response Return Codes

Return Code	Text	Description	Data Section Contents
0	**SUCCESS**	Everything worked.	**NONE**
1012	**INVALID**_VALUE	One of the fields specified had an invalid value.	The offending field is returned in the data section: `data (param <field label>)`
1042	**NO_LIST**_FOR_IMEI	A minimum of 1 list type must be provisioned for the resulting **IMEI**.	**NONE**
1037	**CHECK_ DIGIT_ ERROR**	The check digit provisioned did not match the calculated check digit.	imei, bimei, or eimei data (param imei xxxxx)
1013	**NOT_FOUND**	The requested **IMEI** or **IMEI** block was not found.	**NONE**
1011	**WRITE_IN**_READ_TXN	The `update` command was sent on a **read** only transaction.	**NONE**
1029	**TXN_TOO_BIG**	This request would cause current transaction to be larger than limit.	**NONE**
1017	**NO_UPDATES**	The database already contains the data in the request. No update was necessary.	**NONE**
1009	**NO_ACTIVE**_TXN	There was no currently active transaction for this connection.	**NONE**
1006	**NO_WRITE**_PERMISSION	The **PDBI** client making request does not have write access permissions.	**NONE**

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

Return Code	Text	Description	Data Section Contents
1005	**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**.

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 syntax:

```
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 syntax:

```
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-FlexIMSI**.

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 syntax:

```
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-FlexIMSI**.

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 syntax:

```
dlt_eir([iid XXXXX,]imsi XXXXX)
```

Delete IMEI Data Responses

The return codes in Table 3-20 may result from the Delete **IMEI** request. See PDBI Message Error Codes for the recommended actions to help resolve the error related return codes.

Table 3-20 Update IMEI Response Return Codes

Return Code	Text	Description	Data Section Contents
0	**SUCCESS**	Everything worked.	**NONE**
1012	**INVALID**_VALUE	One of the fields specified had an invalid value.	The offending field is returned in the data section: `data (param <field label>)`
1037	**CHECK _DIGIT _ERROR**	The check digit provisioned did not match the calculated check digit.	imei, bimei, or eimei data (param imei xxxxx)
1013	**NOT_FOUND**	The requested **IMEI** or **IMEI** block was not found.	**NONE**
1038	**IMSI**_NOT_FOUND	**IMSI** not found on specified **IMEI**.	**NONE**
1011	**WRITE_IN**_READ_TXN	The `delete` command was sent on a **read** only transaction.	**NONE**
1029	**TXN_TOO_BIG**	This request would cause current transaction to be larger than limit.	**NONE**
1017	**NO_UPDATES**	The database already contains the data in the request. No update was necessary.	**NONE**
1009	**NO_ACTIVE**_TXN	There was no currently active transaction for this connection.	**NONE**
1006	**NO_WRITE**_PERMISSION	The **PDBI** client making request does not have write access permissions.	**NONE**
1005	**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.

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 syntax:

```
rtrv_eir([iid XXXXX,] imei XXXXX)
```

Retrieve IMEI Data: 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.

allow
(Optional) Filters request to retrieve the **IMEIs** found on the Allow 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**).

block
(Optional) Filters request to retrieve the **IMEIs** found on the block 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 syntax:

```
rtrv_eir([iid XXXXX,] bimei XXXXX, eimei XXXXX, [type
<block/single>,] [allow <yes/no>,] [gray <yes/no>,] [block
<yes/no>,] [imsi XXXXX] [data <all/count>] [num 0..40000000])
```

Retrieve IMEI Data Responses

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 ##, allow yes/no, gray yes/no, block
yes/no, [imsis (xxxxx, ....  xxxxx....)),
. . .
( . . . ) ) )
```

- Response syntax for an **IMEI** block query:

```
data (segment XXXXX,
meiblock (imeiblock (bimei xxxxx, eimei xxxxx, allow yes/no, gray
yes/no, block 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. See PDBI Message Error Codes for the recommended actions to help resolve the error related return codes.

Table 3-21 Retrieve IMEI Response Return Codes

Return Code	Text	Description	Data Section Contents
0	**SUCCESS**	Everything worked.	**NONE**
1011	**NOT_FOUND**	**IMEI** (block) not found.	**NONE**
1012	**INVALID_VALUE**	One of the fields specified had an invalid value.	Offending field is returned in data section: **data (param <field label>)**
1016	**PARTIAL_SUCCESS**	The request has succeeded, but this is only one of many responses.	Depends on the request type, etc.
1029	**NO_ACTIVE_TXN**	There is no currently active transaction for this connection.	**NONE**

Table 3-21 (Cont.) Retrieve IMEI Response Return Codes

Return Code	Text	Description	Data Section Contents
1037	**CHECK_ DIGIT _ERROR**	The check digit provisioned did not match the calculated check digit.	imei, bieme or eimei data (param imei xxxxx)
1038	**IMSI**_NOT_FOUND	The **IMSI** requested as part of the filter does not exist .	**NONE**

Request Service Module Card Report

This command is used to retrieve the Service Module card Report in a synchronous manner. This command is not required to be sent from inside a transaction.

Retrieve the Service Module Card Report Request

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 Service Module card exception list.

The **rtrv_dsmrpt** command defines the request message to retrieve the Service Module card 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 requester specify whether or not they want to the see list of **Service Module** cards that were not at the main database level mentioned in the report.

Values:
status (default) - Return the list of Service Module cards.
none – Do not return the list of Service Module cards.

Request syntax:

```
rtrv_dsmrpt([iid XXXXX,] [percent ###], [level #####], [data
<none/except>])
```

Request Service Module Card Report Response

The data section of a successful Service Module card 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 Service Module cards that satisfy the report have levels equal to or greater than this value.
Values: **0 – 4294967295**

percent
The percentage of known Service Module cards that meet or exceed the level specified.
Values: **0 – 100**

numdsms
The total number of known Service Module cards in the customer's network.
Values: **0 – ####**

dsms
List of Service Module cards that did not have a database level equal to or larger than main message's level. Each Service Module card contains the following information.

clli
Identifier for the Service Module card's EAGLE node. Values: String up to 11 characters long

cardloc
Location identifier for the Service Module card in the EAGLE node. Values: Four digit number

status
The database status of the Service Module 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 Service Module card (write failed to database)
inconsistent - Data mismatch between EPAP RTDB and Service Module card RTDB.
corrupt - Internal error on Service Module 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 Service Module card report request. See PDBI Message Error Codes for the recommended actions to help resolve the error related return codes.

Table 3-22 Retrieve Service Module Card Report Response Return Codes

Return Code	Text	Description	Data Section Contents
0	**SUCCESS**	Everything worked.	See above.
1012	**INVALID_VALUE**	One of the fields specified had an invalid value.	Offending field is returned in data section: **data (param <field label>)**
1016	**PARTIAL_SUCCESS**	The request has succeeded, but this is only one of many responses.	Depends on the request type, etc.

Retrieve Service Module Card 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 Service Module card exception list.

The **rtrv_dsmrpt** command defines the request message to retrieve the Service Module card 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 requester specify whether or not they want to the see list of **Service Module** cards that were not at the main database level mentioned in the report.

Values:
status (default) - Return the list of **Service Module** cards.
none – Do not return the list of **Service Module** cards.

Request syntax:

```
rtrv_dsmrpt([iid XXXXX,] [percent ###], [level #####], [data
<none/except>])
```

Retrieve Service Module Card Report: Response

The data section of a successful Service Module card 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 **Service Module** cards that satisfy the report have levels equal to or greater than this value.
Values: **0 – 4294967295**

percent
The percentage of known **Service Module** cards that meet or exceed the level specified.
Values: **0 – 100**

numdsms
The total number of known **Service Module** cards in the customer's network.
Values: **0 – ####**

dsms
List of Service Module cards that did not have a database level equal to or larger than main message's level. Each Service Module card contains the following information.

clli
Identifier for the **Service Module** card's **EAGLE** node. Values: String up to 11 characters long

cardloc
Location identifier for the **Service Module** card in the **EAGLE** node. Values: Four digit number

status
The database status of the **Service Module** 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 **Service Module** card (write failed to database)
inconsistent - Data mismatch between EPAP RTDB and Service Module card RTDB.

corrupt - Internal error on **Service Module** 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 Service Module card report request. See PDBI Message Error Codes for the recommended actions to help resolve the error related return codes.

Table 3-23 Retrieve Service Module Card Report Response Return Codes

Return Code	Text	Description	Data Section Contents
0	**SUCCESS**	Everything worked.	See above.
1012	**INVALID_VALUE**	One of the fields specified had an invalid value.	Offending field is returned in data section: **data (param <field label>)**
1016	**PARTIAL_SUCCESS**	The request has succeeded, but this is only one of many responses.	Depends on the request type, etc.

Retrieve a List of the Service Module Cards

This command is used to retrieve all or a subset of the **Service Module** cards known to the **PDBA**. It does not need to be sent from inside a transaction. This request is different that the Service Module card Report in that it does not attempt to determine any percent complete at a given level. It simply returns all of the **Service Module** cards that meet the filter criteria.

The **rtrv_dsmlist** command defines the request message to retrieve the **Service Module** card data.

Parameters :

clli
(Optional) Retrieve only the **Service Module** cards on the specified **EAGLE** node.

Values:
1 - 11 alphanumeric characters, hyphen, or underscore.

cardloc
(Optional) Retrieve only the **Service Module** cards that are in the specified card location.

Values:
Four digit number.

status
(Optional) Retrieve only the **Service Module** 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 **Service Module** card (i.e. write failed to database)
inconsistent - Data mismatch between EPAP RTDB and Service Module card RTDB.
corrupt - Internal error on **Service Module** card (checksum failure)

data
(Optional) Debugging option. Lets the requester specify whether they want to the see all available information the **PDBA** has about cards.

Values:
status – Return the standard list of data for each card (default)
all – Return additional debug information

Request syntax:

```
rtrv_dsmlist([iid XXXXX,] [clli XXXX], [cardloc ####], [status
<value list above>])
```

Retrieve a List of the Service Module cards 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 Service Module cards that did not have a database level equal to or larger than main message's level. Each Service Module card contains the following information.

clli
Identifier for the Service Module card's **EAGLE** node. Values: String up to 11 characters long

cardloc
Location identifier for the Service Module card in the EAGLE node. Values: four-digit number

status
The database status of the **Service Module** 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 **Service Module** card (write failed to database)
inconsistent - Data mismatch between EPAP RTDB and Service Module card RTDB.
corrupt - Internal error on **Service Module** 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 3-24 indicate the result of the Retrieve Service Module card list report request. See PDBI Message Error Codes for the recommended actions to help resolve the error related return codes.

Table 3-24 Retrieve Service Module Card List Response Return Codes

Return Code	Text	Description	Data Section Contents
0	**SUCCESS**	Everything worked.	See above.
1012	**INVALID_VALUE**	One of the fields specified had an invalid value.	Offending field is returned in data section: **data (param <field label>)**
1013	**NOT_FOUND**	There were no **Service Module** cards found. If one or more filters were specified, then there were no cards that matched the filter	**NONE**
1016	**PARTIAL_SUCCESS**	The request has succeeded, but this is only one of many responses.	Depends on the request type, etc.

4
PDBI Sample Sessions

This chapter contains example flow scenarios for the PDBI request/response messages.

Introduction

This chapter contains sample usages of the **PDBI**. The message exchanges are shown in Table 4-1 to Table 4-16. 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**.
←	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 9195551000, sp 9195551216)	A new **DN** has been created for an existing **IMSI**. The **DNs** associated with the **IMSI** are 9195551212, 9195551213, and 9195551216. The subscription remains on **SP** 9195550000.
←	rsp (iid 4, rc 0)	
→	ent_sub (iid 5, imsi 9195552000, sp 9195550000)	A **IMSI**-only subscription has been created. The **IMSI** is 9195552000. The subscription is on **SP** 9195550000.
←	rsp (iid 5, rc 0)	
→	ent_sub (iid 6, 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 6, rc 0)	
→	ent_sub (iid 7, 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 7, rc 0)	
→	end_txn (iid 8)	The **write** transaction has been ended. The updates have been written to the **PDB** and will be sent to the **EAGLE**.
←	rsp (iid 8, rc 0, data (dblevel 1))	
→	disconnect (iid 9)	The client is done and has disconnected.
←	rsp (iid 9 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 anone or more existing stand-alone **DN** to a new **SP**, and
- Move existing **DN** 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 **DN** 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 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)	

This example shows how the `upd_sub` command works with respect to the V-Flex feature. The following actions are shown in the example:

- Create new DN with a VMS
- Create new DN on SP/RN
- Move an existing DN to a new SP/RN
  - This illustrates how update works to replace SP/RN when SP/RN association already existed.
- Add a VMS association to an existing DN
  - This illustrates how update works to add a 2nd NE association when an association already existed.
- Remove an NE association to make room for a 2nd NE association

> **Note:**
>
> All VMSs, GRNs, & SPs in the following example are assumed to be provisioned prior to this example.

Table 4-4 Update Subscription Data Example 2

	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, dn 9195551600, vms 9195553000)	Creating a new standalone DN 9195551600 on VMS 9195553000.
←	rsp (iid 3, rc 0)	
→	ent_sub(iid 4, dn 9195551611, sp 9195551100)	One new standalone DN 9195551611 created on SP 9195551100 .
←	rsp (iid 4, rc 0)	
→	upd_sub(iid 5, dn 9195551611, sp 9195556666)	This command moves the specified standalone **DN** to **SP** 9195556666.
←	rsp (iid 5, rc 0)	
→	upd_sub(iid 6, dn 9195551600, sp 9195556666)	This command adds the second NE association to DN 9195551600. DN 9195551600 is now on VMS 9195553000 (V-Flex) and SP 9195551100 (G-Flex).
←	rsp (iid 6, rc 0)	
→	upd_sub(iid 7, dn 9195551600, grn 9196660000)	This command attempts to add a new NE association, but is rejected because it is not clear which NE (there are 2 associated with this DN) to be replaced.
←	rsp (iid 7, rc 1044)	
→	upd_sub(iid 8, dn 9195551600, vms none, grn 9196660000)	This command specifies that the GRN should replace VMS 9195553000. DN 9195551600 is now on GRN 9196660000 (NE Query Only option) and SP 9195551100 (G-Flex).
←	rsp (iid 8, rc 0)	
→	end_txn (iid 9)	The **write** transaction has ended. The updates will be written to the database.
←	rsp (iid 9, rc 0, data (dblevel 2))	
→	disconnect (iid 10)	The client is done and has disconnected.
←	rsp (iid 10`, rc 0)	

This example shows how the `upd_sub` command works with respect to G-Flex and V-Flex feature interoperability. The following actions are shown in the example:

- Create new multi-DN IMSI on SP
- Move DN from IMSI and make it a standalone DN on a VMS

- Move DN from IMSI and make it a standalone DN on a VMS and the same SP as before
- Move multi-DN IMSI to a new SP
- Move the standalone DNs that were on the original subscription onto the new SP
- Update DN with **ASD**
- Remove **ASD** from a DN

> **Note:**
>
> All VMSs, GRNs, & SPs in the following example are assumed to be provisioned prior to this example

Table 4-5 Update Subscription Data Example 3

	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 9195555000, dn 9195551310, dn 9195551311, dn 9195551312, dn 9195551313, dn 9195551314, sp 9195551100)	The IMSI created with 5 DNs. The subscription is on SP 9195551100.
←	rsp (iid 3, rc 0)	
→	upd_sub(iid 4, dn 9195551310, vms 9195551400)	The DN 9195551310 has been moved from IMSI 9195555000 to VMS 9195551400. It is no longer associated with SP 9195551100 and will not be routed by G-Flex.
←	rsp (iid 4, rc 0)	
→	upd_sub(iid 5, dn 9195551311, vms 9195551400, sp 9195551200)	The DN 9195551311 has been moved from IMSI 9195551000 and made into a standalone DN. The standalone DN 9195551311 is now on VMS 9195551400 and SP 9195551200. G-Flex routing for this DN will not be any different than before this update.
←	rsp (iid 5, rc 0)	
→	upd_sub(iid 6, imsi 9195555000, sp 9195556666)	This command moves the specified IMSI at its 3 remaining DNs to the SP 9195556666.
←	rsp (iid 6, rc 0)	
→	upd_sub(iid 7, dn 9195551310, sp 9195556666)	The DN was incorrectly moved off of the IMSI and no longer being routed by G-Flex. This update will correct the error in provisioning by putting the DN on the SP that the rest of the original subscription is on. DN 9195551310 was already on VMS and it is now on SP 9195556666. DN is now being routed by G-Flex like the rest of the original subscription.
←	rsp (iid 7, rc 0)	

Table 4-5 (Cont.) Update Subscription Data Example 3

	Message	Description
→	upd_sub(iid 8, dn 9195551311, sp 9195556666)	The standalone DN is no longer on this IMSI must be moved in a separate update command to the SP 9195556666. DN 9195551311 was already on the VMS and was just moved from the old SP to the new SP. DN 9195551311 is now on VMS 9195551400 and SP 9195556666.
←	rsp (iid 8, rc 0)	
→	upd_sub(iid 80, dn 9195551311, asd 001F247A).	This DN now has additional subscriber data, 001F247A.
←	rsp (iid 80, rc 0)	
→	upd_sub(iid 81, dn 9195551311, asd none)	This DN now has no additional subscriber data.
←	rsp (iid 81, rc 0)	
→	end_txn (iid 9)	The **write** transaction has ended. The updates will be written to the database.
←	rsp (iid 9, rc 0, data (dblevel 2))	
→	disconnect (iid 10)	The client is done and has disconnected.
←	rsp (iid 10, rc 0)	

Simple Queries

This example shows a `read` transaction that queries the data populated in a previous example.

Table 4-6 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

Table 4-6 (Cont.) Simple Queries Example

	Message	Description
←	rsp (iid 4, rc 0, data (imsis ((imsi 9195551000, sp 9195556666), (imsi 9195552000, sp 9195550000), (imsi 9195552001, sp 9195550000))))	of **DNs** will be omitted from the **IMSI** information.
→	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 9195551216, imsi 9195551000, sp 9195556666), (dn 9195551500, sp 9195555555), (dn 9195551501, imsi 9195552001, sp 9195550000), (dn 9195551502, sp 9195550000), (dn 9195551503, sp 9195550000))))	
→	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), (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 4-7 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, (dn 9195551216, imsi 9195551000, sp 9195556666), sp 9195550000))))	
←	rsp (iid 3, rc 0, data (dns ((segment 3, dns ((dn 9195551501, imsi 9195552001, sp 9195550000)) (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 4-8 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))	

Table 4-8 (Cont.) Abort Transaction Example

	Message	Description
→	abort_txn(iid 6)	The client decided to abort the transaction because the previous update failed. This will cause the two **DN**s 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 **DN**s 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 4-9 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 4-10 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 4-11 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 4-12 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, allow yes, gray yes, imsi 9199301234, imsi 9199302266)`	A single **IMEI** is created on both the allow and gray lists with 2 **IMSIs** associated with it. **SVN** is 0.
←	`rsp (iid 3, rc 0)`	
→	`upd_eir(iid 4, imei 12345678901234, allow no, block yes)`	The lists associated with the **IMEI** is now changed to be Block and Gray. The allow 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 4-13 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, block yes)`	An **IMEI** Block is created with the block list.

Table 4-13 (Cont.) IMEI Block Data Example

	Message	Description
←	`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 block and Gray. Note: the block 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**.
←	`rsp (iid 6, rc 0, data (dblevel 1))`	
→	`disconnect(iid 7)`	The client is done and has disconnected.
←	rsp (iid 7, rc 0)	

Asynchronous Service Module Card Report

This example shows a connection that has asked to receive the Service Module card Report every 10 seconds with a report complete percent of 90.

Table 4-14 Asynchronous Service Module Card 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 **Service Module card** 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 Service Module card 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 Service Module Card Report

This example shows a connection that uses the rtrv_dsmrpt request to receive the **Service Module card** Report.

Table 4-15 Synchronous Service Module Card 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 Service Module card Report.
←	rsp (rc 0, data (connectId 1, side active))	
→	rtrv_dsmrpt(percent 90)	The connection requests a Service Module card Report with a specific percent complete value of 90. The level 2912 is returned because only two Service Module 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 Service Module card 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.

Service Module Card List

This example shows a connection requesting to see the list of **Service Module** cards. There are only five cards in this example to make the responses smaller.

Table 4-16 Service Module Card 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()	
←	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))))	The connection requests to see the Service Module card list. No filter is used, so all Service Module cards should be returned.
→	`rtrv_dsmlist(clli atlanta)`	The connection requests the Service Module card list with a

Table 4-16 (Cont.) Service Module Card List Example

	Message	Description
←	`rsp (rc 0, data (dsms (dsm (clli atlanta, cardloc 1405, status coherent, level 2910), dsm (clli atlanta, cardloc 1407, status coherent, level 2918))))`	filter for **CLLIs** with the value of "atlanta".
→	`rtrv_dsmlist(clli atlanta, cardloc 1407)`	The connection requests the Service Module card 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 Service Module card 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 Service Module card 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

This chapter lists the PDBI error codes and text.

PDBI Message Error Codes

Table A-1 lists the error codes, associated text generated by the PDBI, and recommended actions.

Table A-1 PDBI Message Error Codes

Error Code	Text	Recommended Action
0	PDBI_SUCCESS =0	No action necessary. This indicates a successful response from the PDBA.
1001	PDBI_INTERNAL_ERROR = 1001	Contact Oracle.
1002	PDBI_NOT_CONNECTED	Establish an active connection before re-issuing this command.
1003	PDBI_ALREADY_CONNECTED	No action necessary. This is a redundant connection attempt.
1004	PDBI_PARSE_FAILED	Refer to the data section of the return message to determine the cause of the parse failure.
1005	PDBI_WRITE_UNAVAIL	Wait for write transaction to close before re-attempting this command.
1006	PDBI_NO_WRITE_PERMISSION	No action is necessary unless the IP address of this PDBI client should have WRITE access permissions. If that is the case, add WRITE permissions for the IP address of the PDBI client.
1007	PDBI_NO_MATE	Contact Oracle. This indicates that the Active PDB has lost contact with Standby.
1008	PDBI_STANDBY_SIDE	Connect to the Active PDBA for write operations.
1009	PDBI_NO_ACTIVE_TXN	Open a write transaction before retrying this command
1010	PDBI_ACTIVE_TXN	Wait for the transaction to complete before re-attempting the switchover.
1011	PDBI_WRITE_IN_READ_TXN	Reattempt this command after the read transaction is terminated and a write transaction is opened.
1012	PDBI_INVALID_VALUE	Re-enter the command using the correct value in the offending field returned in the data section.

Table A-1 (Cont.) PDBI Message Error Codes

Error Code	Text	Recommended Action
1013	PDBI_NOT_FOUND	This error is dependent on the contents of the database. No action is required, unless it is known that the item in question should be in the database. If this is the case, contact Oracle.
1014	PDBI_CONFLICT_FOUND	Using the *force* parameter can override this check and eliminate the return of this error code for certain commands.
1015	PDBI_ITEM_EXISTS	No action necessary. The PDB already has the requested entry in the database.
1016	PDBI_PARTIAL_SUCCESS	No action necessary. This error indicates that more messages are to follow. PDBI_SUCCESS will be returned to indicate the last message has been sent.
1017	PDBI_NO_UPDATES	No action necessary. The PDB already has the requested entry in the database.
1019	PDBI_BAD_ARGS	There is a problem with the syntax of the command. Refer to the data section of the response message to determine what problem was encountered.
1020	PDBI_TOO_MANY_CONNECTIONS	One of the PDBI connections needs to be terminated before this client can gain a connection.
1021	PDBI_NE_NOT_FOUND	No action required unless it is known that the NE in question is/should be in the database. If that is the case, please contact Oracle.
1022	PDBI_CONTAINS_SUBS	To delete the NE in question, all subscription data must be removed from the NE.
1023	PDBI_UNKNOWN_VERSION	Use the correct version on the connect message.
1025	PDBI_UNIMPLEMENTED	Contact Oracle.
1026	PDBI_MATE_BUSY	Retry command checking to see if mate releases write transaction.
1027	PDBI_IMSI_DN_LIMIT	EPAP only supports assigning a maximum of 8 DNs to a single IMSI
1028	PDBI_BAD_IMPORT_CMD	Wrong command is written in the import file. Remove unsupported commands and try again.
1029	PDBI_TXN_TOO_BIG	Reduce the transaction size.
1030	PDBI_DB_MAINT_REQD	Contact Oracle
1031	PDBI_DB_EXCEPTION	The client program should retry the transaction. If the error persists, contact Oracle.
1032	PDBI_MAX_IMSI_LIMIT	Contact Oracle. Database capacity for IMSIs has been reached.

Table A-1 (Cont.) PDBI Message Error Codes

Error Code	Text	Recommended Action
1033	PDBI_MAX_DN_LIMIT	Contact Oracle. Database capacity for DNs has been reached..
1034	PDBI_MAX_DNBLK_LIMIT	Contact Oracle. Database capacity for DN BLOCKs has been reached.
1035	PDBI_MAX_NE_LIMIT	Contact Oracle. Database capacity for NEs has been reached.
1036	PDBI_REPLICATING	Do not attempt the switchover until the replication is complete.
1037	PDBI_CHECK_DIGIT_ERROR	The check digit must provided by the Customers Client Software does not match the EPAPs (via calculated algorithm).
1038	PDBI_IMSI_NOT_FOUND	No action necessary.
1039	PDBI_IMEI_IMSI_LIMIT	Limit has been reached. EPAP only supports assigning a maximum of 8 IMSIs to a single IMEI.
1040	PDBI_MAX_IMEI_LIMIT	Contact Oracle. Database capacity for IMEIs has been reached.
1041	PDBI_MAX_IMEI_BLK_LIMIT	Contact Oracle. Database capacity for IMEI BLKs has been reached.
1042	PDBI_NO_LIST_FOR_IMEI	Provision at least one list type for IMEI object. Each IMEI object must have a list type attached.
1043	PDBI_BAD_SWITCH_IN_APB	The PDBA status is controlled by the PDBA Proxy feature and cannot be changed manually. Call Oracle for more information.
1044	PDBI_SUB_NE_LIMIT	Limit has been reached. DN or DN Block can have a maximum of 2 NE associations.
1045	PDBI_CMD_LENGTH_EXCEEDED	Command length exceeded 247 characters. Remove unnecessary characters (including white space and parameters that are specified as the default value) or perform provisioning in two steps by using an enter command followed by an update command.
1046	PDBI_MAX_ASD_LIMIT	Contact Oracle. Database capacity for ASD records has been reached.
1047	PDBI_DN_NOT_FOUND	The DN specified by nsdn does not exist.
1048	PDBI_MAX_ASSOCIATIONS	This request contains too many associations for a DN or DN Block. The error is raised when TIF Number Substitution is combined with other associations and distinct from SUB_NE_LIMIT.
1049	PDBI_UNRESOLVED_DEPENDENCY	The record already refers to another record through TIF Number Substitution. The NSDN must be updated to none before a new association can be formed.
1050	PDBI_INCOMPATIBLE_ST	The specified ST value conflicts with the ST value of NSDN.
1051	PDBI_INCOMPATIBLE_ROP	The GRN ROP value conflicts with the GMT value for the specified ASD.

Table A-1 (Cont.) PDBI Message Error Codes

Error Code	Text	Recommended Action
1052	PDBI_DNB_SAME_PROPERTIES	The new DN Block requested by the operator is a subset of an existing block with same properties.
1053	PDBI_MULTI_DNB_CONFLICT	The new DN Block could not be added to the database as multiple conflicting DN Blocks were found within given bdn-edn.
1054	PDBI_DNB_SPLIT_NOT_ALLOWED	The new DN Block tries to split an existing DN Block that is not allowed to be split or a new individual DN is entered within an existing DN Block that is not allowed to be split.
1055	PDBI_DNB_PARENT_PROPERTY_MISMATCH	Fragments of a master range have differing attributes. No automated resolution is possible to coalesce these records to satisfy a delete command.
1056	PDBI_DNB_DLT_NOT_ALLOWED	Fragments of a master range cannot be deleted while subranges are present.
1057	PDBI_TXN_TIMEOUT	Retry command checking to see if the write transaction does not timeout. Contact Oracle.
1058	PDBI_IMSI_FULL	Cannot add new IMSI to the database. Adding new IMSI would exceed the supported SMxG card size. Contact Oracle.
1059	PDBI_IMEI_FULL	Cannot add new IMEI to the database. Adding new IMEI would exceed the supported SMxG card size. Contact Oracle.
1060	PDBI_DN_FULL	Cannot add new DN to the database. Adding new DN would exceed the supported SMxG card size. Contact Oracle.
1061	PDBI_ASD_FULL	Cannot add new ASD to the database. Adding new ASD would exceed the supported SMxG card size. Contact Oracle.
1062	PDBI_IMSI_ENT_NOT_ALLOWED	The RTDB is either down or incoherent. Contact Oracle.
1063	PDBI_IMEI_ENT_NOT_ALLOWED	The RTDB is either down or incoherent. Contact Oracle.

B

TIF Number Substitution Relationships

This chapter includes information to support the TIF Number Substitution feature (TIF NS).

TIF Number Substitution Relationships

The TIF Number Substitution (TIF NS) feature enables subscriber DN records to associate with DNs and DN Blocks for TIF Number Substitution. All DNs and DN Blocks used for TIF NS shall be provisioned as public or private.

Two types of substitutions are supported:

1. Public has Private Number: A DN or DN Block substitutes to (points to) a DN that is Private. The originating DN or DN Block is Public.
2. Private has Public Number: A DN or DN Block substitutes to (points to) a DN that is Public. The originating DN or DN Block is Private.

Two types of cardinality are supported:

1. 1-way Substitution: A DN or DN Block points to a DN. An arbitrary number of DNs and DN Blocks can point to the same DN. The pointed to DN points to nobody.
2. Subscriber Pair: Two DNs may be linked in a pair. A DN in a TIF NS pair can only point to the DN pointing to it (other DN in pair). No other DNs or DN Blocks may point to either DN in such a pair.

This table shows the combination of associations that are supported and prohibited to subscriber records. The left-most column lists the subscriber record type, and the right columns list its supported and prohibited associations.

Table B-1 Supported and Prohibited Subscriber Associations

Type	Associations					
	NSDN	ASD	SP	RN	VMS	GRN
DN, DN Block	None	None	Any One			
DN, DN Block	None	None	Any Two Unique NE Types (Except SP + RN)			
DN, DN Block	Supported	None	Any One			
DN, DN Block	None	Supported	Any One			
DN, DN Block	Prohibited	Supported	Any Two Unique NE Types (Except SP + RN)			
DN, DN Block	Supported	Supported	Any One			

Table B-1 (Cont.) Supported and Prohibited Subscriber Associations

Type	Associations					
	NSDN	ASD	SP	RN	VMS	GRN
MSISDN	None	None	From IMSI	Prohibited		
MSISDN	Supported	None	From IMSI	Prohibited		
MSISDN	None	Supported	From IMSI	Prohibited		
MSISDN	Supported	Supported	From IMSI	Prohibited		

This is a list of rules for TIF NS:

1. An arbitrary number of DN and DN Blocks can number substitute to the same DN. This DN, called the NSDN, cannot substitute to anybody. The only exception is called a subscriber pair.
2. Subscriber pair: Two DNs that number substitute to each other. Nobody else can number substitute to either DN in this pair.
3. DN Blocks can number substitute to a DN. However, nobody can number substitute to a DN Block.
4. The default subscriber type is public. If subscriber type st has never been explicitly set on a record, the record is public.
5. Only DN and DN Blocks with the public subscriber type can be linked to a private DN through TIF NS. Likewise, only private DN and DN Blocks can linked to a public DN.
6. Subscriber type is set when the NSDN is defined and TIF NS is created. It cannot be changed while the TIF NS relationship persists.
7. There are two ways a DN or DN Block can be removed from TIF NS.
   - Directly, if it has an NSDN. It may be updated with ‘nsdn none’. This is to be performed on each DN in the case of a subscriber pair.
   - Indirectly, if it does not have an NSDN. All the DN and DN Blocks that substitute to it have been updated with ‘nsdn none’.
8. A DN cannot be deleted while other records refer to it by TIF NS.
9. A DN or DN Block with NSDN is limited to having one NE (sp, rn, vms, grn).
10. A DN or DN Block with two NE (sp, rn, vms, grn) and ASD, cannot have NSDN. They cannot substitute to another number.

C

TIF Linkset Based Blocklist Feature

This chapter includes information to support the TIF Linkset Based blocklist feature.

The TIF Linkset Based Blocklist functionality enables a misused user to still make legitimate calls in case it is blocked on a particular linkset. The blocklisted information for each number includes the blocklisted SetID. Therefore, all the messages arriving on EAGLE are screened with the following combination:

- The blocklisted SetID referred in incoming linkset
- The blocklisted SetID configured in RTDB

This functionality enables a misused user to still make legitimate calls in case it is blocked on a particular linkset.

The following table provides a summary of the Service Action used specifically for TIF Linkset Based Blocklist:

Table C-1 Summary of TIF Linkset Based Blocklist Service Actions

Service Action	Description	Function	Precedence
TIFLSBL	CgPN Service Action	Indicates that a Release should be generated if the blocklisted `SetID` (CgPNBLSet) configured with the incoming linkset matches with the blocklisted `SetID` configured against the DN in the RTDB	90

Configuration Options Used

No specific configuration options are used.

Action Performed

Indicates that a Release should be generated if:

- The blocklisted `SetID` (CgPNBLSet) configured with the incoming linkset matches with the blocklisted `SetID` configured against the DN in the RTDB.
- No Number Substitution CgPN Service Action is configured after TIFLSBL.

Terminating Action

Yes

D
DN Block Self Healing

This appendix includes information about the DN Block Self Healing feature.

DN Block Self Healing

The DN Block Self Healing feature allows the EPAP database to **self heal** when a command is executed to create a new DN Block that conflicts with one of the existing DN Blocks. In addition to this primary function, the feature also allows the defragmentation of the DN Blocks – split a DN Block into child DN Blocks upon addition of conflicting DN Blocks and back into their parent DN Block upon deletion of particular child DN Blocks.

This feature supports single DN Block conflict in the EPAP database.

1. The DN Block Self Healing feature enables EPAP users to add new DN Blocks that conflict with an existing DN Block in the EPAP database. The following changes occur when the self healing EPAP database encounters this conflict:
   - Pre-existing/conflicting DN Block is automatically deleted from the EPAP database.
   - New DN Blocks are created successfully with new attributes.
   - Old DN Block are split to create more DN Blocks for the range not covered by new DN block.
2. Upon deletion of the new DN Block, the DN Block Self Healing feature returns a range of numbers to the original DN Block, which was split as a result of addition of new DN Block. This splitting action is described in the previous item.

Error Codes for DN Block Self Healing

These error codes are associated with the DN Block Self Healing feature.

- **PDBI_MULTI_DNB_CONFLICT** - More than one conflicting DN Block was found in the EPAP dtabase.
- **PDBI_DNB_SAME_PROPERTIES** - A DN Block with the same properties exists in the database and is a superset of the requested DN Block. Splitting eligibility of a DN Block is not considered a property for this error.
- **PDBI_DNB_SPLIT_NOT_ALLOWED** - The user is attempting to split an existing DN Block that was specified at the time of its creation or update as ineligible for splitting.
- **PDBI_DNB_DLT_NOT_ALLOWED** - The user is attempting to delete a fragment of a master range while its subranges are present.
- **PDBI_DNB_PARENT_PROPERTY_MISMATCH** - The user is attempting to delete fragments of a master range which have differing attributes. Splitting eligibility of a DN Block is not considered a property for this error.

Rules for DN Block Self Healing

The list of rules for the DN Block Self Healing feature are as follows:

- When the new DN Block is a subset of an already existing DN Block with different properties, the old DN Block is split into either 2 or 3 new DN Blocks.
- An error is returned when the new DN Block has bdn and edn the same as an existing DN Block.
- An error is returned when the new DN Block has single DN address.
- An error is returned when the resulting DN Block has single DN address.
- An error is returned when more than one conflicting DN Block exists in the EPAP database.
- When the new DN Block is a subset of an existing block with same properties, the creation of new DN Block is not allowed.
- When the new DN Block conflicts with an existing block and is not its subset, the creation of new DN Block is not allowed.
- An error is returned when the user tries to insert a DN Block that conflicts with a DN Block that has itsa `split` option set to `no`.
- An error is returned when the user tries to insert a DN that conflicts with a DN Block that has its `split` option set to `no`.
- When a DN Block that was created as a result of splitting an existing DN Block is deleted, the complete DN Block that existed before the split is returned to the EPAP database.

Support for DN Block Self Healing

Table D-1 illustrates whether the DN Block Self Healing is supported (*Yes*) or prohibited (*No*) and the corresponding error that is returned.

Table D-1 Support for DN Block Self Healing

Step Performed	Supported?	Error Returned
New DN Block is added which is a subset of existing DN Block and with different properties	Yes	No error
New DN Block has bdn and edn same as existing block	No	PDBI_ITEM_EXISTS
New DN Block has single DN address	No	PDBI_BAD_ARGS
Resulting DN Block has single DN address	No	PDBI_BAD_ARGS
New DN Block is added which conflicts with more than one DN Block.	No	PDBI_MULTI_DNB_CONFLICT
New DN Block is a subset of an already existing DN Block with same properties	No	PDBI_DNB_SAME_PROPERTIES
New DN Block conflicts with the existing DN Block and is not its subset	No	PDBI_CONFLICT_FOUND
New DN Block conflicts with the existing DN Block with its split option set as false	No	PDBI_DNB_SPLIT_NOT_ALLOWED
New DN conflicts with the existing DN Block with its split option set as false	No	PDBI_DNB_SPLIT_NOT_ALLOWED