Copyright

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. 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 Xeon 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, Opteron, the AMD logo, and the AMD Opteron 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 on 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. 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.
Contents

About This Document ............................................................................................................. vii
Document Conventions ......................................................................................................... viii

Chapter 1
System Overview ....................................................................................................................... 1
Overview ..................................................................................................................................... 1
Introduction .............................................................................................................................. 1
OSA Charging and Account Management ................................................................................. 4
OSA Account Management ..................................................................................................... 5
OSA Charging Management .................................................................................................. 6
Amount-Based Transactions ................................................................................................. 7
Unit-Based Transactions ...................................................................................................... 8
Communications Framework ............................................................................................... 10
Sending Notification Short Messages .................................................................................. 11
Post-Paid Charging .............................................................................................................. 11

Chapter 2
Configuration ............................................................................................................................ 13
Overview ..................................................................................................................................... 13
Configuration Overview ......................................................................................................... 13
eserv.config Configuration ................................................................................................. 14
Configuring osaChamScs ....................................................................................................... 15
Configuring osaChamBe ......................................................................................................... 33
Client Configuration ............................................................................................................. 38

Chapter 3
System Alarms ........................................................................................................................ 47
Overview ..................................................................................................................................... 47
Alarm Topic Description ....................................................................................................... 47
OSA Client Interface Alarms ................................................................................................. 49

Chapter 4
Background Processes ............................................................................................................. 71
Overview ..................................................................................................................................... 71
osaServerNotificationHandlers ............................................................................................ 71
osaVWARSQuery Handlers .................................................................................................. 72
osaVWARSNotificationPlugins .......................................................................................... 72
libnotificationQM ................................................................................................................ 72
osaChamScs ........................................................................................................................... 73
runOsaChamScs .................................................................................................................... 73
ccssSMDispatcher ............................................................................................................... 73
signalOsaChamScs ............................................................................................................... 73
OSA CHAM SCS Gateway .................................................................................................... 74
Billing Engine EDRs ............................................................................................................. 74
Statistics ................................................................................................................................. 76
Alarms ....................................................................................................................................... 79
Chapter 10
Application Charging Manager Functionality .................................. 125
Overview .................................................................................. 125
SessionAborted() ..................................................................... 125
Application Charging Manager Parameters ..................................... 125

Chapter 11
Charging Session Functionality .................................................... 127
Overview .................................................................................. 127
CreditAmountReq() .................................................................... 127
CreditUnitReq() ......................................................................... 128
DebitAmountReq() ...................................................................... 128
DebitUnitReq() .......................................................................... 129
DirectCreditAmountReq() ........................................................... 129
DirectCreditUnitReq() ................................................................. 137
DirectDebitAmountReq() .............................................................. 143
DirectDebitUnitReq() ................................................................. 146
ExtendLifeTimeReq() ................................................................. 153
GetAmountLeft() ...................................................................... 153
GetLifeTimeLeft() .................................................................... 154
GetUnitLeft() .......................................................................... 154
RateReq() ................................................................................ 155
Release() ................................................................................ 157
ReserveAmountReq() ................................................................. 157
ReserveUnitReq() ..................................................................... 160
Charging Session Parameters ..................................................... 166

Chapter 12
Charging Session OSA Plus Functionality ..................................... 169
Overview .................................................................................. 169
CreditAmountWithAppChargingSessionReq() ................................ 169
CreditUnitWithAppChargingSessionReq() ..................................... 170
DebitAmountWithAppChargingSessionReq() ................................ 171
DebitUnitWithAppChargingSessionReq() ....................................... 172
DirectCreditAmountWithAppChargingSessionReq() ..................... 172
DirectCreditUnitWithAppChargingSessionReq() ......................... 177
DirectDebitAmountWithAppChargingSessionReq() ......................... 184
DirectDebitUnitWithAppChargingSessionReq() ............................. 187
ExtendLifeTimeWithAppChargingSessionReq() ......................... 194
GetAmountLeftWithAppChargingSession() ................................ 194
GetLifeTimeLeftWithAppChargingSession() ................................ 195
GetUnitLeftWithAppChargingSession() ...................................... 195
RateWithAppChargingSessionReq() ............................................ 196
ReserveAmountWithAppChargingSessionReq() ......................... 198
ReserveUnitWithAppChargingSessionReq() ................................. 201

Chapter 13
Application Charging Session Functionality ................................. 209
Overview .................................................................................. 209
CreditAmountErr () .................................................................. 209
CreditAmountRes() ................................................................... 210
CreditUnitErr () ........................................................................................................... 210
CreditUnitRes() .......................................................................................................... 210
DebitAmountErr () ....................................................................................................... 211
DebitAmountRes() ......................................................................................................... 211
DebitUnitErr () ............................................................................................................ 211
DebitUnitRes() ............................................................................................................. 212
DirectCreditAmountErr () .......................................................................................... 212
DirectCreditAmountRes() ............................................................................................. 212
DirectCreditUnitErr () ................................................................................................ 213
DirectCreditUnitRes() ................................................................................................. 213
DirectDebitAmountErr () ............................................................................................ 213
DirectDebitAmountRes() .............................................................................................. 214
DirectDebitUnitErr () .................................................................................................. 214
DirectDebitUnitRes() ................................................................................................... 214
ExtendLifeTimeErr () ................................................................................................... 215
ExtendLifeTimeRes() .................................................................................................... 215
RateErr () .................................................................................................................... 215
RateRes() ...................................................................................................................... 216
ReserveAmountErr () .................................................................................................. 216
ReserveAmountRes() ................................................................................................... 216
ReserveUnitErr () ......................................................................................................... 217
ReserveUnitRes() ......................................................................................................... 217
SessionEnded () ........................................................................................................... 217
Application Charging Session Parameters .................................................................. 218

Chapter 14

About Installation and Removal .................................................................................. 223

Overview ...................................................................................................................... 223
Installation and Removal Overview ............................................................................ 223
Execution - osaChamScs .............................................................................................. 224
Auxilliary Scripts (SignalOsaChamScs.sh) ................................................................. 224

NCC Glossary of Terms .............................................................................................. 227

Index ............................................................................................................................ 233
About This Document

Scope

The scope of this document includes all functionality a user must know in order to effectively operate the OSA CHAM Client Interface and BE Plugins. It does not include detailed design of the service.

Audience

This guide is written primarily for OSA Client Interface and BE Plugin administrators. However, the overview sections of the document are useful to anyone requiring an introduction.

Prerequisites

Although there are no prerequisites for using this guide, familiarity with the target platform would be an advantage.

A solid understanding of UNIX and a familiarity with IN concepts are an essential prerequisite for safely using the information contained in this technical guide. Attempting to install, remove, configure or otherwise alter the described system without the appropriate background skills, could cause damage to the system; including temporary or permanent incorrect operation, loss of service, and may render your system beyond recovery.

This manual describes system tasks that should only be carried out by suitably trained operators.

Related documents

The following documents are related to this document:

- OSA Charging and Account Management (CHAM) SCS - SRS Release 2 version 0.0.0.4
- BE Client Interface
- BE Server Plugins
- CCS User's Guide
- EDR Reference Guide
- SMS Technical Guide
- VWS Technical Guide
# Document Conventions

## Typographical Conventions

The following terms and typographical conventions are used in the Oracle Communications Network Charging and Control (NCC) documentation.

<table>
<thead>
<tr>
<th>Formatting convention</th>
<th>Type of information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Special Bold</strong></td>
<td>Items you must select, such as names of tabs. Names of database tables and fields.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Name of a document, chapter, topic or other publication. Emphasis within text.</td>
</tr>
<tr>
<td><strong>Button</strong></td>
<td>The name of a button to click or a key to press. <strong>Example:</strong> To close the window, either click <strong>Close</strong>, or press <strong>Esc</strong>.</td>
</tr>
<tr>
<td><strong>Key+Key</strong></td>
<td>Key combinations for which the user must press and hold down one key and then press another. <strong>Example:</strong> <strong>Ctrl+P</strong>, or <strong>Alt+F4</strong>.</td>
</tr>
<tr>
<td><strong>Monospace</strong></td>
<td>Examples of code or standard output.</td>
</tr>
<tr>
<td><strong>Monospace Bold</strong></td>
<td>Text that you must enter.</td>
</tr>
<tr>
<td><code>variable</code></td>
<td>Used to indicate variables or text that should be replaced.</td>
</tr>
<tr>
<td><code>menu option &gt; menu option &gt;</code></td>
<td>Used to indicate the cascading menu option to be selected, or the location path of a file. <strong>Example:</strong> <code>Operator Functions &gt; Report Functions</code> <strong>Example:</strong> <code>/IN/html/SMS/Helptext/</code></td>
</tr>
<tr>
<td><code>hypertext link</code></td>
<td>Used to indicate a hypertext link on an HTML page.</td>
</tr>
</tbody>
</table>

Specialized terms and acronyms are defined in the *Glossary* at the end of this guide.
## Overview

### Introduction

This chapter provides a high-level overview of the application. It explains the basic functionality of the system and lists the main components.

It is not intended to advise on any specific Oracle Communications Network Charging and Control (NCC) network or service implications of the product.

### In this chapter

This chapter contains the following topics.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>OSA Charging and Account Management</td>
<td>4</td>
</tr>
<tr>
<td>OSA Account Management</td>
<td>5</td>
</tr>
<tr>
<td>OSA Charging Management</td>
<td>6</td>
</tr>
<tr>
<td>Amount-Based Transactions</td>
<td>7</td>
</tr>
<tr>
<td>Unit-Based Transactions</td>
<td>8</td>
</tr>
<tr>
<td>Communications Framework</td>
<td>10</td>
</tr>
<tr>
<td>Sending Notification Short Messages</td>
<td>11</td>
</tr>
<tr>
<td>Post-Paid Charging</td>
<td>11</td>
</tr>
</tbody>
</table>

### Introduction

#### Purpose

The OSA Gateway Client Interface allows proprietary ASPs to utilise the billing, account management and functionality of the Oracle Voucher and Wallet Server (VWS) using the interface defined in the 3GPP OSA/Parlay specifications.

The OSA specifications define an architecture that enables application service providers to make use of network functionality through an open, standardized interface API. The purpose of this interface is to shield the complexity of the network, its protocols and specific implementations from the client application.

The OSA CHAM (Charging and Account Management) gateway implements a subset of the OSA APIs for the Account Management and Charging Service Capability Features.
System architecture

Here is an example of the system architecture of the OSA Gateway solution.

Process

This shows how an OSA action is processed.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The OSA Client communicates with the Service Capability Server, running the OSA CHAM Gateway. <code>osaChamScs</code> provides a connection point for the client.</td>
</tr>
<tr>
<td>2</td>
<td>During billing operations (reservations, debits and credits) <code>BeClient</code> communicates with the VWS through <code>beServer</code>. OSA CHAM plugins to <code>beServer</code> handle requests.</td>
</tr>
<tr>
<td>3</td>
<td>OSA CHAM plugins enable <code>beVWARS</code> to execute OSA CHAM actions.</td>
</tr>
<tr>
<td>4</td>
<td><code>osaChamScs</code> generates Call Data Records which are saved to a file and moved onto the SMS, or other CDR processing platform.</td>
</tr>
</tbody>
</table>

Components

The OSA Gateway Client Interface and the Billing Engines:

- may or may not be implemented on separate networks.
- will be run on separate machines.
The Client Application can be any third-party solution that connects to the OSA Gateway Client Interface.

**osaChamScs**

The OSA CHAM SCS Client Interface receives requests from the OSA Client Application and converts these to BE Client messages. Communication between the Billing Engine and the client interface is done using the OSA CHAM SCS BE plugins. The client interface implements methods that are supported by the OSA Gateway. Only a subset of the complete specification is supported by the client interface.

**osaChamBe**

The OSA CHAM SCS BE plugins allow communication between the OSA CHAM SCS Client Interface and the Billing Engine. These are the plugins supplied at this time:

- Charging plugin which will allow reservations and queries to the VWARS component of the billing engine, and
- Account Management plugin which will allow the query of an account based on MSISDN, wallet-type name and balance-type name, balance type cascade.
- Notification plugin which allows notification events to be set on subscriber accounts i.e. for charging, recharging or low balance notifications. This will result in the subscriber receiving notifications whenever one of the events is triggered.

**osaChamSms**

The OSA CHAM SMS package adds entries in the SMS replication table ‘REP_CNF_GROUP’ on the SMF. This action will cause the required tables to be replicated from the SMF to the target platform where the OSA Gateway resides. If the OSA Gateway is not being used in conjunction with CCS, or the customer has not purchased CCS, then this optional package need not be installed.

The following tables also need to be replicated to each of the billing engines:

1. OSA_NOTIFICATION
2. OSA_TAGGED_BALANCE
Relationship diagram

This diagram shows the relationships between the OSA CHAM Gateway and other Oracle machines and applications.

- ACS and CCS provide call and data services.
- VWS executes tariffing and business rule requests.
- SMS provides platform support systems.
- OSA CHAM SCS on the SCS provides the gateway to the client application.

OSA Charging and Account Management

Introduction

OSA CHAM supports the following Account and Charging actions:

- OSA Account Management
  - Query an account
  - Account transaction history
  - Notifications
- OSA Charging Management
  - Amount-based reservations/debits/credits
  - Amount-based direct debits/credits
  - Unit-based reservations/debits/credits (Unit seconds and unit named events)
  - Unit-based direct debits/credits (Unit seconds and unit named events)
- Support operations to query outstanding reservation amounts, reservation lifetimes and rating tariffs

**OSA Account Management**

**Introduction**

Account Management consists of communication between the Client Account Manager and the Gateway Account Manager. In most cases, the message flows originate with the client.

OSA Account Management functions are:
- Query an account
- Account transaction history
- Notifications

**Message flow diagram**

Here is an example message flow for Account Management.

1. The `setCallback` method must be the first in the sequence of method calls.
2. This is followed by an Account management method call (for example: `queryBalanceReq`).

**Querying an account**

The Query Balance Request finds:
- the value of a specific Wallet Balance and additional Wallet information.
- the value of a specific Wallet Balance cascade and additional Wallet information.

**Account transaction history**

You can obtain the transaction history for a single Wallet and Balance Type over a given time period.

**Notification triggers**

You can create, change and disable cause notification triggers on Accounts/Wallets on the Billing Engine so the client receives report notifications when a charging event fires the trigger.
OSA Charging Management

Introduction

Charging Management consists of communication between the Client Charging Manager and the Gateway Charging Manager. In most cases, the message flows originate with the client.

Message flow diagram

Here is an example message flow for Charging Management.

1. The setCallback method must be the first in the sequence of method calls.
2. This is followed by a createChargingSession method call. This returns a new charging session to the client.

Charging sessions

Charging sessions will be closed as a result of one of the following events:

- It is released by the client application using the release method.
- A session times out on the OSA Gateway and the BE and the sessionEnded method is invoked on the client application.
- An exceptional event causes the OSA Gateway to invoke the sessionAborted method on the client application.

A Charging session may consist of a sequential series of reservation sessions interspersed with direct debits and credits.

Amount and unit-based methods cannot be mixed within a single reservation.

Charging session management

Charging Session Management consists of communication between the Client Charging Session and the Gateway Charging Session. In most cases, the message flows originate with the client.

The createChargingSession method must have been previously called to create the current charging session.

The session ID uniquely identifies the charging session.

The request number identifies the sequence of the request within the charging session.
Amount-Based Transactions

Charging management

For amount-based charging management, you can:

- Create multiple charging sessions to allow multiple amount reservations to be opened in parallel.
- Use the reserve amount request to make an amount-based reservation on a Wallet/Balance Type or to increase the reserved amount on an existing reservation.
- Use the debit/credit amount request to debit or credit against a current reservation.
- Use the reserve amount request to extend the reservation lifetime and/or increase the reserved amount.
- Use the debit/credit amount request to confirm debited funds and charge a Wallet/Balance Type.
- Use the direct debit/credit amount request to make a direct/credit charge against the Wallet/Balance Types.
- Use the direct debit/credit amount request to change the Wallet expiry date or extend the Balance expiry date.
- Use balance type cascade to ignore the balance type. If the balance type cascade is specified then the balance type will be ignored.

Charging party fields

Charging management uses OSA-specific functionality to perform amount-based reservations and direct debits.

Charging parameters are common for all methods, however due to differences between method types, certain parameters will only be used in some method invocations.

The lifetime field is used in the initial reservation request to set the lifetime of the reservation. In subsequent requests the lifetime value is added to the remaining lifetime of the reservation.

Wallet Type and Balance Type can be used to specify a particular Wallet/Balance Type to be charged. If not specified, the defaults for this Account will be used.

Wallet and Balance expiry dates can be changed by using Balance validity and Wallet validity fields in the direct debit/credit amount requests.

It is also possible to extend the lifetime of the current reservation on the OSA Gateway and the BE, using an extend lifetime request operation.

Balance type cascade can be used to ignore the balance type. If the balance type cascade is specified then the balance type will be ignored.

OSA discount

OSA Discount is a percentage Discount applied to the amount before charging is applied.

Valid values are 0 – 10000 where these are in 1/100th of a percent. Therefore, 500 would equate to a 5% discount.

The Discount will be applied to the charge using the following rules:

- For amount-based charges only an OSA Discount may be applied to the charge.
- There are 3 ways the OSA Discount is applied (depending on the BE configuration):
  - Override - ignores OSA discount
  - Cumulative - applies OSA discount
  - Compound - applies OSA discount
Example EDRs

Here are some example amount-based EDRs.

- Example amount-based confirmation OSA gateway EDR:
  
  ```
  21|Session=New ChargingSession|Merchant=Merchant Id/1|
  Account=91223289900/Personal/General Cash|ReservedAmount=50|
  ConfirmedAmount=20|Reservation=20031106133119|
  Confirmed=20031106133147|RES=50;LIF=180;DEB=20|
  ApplicationDescription=ReserveAmountReq:DebitAmountReq|
  UnitType=amount|DiscountOverride=0|N
  ```

- For amount-based direct charges an EDR will always be written on the SCS. No Billing Engine Rate request is required as the amount is already known. Example direct debit/credit amount-based OSA Gateway EDR:
  
  ```
  23|Session=New ChargingSession|Merchant=Merchant Id/1|
  Account=91223289900/Personal/General Cash|Amount=-10|
  Postpaid|Time=20031106133230|ApplicationDescription=DirectDebitAmountReq|
  UnitType=amount|DiscountOverride=0|N
  ```

- Example amount-based reservation BE EDR:
  
  ```
  BILLING_ENGINE_ID=1|SCP_ID=198248872|SEQUENCE_NUMBER=4494|CDR_TYPE=21|
  RECORD_DATE=20031119140101|ACCT_ID=2146|ACCT_REF_ID=2123|WALLET_TYPE=23|
  BALANCE_TYPE=49|BALANCES=9270|COSTS=10|
  APPLICATION_DESC=ReserveAmountReq:DebitAmountReq:ReserveAmountReq(subsequent)
  ```

- Example amount-based direct debit/credit BE EDR:
  
  ```
  BILLING_ENGINE_ID=1|SCP_ID=198248872|SEQUENCE_NUMBER=4495|CDR_TYPE=23|RECORD_DATE=20031119140145|
  ACCT_ID=2146|ACCT_REF_ID=2123|WALLET_TYPE=23|BALANCE_TYPE=49|BALANCES=9260|COSTS=-10|
  APPLICATION_DESC=DirectDebitAmountReq|OLD_BALANCE_EXPIRIES=0|NEW_BALANCE_EXPIRIES=0|
  BONUS_TYPE=CUSTOM|REFERENCE=OSA Bonus
  ```

For more information, see OSA CHAM EDRs (see “OSA CHAM SCS Gateway” on page 74) and Billing Engine EDRs (on page 74).

Unit-Based Transactions

Charging management

For unit-based charging management, you can:

- Create multiple charging sessions to allow multiple unit reservations to be opened in parallel.
- Use the reserve unit request to make an unit-based reservation on a Wallet/ Balance Cascade or to increase the reserved unit on an existing reservation.
- Use the debit/credit unit request to debit or credit against a current reservation.
- Use the reserve unit request to extend the reservation lifetime and/ or increase the reserved unit.
- Use the debit/credit unit request to confirm debited funds and charge a Wallet/ Balance Cascade.
- Use the direct debit/credit unit request to make a direct/credit charge against the Wallet/ Balance Cascade.

Charging party fields

Charging management uses existing CCS functionality to perform unit-based reservations and direct debits.
Wallet Type can be used to specify a particular Wallet type to be charged. If not specified the Account default Wallet type will be used.

All unit-based reservations use the Balance Cascade defined through CCS.

Wallet and Balance expiry dates can be changed by the unit-based methods

An extend lifetime request (ExtendLifeTimeReq() method) can be used to extend the lifetime of a current reservation on the OSA Gateway and the BE.

### Lifetime

The lifetime field in the initial reservation request determines the lifetime of the reservation on the OSA Gateway and the Billing Engine. If not specified, the default lifetime is 30 seconds.

For unit seconds-based reservations, the lifetime also determines the number of unit seconds reserved (unless this value is scaled down to match the available Wallet Balance).

- The reservation lifetime on the OSA Gateway will be set to the number of units (seconds) which the BE returns in the acknowledgement.
- If the lifetime requested is less than the BE minimum call length, the acknowledgement will return zero. This will close the session immediately.
- In subsequent reservations, the lifetime will be extended by the lifetime value from the initial reservation.

For unit-named event based reservations, the reservation lifetime on the OSA Gateway will be set to the number of seconds the BE returns in the acknowledgement. The lifetime field in subsequent reservations is ignored, the reservation lifetime cannot be extended.

### OSA Discount

OSA Discount is a percentage discount applied to the final cost before charging is applied.

Valid values are 0 – 10000 where these are in 1/100th of a percent. Therefore, 500 would equate to a 5% discount.

The Discount will be applied to the charge using the following rules:

- For unit named-events only an OSA Discount may be applied to the charge.
- For unit seconds a CCS Tariff Discount may also be applied to the charge.
- There are 3 ways the OSA Discount is applied (depending on the BE configuration):
  - Override - ignores OSA Discount
  - Cumulative - combines CCS Discount and OSA Discount
  - Compound - applies CCS Discount, then applies OSA Discount

### Example EDRs

Here are some example unit-based BE EDRs.

- Example unit-seconds based reservation BE EDR:
  ```
  BILLING_ENGINE_ID=1|SCP_ID=198248872|SEQUENCE_NUMBER=4572|CDR_TYPE=24|RECORD_DATE=20031119155328|ACCT_ID=2146|ACCT_REF_ID=2123|CLI=01473|ACS_CUST_ID=22|BALANCE_TYPES=49|BALANCES=60|COSTS=83|ACCOUNT_TYPE=22|CASCADE_ID=21|RATES=20,20,20,20,10,10,10,10|LENGTHS=60,60,60,60,60,60,60,60,-1|DISCOUNTS=250000,250000,250000,250000,250000,250000,250000,250000,500000|MAX_CHARGE=-1|DURATION=360|TN=01394|TCS=20031119154923|TCE=20031119155523|CS=S|DISCOUNT_TYPE=CUMULATIVE|APPLICATION_DESC=ReserveUnit(Seconds)Req:DebitUnitReq
  ```
Example unit-named event based reservation BE EDR:

```
BILLING_ENGINE_ID=1|SCP_ID=198248872|SEQUENCE_NUMBER=4487|RECORD_DATE=20031119135502|ACCT_ID=2146|ACCT_REF_ID=2123|APPLICATION_DESC=ReserveUnit(NamedEvents)Req:DebitUnitReq:CreditUnitReq|ACS_CUST_ID=22|CS=S|TCS=20031119135121|BALANCE_TYPES=49|BALANCES=8985|COSTS=700|ACCOUNT_TYPE=22|EVENT_CLASS=OSA2BE|EVENT_NAME=event1|EVENT_COST=100|EVENT_COUNT=7|DISCOUNT=0|CASCADE=21
```

Example unit-seconds based direct credit/debit BE EDR:

```
BILLING_ENGINE_ID=1|SCP_ID=198248872|SEQUENCE_NUMBER=4490|RECORD_DATE=20031119135814|ACCT_ID=2146|ACCT_REF_ID=2123|CLI=01473|ACS_CUST_ID=22|BALANCE_TYPES=49|BALANCES=8285|COSTS=15|ACCOUNT_TYPE=22|CASCADE_ID=21|RATES=20,20,20,20,20,10,10,10,10,10|LENGTHS=60,60,60,60,60,60,60,60,60,60,-1|DISCOUNTS=250000,250000,250000,250000,250000,250000,250000,250000,500000,500000|MAX_CHARGE=100000|DURATION=10|TN=01394|TCS=20031119135432|TCE=20031119135442|CS=S|DISCOUNT_TYPE=CUMULATIVE|APPLICATION_DESC=DirectCreditUnit(Seconds)Req
```

Example unit-named event based direct credit/debit BE EDR:

```
BILLING_ENGINE_ID=1|SCP_ID=198248872|SEQUENCE_NUMBER=4492|RECORD_DATE=20031119135935|ACCT_ID=2146|ACCT_REF_ID=2123|APPLICATION_DESC=DirectDebitUnit(NamedEvents)Req|ACS_CUST_ID=22|CS=S|TCS=20031119135554|BALANCE_TYPES=49|BALANCES=8270|COSTS=-1000|ACCOUNT_TYPE=22|EVENT_CLASS=OSA2BE|EVENT_NAME=event1|EVENT_COST=100|EVENT_COUNT=10|DISCOUNT=0|CASCADE=0
```

For more information, see Billing Engine EDRs (on page 74).

Post-paid charging

For unit-second based direct charges, the MSISDN will be checked against ranges defined in the configuration file on the SCS. This configuration parameter will provide the post-paid subscriber number that will be used to determine the Rating information on the Billing Engine.

For unit Named Event-based direct charges, only the Named Event is required to determine the Rating information on the BE.

Using the Rating information from the BE the OSA Gateway will calculate the cost for the requested units and then write a CDR on the SCS.

```
23|Session=New ChargingSession|Merchant=MerchantId/1|Account=912233289900/Personal|Amount=-10|Postpaid|Time=20031106133230|ApplicationDescription=DirectDebitUnitReq|UnitType=tariff.91.44|DiscountOverride=0|N
```

Communications Framework

Introduction

The client application and the OSA Gateway communicate using the Common Object Reference Broker Architecture (CORBA).

On start-up, the:

- OSA gateway writes the Interoperability Object References (IORs) to a known location which is specified by configuration parameters.
  The IORs are composed of the IP Address and port number that the OSA gateway is listening on for a connection attempt from the client application.
Sending Notification Short Messages

Introduction

Some client actions may have a flag set to send a notification SMS to the subscriber.

Diagram

OSA CHAM Gateway supports notification SMS functionality as shown in this diagram.

Process

Here is the notification SMS process.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The OSA Client sends a request for a subscriber notification to osaChamScs on the SCS.</td>
</tr>
<tr>
<td>2</td>
<td>osaChamScs writes the required notification details to the fifo file for ccssSMDispatcher.</td>
</tr>
<tr>
<td>3</td>
<td>ccssSMDispatcher creates a notification request for ccssSSMMaster on the SLC.</td>
</tr>
<tr>
<td>4</td>
<td>ccssSSMMaster creates the notification SMS and forwards it to NotificationIF.</td>
</tr>
<tr>
<td>5</td>
<td>NotificationIF forwards the SMS to SMSC IF, which sends it to an SMSC for delivery to the subscriber.</td>
</tr>
</tbody>
</table>

Post-Paid Charging

Introduction

Post Paid charging:

- only applies to direct debit and direct credit transactions.
Chapter 1

- processes the Call Data Records (CDRs) to calculate the user's charges after the event.
- will only occur if the MSISDN does not exist in the CCS database; the assumption is made that this is not a pre-paid account.

This table shows how a cost for the charge is determined.

<table>
<thead>
<tr>
<th>Direct Charge Type</th>
<th>a rating request is made using...</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit-seconds based</td>
<td>a configured MSISDN</td>
</tr>
<tr>
<td>unit-named event based</td>
<td>the named event</td>
</tr>
</tbody>
</table>
Chapter 2

Configuration

Overview

Introduction

This chapter explains how to configure the Oracle Communications Network Charging and Control (NCC) application.

In this chapter

This chapter contains the following topics.

- Configuration Overview
- eserv.config Configuration
- Configuring osaChamScs
- Configuring osaChamBe
- Client Configuration

Configuration Overview

Introduction

This topic provides a high level overview of how the OSA CHAM Client interface and BE plugins are configured.

There may be other configuration options, added to the configuration files that are not explained in this chapter. These configuration options are required by the application and should not be changed by the user.

Configuration process overview

This table describes the process of configuring OSA CHAM for the first time.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The eserv.config file must be configured for the SMSs. The example file should be copied into the main eserv.config, and any required details configured. For more information, see eserv.config Configuration (on page 14).</td>
</tr>
<tr>
<td>2</td>
<td>The eserv.config.scs file must be configured for the SLCs. Any required details may be configured.</td>
</tr>
<tr>
<td>3</td>
<td>The eserv.config_be file must be configured for the BEs. The example file should be copied into the eserv.config_be file and any required details configured.</td>
</tr>
</tbody>
</table>

Introduction

The BE plugins require certain parameters to be read from a configuration file. A sample of these parameters has been provided below, as they should appear in the 'eserv.config' file. During the package installation process it will be necessary to add the osaCham section, with sub-sections:
Chapter 2

- NotificationDatabase
- NotificationQueueManager
- NotificationEvents, and the
- NotificationServerHandler

The osaCham section is parsed by the OSA bOracleer plugins and beVWARS handlers.

You must also merge OSA-specific entries to the following sections:

- BE
- beVWARS
- CCS

This is the same configuration file used by other modules on the target platform. The plugins will not read 'example.Oracle.conf.be' file; it has been provided as an example only.

**Configuration components**

OSA CHAM is configured by the following components:

<table>
<thead>
<tr>
<th>Component</th>
<th>Locations</th>
<th>Description</th>
<th>Further Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>eserv.config</td>
<td>all SLCs</td>
<td>OSA CHAM Client interface is configured by the osaCham section of eserv.config.scs.</td>
<td>eserv.config Configuration (on page 14).</td>
</tr>
<tr>
<td>eserv.config_be</td>
<td>all VWSs</td>
<td>OSA CHAM BE plugins are configured by the osaCham section of eserv.config.be</td>
<td>eserv.config Configuration (on page 14).</td>
</tr>
<tr>
<td>SLEE.cfg</td>
<td>all SLCs and BEs</td>
<td>SLEE.cfg sets up SLEE interfaces and applications.</td>
<td>SLEE.cfg.</td>
</tr>
<tr>
<td>osaChamClient.conf</td>
<td>host machine</td>
<td>Configuration file includes defaults used within the client and location for OSA CHAM gateway CORBA location files.</td>
<td>osaChamClient.conf parameters (on page 39)</td>
</tr>
</tbody>
</table>

**eserv.config Configuration**

**Introduction**

The *eserv.config* file is a shared configuration file, from which many NCC applications read their configuration. Each NCC machine (SMS, SLC, and VWS) has its own version of this configuration file, containing configuration relevant to that machine. The *eserv.config* file contains different sections; each application reads the sections of the file that contains data relevant to it.

The *eserv.config* file is located in the `/IN/service_packages/` directory.

The *eserv.config* file format uses hierarchical groupings, and most applications make use of this to divide up the options into logical groupings.

**eserv.config files delivered**

Most applications come with an example *eserv.config* configuration in a file called *eserv.config.example* in the root of the application directory, for example, `/IN/service_packages/eserv.config.example`. 
Editing the file

Open the configuration file on your system using a standard text editor. Do not use text editors, such as Microsoft Word, that attach control characters. These can be, for example, Microsoft DOS or Windows line termination characters (for example: ^M), which are not visible to the user, at the end of each row. This will cause file errors when the application tries to read the configuration file.

Always keep a backup of your file before making any changes to it. This will ensure you have a working copy to which you can return.

Loading eserv.config changes

If you change the configuration file, then you must restart the appropriate parts of the service to enable the new options to take effect.

Configuring osaChamScs

Introduction

The client interface requires the eserv.config.scs configuration file to retrieve certain parameters. This file is located in the following directory:

```
/IN/service_packages
```

If the file cannot be found, or if read permissions are not set, then a suitable ‘CRITICAL ERROR’ is displayed to the standard output and the client interface exits.

Example configuration file

Here is the example.eserv.config.scs file with comments removed.

```python
osaCham = {
    AllowDirectPostPaid = false
    AllowInvalidReleaseSequenceNumber = true
    MaximumAmountValue = 1000
    MaximumSecondsValue = 3600
    MaximumEventsValue = 100
    ORBDebugLevel = 0
    ORBIIOPort = 1025
    ORBIIOPAddr = "xxx.xxx.xxx.xxx"
    ORBIIOPSetKeepalive = true
    OracleUsernamePassword = "scp/scp"
    requestTimeout = 10000
    CDRdirectory = "/IN/service_packages/OSA/cdr"
    CDRMaxAge = 60
    CDRMaxSize = 100
    fifoFileName = "/tmp/ccsSSMRequest.fifo"
    SMSSLanguage = "unknown"
    AlwaysSendNotification = false
    AlwaysUseAbsoluteCostValueForNotifications = false
    OverrideInitialTariffCdrType = true
    OverrideInitialEventCdrType = true
    ipAccountManagerIORFile = "/tmp/ipAccountManagerIOR.txt"
    ipChargingManagerIORFile = "/tmp/ipChargingManagerIOR.txt"
    clientName = "exampleConfigBeClient"
    connectionRetryTime = 5
    heartbeatPeriod = 5000000

    plugins = [
        
```
Chapter 2

```java
config="voucherRechargeOptions",
library="libccsClientPlugins.so",
function="makeVoucherRechargePlugin"
}
{
 config="voucherTypeRechargeOptions",
library="libccsClientPlugins.so",
function="makeVoucherTypeRechargePlugin"
}
{
 config="",
library="libclientBcast.so",
function="makeBroadcastPlugin"
}

voucherRechargeOptions = {
 voucherRechargeTriggers = ["VRW "]
}

voucherTypeRechargeOptions = {
 voucherTypeRechargeTriggers = ["VTR "]
}

beLocationPluginLib = "/IN/service_packages/CCS/lib/libGetccsBeLocation.so"

PostPaidMSISDNInvalidRange = [
{
 rangeStart = "0"
 rangeStop = "99999999999"
 postPaidMsisdn = "441473289900"
},
{
 rangeStart = "100000000000"
 rangeStop = "199999999999"
 postPaidMsisdn = "4412061234567"
},
{
 rangeStart = "200000000000"
 rangeStop = "299999999999"
 postPaidMsisdn = "481263833252"
}
]

throttleRate = 10
notificationPollInterval = 30
notificationRequestsEnabled = false
overrideDiscountType = "Compound"

WalletNameTranslations = {
 Personal = "Primary"
 Business = "Secondary"
}

TimeUnitExponent = 2
allowNonCashBalanceForAmountMethods = false

NotificationDatabase = {
 ArraySize = 10
 ExpiryDays = 5
 IntermediateCommit = true
}
```
localTZ = "GMT"
allowBalanceDateAfterWalletDate = false
remoteSMF1Name = "usms1_SMF"
}

CCS = {
    ccsSSMMaster = {
        filename = "/tmp/ccsSSMRequest.fifo"
        hostname = "ctlbscp"
        port = 1495
        dispatcherOracleUsername = "scp"
        dispatcherOraclePassword = "scp"
    }
}

**Parameters of osaCham**

Individual parameters used in the osaCham section of Oracle.conf.scs are described below.

**allowBalanceDateAfterWalletDate**

**Syntax:**

allowBalanceDateAfterWalletDate = <tf>

**Description:**

Allow a balance expiry date later than the wallet expiry date when set by extension or by exact date.

**Type:**

Boolean

**Optionality:**

Optional (default used if not set).

**Allowed:**

true, false

**Default:**

false

**Notes:**

Example:

allowBalanceDateAfterWalletDate = false

**AllowDirectPostPaid**

**Syntax:**

AllowDirectPostPaid = <tf>

**Description:**

This parameter permits or refuses direct debits or direct credits on post-paid (non-existent) accounts.

**Type:**

Boolean

**Optionality:**

Allowed:

true Direct debits and credits will appear to be successful on post-paid accounts, though no billing will actually happen. An EDR is written, allowing a summary bill to be sent later.

false Direct debits and credits will not be successful on post-paid accounts.

**Default:**

Notes:

Example:

AllowDirectPostPaid = true

**AllowInvalidReleaseSequenceNumber**

**Syntax:**

AllowInvalidReleasedSequenceNumber = <tf>

**Description:**

This parameter can permit an invalid sequence number in a release request. The parameter allows crashed clients to dispose of previous sessions easily.

**Type:**

Boolean
Optionality:
Allowed: true  A release request may contain an invalid sequence number.
false  A release request may not contain an invalid sequence number.

Default:
Notes:
Example: AllowInvalidReleasedSequenceNumber = true

allowNonCashBalanceForAmountMethods
Syntax: allowNonCashBalanceForAmountMethods = <tf>
Description: Allow non-cash balance types for amount methods.
Type: Boolean
Optionality: Optional (default used if not set).
Allowed: true, false
Default: false
Notes: This is an extension and is not part of the OSA standard.
Example: allowNonCashBalanceForAmountMethods = false

AlwaysSendNotification
Syntax: AlwaysSendNotification = <tf>
Description: Whether to always send a Notification, regardless of what is specified in the method parameter.
If the configuration parameter is:
- ‘true’ then a short message request will be sent.
- ‘false’ then the request type method parameter will be used to determine whether or not to send the short message request.
Type: Boolean
Optionality: Optional
Allowed: true, false
Default: false
Notes:
Example: AlwaysSendNotification = true

AlwaysUseAbsoluteCostValueForNotifications
Syntax: AlwaysUseAbsoluteCostValueForNotifications = <tf>
Description: Whether or not to always use the absolute value for the recharge cost in the notification.
Type: Boolean
Optionality: Optional (default used if not set).
Allowed: true, false
Default: false
Notes:
Example: AlwaysUseAbsoluteCostValueForNotifications = false
### beLocationPluginLib

**Syntax:**

```
beLocationPluginLib = "<PathAndName>"
```

**Description:**
The path and name of the BE location library plug-in.

**Type:**

String

**Optionality:**

Allowed

**Default:**

You can use the beLocationPluginLib as an alternative to the billingEngines parameter array. See billingEngines. If a billingEngines parameter array is defined in the configuration file, the beLocationPluginLib parameter is ignored.

**Example:**

```
beLocationPluginLib = "/IN/service_packages/CCS/lib/libGetccsBeLocation.so"
```

### CDRDirectory

**Syntax:**

```
CDRDirectory = "<directory>"
```

**Description:**
The directory into which EDR files are written.

**Type:**

String

**Optionality:**

Allowed

**Default:**

"/IN/service_packages/OSA/cdr"

**Notes:**

**Example:**

```
CDRDirectory = "/IN/service_packages/OSA/cdr"
```

### CDRMaxAge

**Syntax:**

```
CDRMaxAge = <time>
```

**Description:**
The maximum age of EDRs, in seconds, before they are written to disk.

**Type:**

Integer

**Optionality:**

Allowed

**Default:**

**Notes:**

**Example:**

```
CDRMaxAge = 60
```

### CDRMaxSize

**Syntax:**

```
CDRMaxSize = <number>
```

**Description:**
The maximum number of EDRs that can accumulate before they are written to disk.

**Type:**

Integer

**Optionality:**

Allowed

**Default:**

**Notes:**

**Example:**

```
CDRMaxSize = 100
```
clientName
Syntax: clientName = "<name>"
Description: Identifies this instance of the OSA gateway to the beServer process on the BE.
Type: String
Optionality: Allowed:
Default:
Notes: The name must be unique in the pool of connecting agents of a BE pair.
Example: clientName = "exampleConfigBeClient"

collectionRetryTime
Syntax: collectionRetryTime = <time>
Description: The interval, in seconds, separating attempts by BeClient to fix a failed connection to a billing engine.
Type: Integer
Optionality: Allowed:
Default:
Notes:
Example: collectionRetryTime = 5

fifoFileName
Syntax: fifoFileName = "<PathAndFileName>"
Description: The path to, and name of, the short message requests file.
Type: String
Optionality: Allowed:
Default:
Notes: The file name must be the same as that of the ccsSSMMaster.
Example: fifoFileName = "/tmp/ccsSSMRequest.fifo"

heartbeatPeriod
Syntax: heartbeatPeriod = <time>
Description: The interval, in microseconds, separating attempts by BDClient to detect a heartbeat on all open billing engine connections.
Type: Integer
Optionality: Allowed:
Default:
Notes:
Example: heartbeatPeriod = 5000000

localTZ
Syntax: localTZ = "<tz>"
Description: Local time zone of the client.
Type: String
Optionality: Optional (default used if not set).
Allowed: The format is as recognised by the unix system. Refer to the ACS Technical Guide for a list of time zones.
Default: "GMT"
Notes: Example: localTZ = "GMT"

ipAccountManagerIORFile
Syntax: ipAccountManagerIORFile = "<PathAndFileName>
Description: The path and name of the file to which the ipAccountManager IOR is written.
Type: String
Optionality: Optional.
Allowed: Default: "/<processesCurrentWorkingDirectory>/ipAccountManagerIOR.txt"
<processesCurrentWorkingDirectory> is the current working directory of the system processes.
Notes: If this parameter is omitted, the default path and name are used.
Example: ipAccountManagerIORFile = "/tmp/ipAccountManagerIOR.txt"

ipChargingManagerIORFile
Syntax: ipChargingManagerIORFile = "<PathAndFileName>
Description: The path and name of the file to which the IpChargingManager IOR is written.
Type: String
Optionality: The ipChargingManagerIORFile parameter is optional.
Allowed: Default: "/<processesCurrentWorkingDirectory>/ipChargingManagerIOR.txt"
<processesCurrentWorkingDirectory> is the current working directory of the system processes.
Notes: If this parameter is omitted, the default path and name are used.
Example: ipChargingManagerIORFile = "/tmp/ipChargingManagerIOR.txt"

MaximumAmountValue
Syntax: MaximumAmountValue = <mav>
Description: The maximum amount allowed in directDebitAmountReq() and directCreditAmountReq() operations.
Type: Integer
Optionality: Optional.
Allowed: Default: Notes: - An amount exceeding <mav> causes a request failure.
- <mav> may not be exceeded when confirming an amount on the billing engine. For this reason, the debitAmountReq() and creditAmountReq() operations are affected by the MaximumAmountValue parameter.
Example: MaximumAmountValue = 1000
MaximumEventsValue
Syntax: \[ \text{MaximumEventsValue} = <\text{mev}> \]
Description: \(<\text{mev}>\) is the number of events allowed in directDebitUnitReq() and
directCreditUnitReq() operations.
Type: Integer
Optionality: Optional.
Allowed: 
Default: Notes: 
• If \(<\text{mev}>\) is exceeded, a request failure is issued.
• \(<\text{mev}>\) may not be exceeded when confirming events on the billing
engine. For this reason, the debitUnitReq() and creditUnitReq()
operations are affected by the MaximumEventsValue parameter.

Example:

MaximumSecondsValue
Syntax: \[ \text{MaximumSecondsValue} = <\text{msv}> \]
Description: The maximum duration, in seconds, allowed in directDebitUnitReq() and
directCreditUnitReq() operations.
Type: Integer
Optionality: Optional.
Allowed: Default: Notes: 
• A duration exceeding \(<\text{msv}>\) will cause a request failure.
• \(<\text{msv}>\) may not be exceeded when confirming time on the billing engine.
  For this reason, the debitUnitReq() and creditUnitReq() operations are
  affected by the MaximumSecondsValue parameter.

Example: \[ \text{MaximumSecondsValue} = 3600 \]

NotificationDatabase
Syntax: \[ \text{NotificationDatabase} = \{ \]
\text{ArraySize} = <\text{as}>
\text{ExpiryDays} = <\text{ed}>
\text{IntermediateCommit} = <\text{tf}>
\} \]
Description: NotificationDatabase is a parameter group.
Type: Parameter group.
Optionality: Allowed: Default: Notes: Example:

ArraySize
Syntax: \[ \text{ArraySize} = <\text{as}> \]
Description: The maximum number of records that can be created or accessed in an OCI call
to the OSA_TAGGED_BALANCE table.
Type: Integer
Chapter 2

### Optionality:
- Allowed:
- Default:
- Notes: This parameter is part of the NotificationDatabase parameter group.
- Example: ArraySize = 10

#### ExpiryDays
- Syntax: ExpiryDays = <ed>
- Description: The time, in days, during which a notification remains valid.
- Type: Integer
- Optionality:
- Allowed:
- Default:
- Notes: This parameter is part of the NotificationDatabase parameter group.
- Example: ExpiryDays = 5

#### IntermediateCommit
- Syntax: IntermediateCommit = <ic>
- Description: This parameter may permit a change to be committed to the OSA_TAGGEBALANCE table between each creation or deletion.
- Type: Boolean
- Optionality:
- Allowed: true The change is committed to the OSA_TAGGEBALANCE table between each creation or deletion.
- false The change is not committed.
- Default:
- Notes: This parameter is part of the NotificationDatabase parameter group.
- Example: IntermediateCommit = true

#### notificationPollInterval
- Syntax: notificationPollInterval = <time>
- Description: The interval, in seconds, separating polls of the BE.
- Type: Integer
- Optionality:
- Allowed:
- Default: 10
- Notes:
- Example: notificationPollInterval = 10

#### notificationRequestsEnabled
- Syntax: notificationRequestsEnabled = <tf>
- Description: Whether or not to enable OSA notifications.
- Type: Boolean
- Optionality: Optional (default used if not set).
Allowed: true, false
Default: false
Notes: If 'false' this will prevent the unnecessary UBE requests: NOTR_Req and NOTC_Req
- NOTR_Req requests are sent at the configured 'notificationPollInterval' interval.
- NOTC_Req requests are sent to clear the currently configured OSA notifications on the UBE.
Example: notificationRequestsEnabled = false

OracleUsernamePassword
Syntax: OracleUsernamePassword = "<name>/<password>"
Description: The Oracle username and password for the SCF.
Type: String
Optionality: Allowed:
Default: "scp/scp"
Notes: Example: OracleUsernamePassword = "scp/scp"

ORBDebugLevel
Syntax: ORBDebugLevel = <level>
Description: Sets the level of debugging information provided.
Type: Integer
Optionality: Allowed: 0 ≤<level>≤ 6.
Default:
Notes: For production systems, set to 0.
Example: ORBDebugLevel = 0

ORBIIOPAddr
Syntax: ORBIIOPAddr = "<IPNumber>"
Description: Binds ORBit's listening socket to address IP number.
Type: String
Optionality: Optional.
Allowed: A string of characters that make up an Internet protocol number expressed in the dotted-decimal format.
Default:
Notes: If the ORBIIOPAddr parameter is not specified, the IP address of the host machine is used.
Example: ORBIIOPAddr = "192.1.2.174"

ORBIIOPPort
Syntax: ORBIIOPPort = <pn>
Description: Binds ORBit's listening socket to port <pn>.
Type: Integer
Optionality: Optional.
Allowed: \( <pn> \geq 1024 \)
Default: 
Notes: 
Example: ORBIIOPPort = 1025

**ORBIIOPSetKeepalive**

**Syntax:** ORBIIOPSetKeepalive = \(<tf>\)

**Description:** Set the keepalive flag on new orb connections.

**Type:** Boolean

**Optionality:** Optional (default used if not set).

**Allowed:** true, false

**Default:** true

**Notes:** If set, the tcp socket will not time out due to lack of activity.

**Example:** ORBIIOPSetKeepalive = true

**overrideDiscountType**

**Syntax:** overrideDiscountType = "<text>"

**Description:** Factors OSA discounts into the BE discounts from the tariff.

**Type:** String

**Optionality:** 

**Allowed:**
- **Override** The OSA discount is ignored and the final rate = the initial rate \( \times \) BE discount.
- **Compound** The discounts are multiplied together and the final rate = initial rate \( \times \) BE discount \( \times \) OSA discount.
- **Cumulative** The discounts are added together and the final rate = initial rate \( \times \) (BE discount \( + \) OSA discount).

**Default:**

**Notes:**

**Example:** overrideDiscountType = "Compound"

**OverrideInitialEventCdrType** - 2.6.0.7 - changed to uppercase

**Syntax:** OverrideInitialEventCdrType = true|false

**Description:** Whether or not to override the CDR type for the reserveUnitReq (events).

**Type:** Boolean

**Optionality:** Optional (default used if not set).

**Allowed:** true, false

**Default:** true

**Notes:**

**Example:** OverrideInitialEventCdrType = true
OverrideInitialTariffCdrType

Syntax: OverrideInitialTariffCdrType = true | false

Description: Whether or not to override the CDR type for the reserveUnitReq (seconds).

Type: Boolean

Optionality: Optional (default used if not set).

Allowed: true, false

Default: true

Notes:

Example: OverrideInitialTariffCdrType = true

plugins

Syntax: plugins = [
    {
        config = "<tag>",
        library = "<lib>",
        function = "<function>"
    }
    {
        <VRT config>
    }
    {
        <Broadcast plugin config>
    }
]

Description: The plugins configuration items are compulsory in the configuration file to enable the new voucher and voucher type recharges.

Type: Parameter array.

Optionality: Allowed:

The voucher recharge plugins are:
- The Voucher Recharge (VRW)
- Voucher Type Recharge (VTR)
- Broadcast plugin for VRW and VTR

Default:

Notes:

Example: plugins = [
    {
        config = "voucherRechargeOptions",
        library = "libccsClientPlugins.so",
        function = "makeVoucherRechargePlugin"
    }
    {
        config = "voucherTypeRechargeOptions",
        library = "libccsClientPlugins.so",
        function = "makeVoucherTypeRechargePlugin"
    }
    {
        config = "",
        library = "libclientBcast.so",
        function = "makeBroadcastPlugin"
    }
]
config
Syntax: config = "<tag>"
Description: The configuration tag of the plugin.
Type: String
Optionality: Mandatory
Allowed: 
Default: 
Notes: This parameter is part of the plugins parameter array.
Example: config = "voucherRechargeOptions"

function
Syntax: function = "<cFunction>"
Description: The name of the external C programming language function within the library.
Type: String
Optionality: Mandatory
Allowed: 
Default: 
Notes: This parameter is part of the plugins parameter array.
Example: function = "makeVoucherRechargePlugin"

library
Syntax: library = "<fileName>"
Description: Names the library that contains the plugin.
Type: String
Optionality: Mandatory
Allowed: 
Default: 
Notes: This parameter is part of the plugins parameter array.
Example: library = "libccsClientPlugins.so"

PostPaidMSISDNInvalidRange
Syntax: PostPaidMSISDNInvalidRange = [
    { 
        rangeStart = "<1stRangeStart>"
        rangeStop = "<1stRangeStop>"
        postPaidMsisdn = "<1stppm>"
    },
    ...
    ...
    
    <nth Range>
}

Description: The PostPaidMSISDNInvalidRange parameter array maps a range of mobile station ISDN (MSISDN) numbers to a valid MSISDN. The valid MSISDN is substituted for the charging party in the rate request that follows.
Type: Parameter array.
Optionality: 

PostPaidMSISDNInvalidRange = [{
  rangeStart = "0"
  rangeStop = "99999999999"
  postPaidMsisdn = "441473289900"
},
{  
  rangeStart = "100000000000"
  rangeStop = "199999999999"
  postPaidMsisdn = "4412061234567"
},
{  
  rangeStart = "200000000000"
  rangeStop = "299999999999"
  postPaidMsisdn = "481263833252"
}]

postPaidMsisdn

Syntax: postPaidMsisdn = "<nthppm>"

Description: The mobile station ISDN number to which the nth number range is mapped.

Type: String

Optionality: Allowed

Notes: This parameter is part of the PostPaidMSISDNInvalidRange parameter array.

Example: postPaidMsisdn = "441473289900"

rangeStart

Syntax: rangeStart = "<nthRangeStart>"

Description: The start number of the nth MSISDN billing range.

Type: String

Optionality: Allowed

Notes: This parameter is part of the PostPaidMSISDNInvalidRange parameter array.

Example: rangeStart = "0"

rangeStop

Syntax: rangeStop = "<nthRangeStop>"

Description: The end number of the nth MSISDN billing range.

Type: String

Optionality: Allowed

Notes: This parameter is part of the PostPaidMSISDNInvalidRange parameter array.

Example: rangeStop = "99999999999"
remoteSMF1Name

Syntax: remoteSMF1Name = "<name>"

Description: The entry in the Oracle tnsnames.ora file for the remote database link to the SMS.

Type: String

Optionality: Optional, but see notes for consequence of not being set.

Allowed: 

Default: No default

Notes: This is used to attempt reconnection if the database link fails. If this is not set, automatic reconnection will not be attempted and an error will be returned to the client.

Example: remoteSMF1Name = "usms1_SMF"

requestTimeout

Syntax: requestTimeout = <period>

Description: The maximum time, in milliseconds, that the gateway will wait for a response from the BE. After that time the gateway will abandon the request.

Type: Integer

Optionality: 

Allowed: 

Default: 

Notes: 

Example: requestTimeout = 10000

SMSLanguage

Syntax: SMSLanguage = "<text>"

Description: Defines the language used for short message requests, if the subscriber’s language cannot be determined from the profile.

Type: String

Optionality: 

Allowed: 

Default: unknown

Notes: 

Example: SMSLanguage = "unknown"

throttleRate

Syntax: throttleRate = <rate>

Description: The number of connections allowed per second.

Type: Integer

Optionality: 

Allowed: 

Default: 

Notes: If 0, throttling is disabled.

Example: throttleRate = 10
Chapter 2

TimeUnitExponent
Syntax:  
```
TimeUnitExponent = <ex>
```  
Description: The 'Time' unit exponent to translate the UBE internal time balance units into seconds.
Type: Integer
Optionality: Optional (default used if not set).
Allowed: valid values 0 .. 4
Default: 2
Notes: Currently the UBE internal time balance units is 100th's second.
Example: TimeUnitExponent = 2

voucherRechargeOptions
Syntax:  
```
voucherRechargeOptions = {
    voucherRechargeTriggers = [<triggers>]
}
```  
Description: The voucher recharge plugin options.
Type: String array
Optionality: Mandatory
Allowed: 
Default: 
Notes: See plugins (on page 26).
Example: voucherRechargeOptions = {
    voucherRechargeTriggers = ["VRW "]
}

voucherRechargeTriggers
Syntax:  
```
voucherRechargeTriggers = ["<trig1 >,<trign >"]
```  
Description: The Voucher Recharge triggers.
Type: String array
Optionality: Mandatory
Allowed: 
Default: 
Notes: 
Example: voucherRechargeTriggers = ["VRW "]

voucherTypeRechargeOptions
Syntax:  
```
voucherTypeRechargeOptions = {
    voucherTypeRechargeTriggers = [<triggers>]
}
```  
Description: The voucher type recharge plugin options.
Type: String array
Optionality: Mandatory
Allowed: 
Default: 
Notes: See plugins (on page 26).
Example:

```csharp
vouchertypeRechargeOptions = {
    vouchertypeRechargeTriggers = ["VTR "]
}
```

**vouchertypeRechargeTriggers**

**Syntax:**

```csharp
vouchertypeRechargeTriggers = ["<trig1>,<trign>"]
```

**Description:**
The Voucher Type Recharge triggers.

**Type:**
String array

**Optionality:**
Mandatory

**Allowed:**

**Default:**

**Notes:**

Example:

```csharp
vouchertypeRechargeTriggers = ["VTR "]
```

**WalletNameTranslations**

**Syntax:**

```csharp
WalletNameTranslations = {
    Personal = "<new name>",
    Business = "<new name>"
}
```

**Description:**
Used when translating between the old and new wallet type names used in Prepaid Charging.

**Type:**
Parameter group.

**Optionality:**
Optional. No wallet type name translation occurs if both 'Personal' and 'Business' are not present.

**Allowed:**

**Default:**

**Notes:**

Example:

```csharp
WalletNameTranslations = {
    Personal = "Primary",
    Business = "Secondary"
}
```

**Business**

**Syntax:**

```csharp
Business = "<newname>"
```

**Description:**
Translating between the old and new wallet type name used in Prepaid Charging.

**Type:**
String

**Optionality:**
Optional. See WalletNameTranslations.

**Allowed:**

**Default:**

**Notes:**

This parameter is part of the WalletNameTranslations parameter group.

Example:

```csharp
Business = "Secondary"
```

**Personal**

**Syntax:**

```csharp
Personal = "<newname>"
```

**Description:**
Translating between the old and new wallet type name used in Prepaid Charging.

**Type:**
String

**Optionality:**
Optional. See WalletNameTranslations.

**Allowed:**

**Default:**

**Notes:**

This parameter is part of the WalletNameTranslations parameter group.

Example:

```csharp
Business = "Secondary"
```
Default: This parameter is part of the WalletNameTranslations parameter group.

Example: Personal = "Primary"

**CCS parameters**

The following sections must be included in the CCS section of the osaChamScs.

Here is an example of the CCS section of the configuration file.

```plaintext
CCS = {
    ccsSSMMaster = {
        filename = "/tmp/ccsSSMRequest.fifo"
        hostname = "ctlbscp"
        port = 1495
        dispatcherOracleUsername = "scp"
        dispatcherOraclePassword = "scp"
    }
}
```

**ccsSSMMaster**

The parameters available to configure the ccsSSMDispatcher binary are described below. The binary is required for sending SMSs to the CCS SSM master on the voice SLC.

**dispatcherOraclePassword**

*Syntax:* dispatcherOraclePassword = "<pw>"

*Description:* Oracle password for ccsSSMDispatcher database connection.

*Type:* String

*Optionality:* Mandatory

*Allowed:*

*Default:*

*Notes:*

*Example:* dispatcherOraclePassword = "scp"

**dispatcherOracleUsername**

*Syntax:* dispatcherOracleUsername = "<uname>"

*Description:* Oracle username for ccsSSMDispatcher database connection.

*Type:* String

*Optionality:* Mandatory

*Allowed:*

*Default:*

*Notes:*

*Example:* dispatcherOracleUsername = "scp"

**filename**

*Syntax:* filename = "<fname>"

*Description:* File name to use for short message requests.

*Type:* String

*Optionality:* Mandatory

*Allowed:*

*Default:*

---

32  NCC OSA CHAM Technical Guide
Notes:
Example: 

filename = "/tmp/ccsSSMRequest.fifo"

hostname
Syntax: 

hostname = "<hname>"
Description: Host name for the machine the ccsSSMMaster is running on.
Type: String
Optionality: Mandatory
Allowed:
Default:
Notes:
Example: 

hostname = "ctlbscp"

port
Syntax: 

port = <pnum>
Description: Port to connect to on the machine the ccsSSMMaster is running on.
Type:
Optionality: Optional
Allowed:
Default:
Notes:
Example: 

port = 1495

Configuring Balance Type Currencies
To ensure that balance type currencies work correctly each CCS balance type must have at least currency one setup in CCS.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>In the SMS Services Menu select the CCS - Resources option.</td>
</tr>
<tr>
<td>2</td>
<td>In the Resources screen, select the Customer from the Service Provider drop down box.</td>
</tr>
<tr>
<td>3</td>
<td>On the Balance Types tab, select the balance type you want to edit from the grid.</td>
</tr>
<tr>
<td>4</td>
<td>Edit each balance type available to ensure that a &quot;Currency One&quot; item has been created for each.</td>
</tr>
<tr>
<td>5</td>
<td>Enter the correct currency code for each balance type. See the CCS User Guide for details on Adding Balance Types.</td>
</tr>
</tbody>
</table>

Configuring osaChamBe

Introduction
The BE plugins require certain parameters to be read from a configuration file. A sample of these parameters has been provided below, as they should appear in the 'eserv.config' file. During the package installation process it will be necessary to add the osaCham section, with sub-sections:

- NotificationDatabase
The osaCham section is parsed by the OSA bOracleer plugins and beVWARS handlers.

You must also merge OSA-specific entries to the following sections:

- BE
- beVWARS
- CCS

This is the same configuration file used by other modules on the target platform. The plugins will not read 'example.Oracle.conf.be' file; it has been provided as an example only.

**osaCham parameters**

The parameters available to configure osaCham are described below.

**queryWalletTolerance**

Time (in milli-seconds) plugins should request the VWARS locks an account for following a query wallet request.

Default: 3000

Allowed: 

Note: The default used if no value specified.

**NotificationDatabase**

Use these parameters to configure the NotificationDatabase section.

**ArraySize**

Maximum number of records that can be created/accessed per OCI call in the OSA_TAGGED_BALANCE table.

Default: 100

Allowed: 

**ExpiryDays**

Time (days) notification is valid for.

Default: 1

Allowed: 

**IntermediateCommit**

If set to true, commits will happen between each record creation/deletion in the OSA_TAGGED_BALANCE table.

Default: true

Allowed: true, false

**NotificationQueueManager**

Use these parameters to configure the NotificationQueueManager.

**QueueSize**

The maximum number of reports that may be queued.

Default: 20

Allowed:
Note: Required.

QueueTimeout
Time (in seconds) before an unclaimed notification report in the queue will be discarded.
Default: 60
Allowed:
Note:

NumberOfSegments
How many segments of shared memory should be created.
Default: 2
Allowed:
Note: Values must be 2 or higher.

SegmentSize
How big (in MB) each segment should be.
Default: 1
Allowed: integer

ShmKeyFile
File name used to create the shared memory key.
Default: "/IN/service_packages/OSA/semKeyFile"
Allowed:

NotificationEvents
Use this parameter to configure the NotificationEvents.

SemaphoreKey
File name used to create the semaphores key.
Default: "/IN/service_packages/OSA/semKeyFile"
Allowed:

BalanceThreshold
Minimum amount for which notifications will be created.
Default: 10
Allowed:

NotificationServerHandler
Use these parameters to configure the NotificationServerHandler.

SemaphoreKey
File name used to create the semaphores key.
Default: "/IN/service_packages/OSA/semKeyFile"
Allowed:
TimeoutCheckPeriod
Frequency (in seconds) for checking for timeouts.
Default: 10
Allowed:

MaxReports
How many reports can be served per request from the OSA Gateway.
Default: 20
Allowed:

MessageTimeout
Time-to-live (in milliseconds) of the response message sent back to the OSA Gateway.
Default: 10,000,000
Allowed:

NumberOfSemaphores
How many semaphores to create.
Default: 1

BE parameters
The following entries need to be carefully merged. The sections will already exist in the eserv.config, so you must not just copy and paste them from the example file.

OSA Service Handlers library
You must add the entry for the OSA Service Handlers library to BE.beServer.handlers, as shown in the example.

Example BE section
Here is the BE section of the example config file.

```json
BE = {
    beServer = {
        handlers = {
            "osaServerNotificationHandlers.so"
        }
    }
}
```

beVWARS parameters
The following entries must be added to the beVWARS section of the config file.

plugins
You must add the OSA Cham entries for the libraries in the plugins sub-section, as shown in the example.

Example beVWARS section
Here is the beVWARS section of the example config file.

```json
beVWARS = {
    plugins = {
        "osaVWARSNotificationPlugins.so" # OSA Cham entry
    }
    handlers = {
```
"osaVWARSQueryHandlers.so"     # OSA Cham entry
}
}

Chapter 2

**CCS parameters**

To integrate OSA with CCS, the ccsVWARSActivation sub-section must be included in the CCS section of the `eserv.config` file, specifying details for all customers that may have account activated due to OSA requests.

Values marked `??` depend on your CCS configuration. For more information, see the *CCS Technical Guide*.

**balanceTypes**

Lists the balance types for the accounts that can be activated through the OSA Gateway. For each balance type you must specify the:

- ACS customer ID,
- the free SMS type and
- the expiry types.

See the example below.

**Example CCS section**

Here is the CCS section of the example config file.

```
CCS = {
  ccsVWARSActivation = {
    balanceTypes = [
      { acsCustomerId = ??,
        freeSmsType = ??
        setExpiryTypes = [ ??, ??, ... ]
      }
    ]
  }
}
```

**Example configuration file**

This is the sample configuration file, `example.Oracle.conf.be`, located in the `/IN/service_packages/etc` directory. Merge the entries in this file with the global configuration file, `eserv.config`.

```
osaCham = {
  queryWalletTolerance = 5000

  NotificationDatabase = {
    ArraySize = 1000
    ExpiryDays = 20
    IntermediateCommit = true
  }

  NotificationQueueManager = {
    QueueSize = 20
    QueueTimeout = 60
    NumberOfSegments = 2
    SegmentSize = 1
    ShmKeyFile = "/tmp/shm"
  }

  NotificationEvents = {
```
SemaphoreKey = "/tmp/sem"
BalanceThreshold = 3000

NotificationServerHandler = {
  SemaphoreKey = "/tmp/sem"
  TimeoutCheckPeriod = 20
  MaxReports = 10
  MessageTimeout = 5000
  NumberOfSemaphores = 1
}

# the following entries need to be carefully merged.
# The sections will already exist, so do not just
# copy and paste.
BE = {
  beServer = {
    handlers = {
      "osaServerNotificationHandlers.so"
    }
  }
}

beVWARS = {
  plugins = {
    "osaVWARSNotificationPlugins.so"  # OSA Cham entry
  }
  handlers = {
    "osaVWARSQueryHandlers.so"        # OSA Cham entry
  }
}

CCS = {
  ccsVWARSActivation = {
    balanceTypes = [
      { acsCustomerId = ??,
        freeSmsType = ??,
        setExpiryTypes = [ ??, ??, ... ]
      }
    ]
  }
}

Client Configuration

Introduction

The osaChamClient requires configuring on the host machine.

- osaChamClient.conf configuration file on the host machine in osaChamClient_v2.X.X/etc directory. This configuration file:
  - includes defaults used within the client, for example the MSISDN and called/calling numbers
  - specifies the location for OSA CHAM gateway CORBA location files which are written when the gateway starts.
- jacobr.properties configuration file on the host machine in osaChamClient_v2.X.X/bin directory.
It includes a definition of the host machine IP address.

**Example osaChamClient.conf file**

Here is an example config for osaChamClient. The comments are removed for clarity.

```plaintext
osaCham = {
    defaultBaseDirectory =
    defaultTmp = tmp
    maxChargingSession = 100
    maxUsers = 10
    maxFileCycles = 10
    msisdn = 361
    billingEngineId = 9999999
    walletId = 1000
    bucketId = 2000
    callingNumber = 07790
    calledNumber = 07870
    discount = 10
    event = My Event Class/My Event Name
    lifeTimeHint = 180
    walletType = Promotional
    balanceType = Promotional Cash
    balanceCascade = General Cascade
    sendSMSFlag = true

    appAccountManagerIORFile = appAccountManagerIOR.txt
    ipAccountManagerIORFile = /export/home/pgoodhan/ior/chamdev/ipAccountManagerIOR.txt
    ipChargingManagerIORFile = /export/home/pgoodhan/ior/chamdev/ipChargingManagerIOR.txt
}
```

**osaChamClient.conf parameters**

Here are the parameters in the osaCham section of the configuration file.

**appAccountManagerIORFile**

Syntax: `appAccountManagerIORFile = <file>`

Description: Default Application Account Manager IOR file location for createNotification queries. The Application Account Manager IOR will be written to this location.

Type: String

Optionality: Allowed

Default: `appAccountManagerIOR.txt`

Notes:

Example: `appAccountManagerIORFile = appAccountManagerIOR.txt`
balanceCascade

Syntax: \texttt{balanceCascade = <name>}

Description: Balance type cascade name to be used in the Charged Party parameter for the following queries:
- \texttt{reserveAmountReq}
- \texttt{directDebitAmountReq}

Balance type cascade name to be used in the SubAddress string parameter for the following query:
- \texttt{queryBalanceReq}

This will replace the balance type name in the charging parameter if present.

Type: String
Optionality: Allowed:
Default: General Cascade
Notes:
Example: \texttt{balanceCascade = General Cascade}

defaultBaseDirectory

Syntax:

Description: Base directory to be used for creating response file.

Type: String
Optionality: Allowed:
Default: 
Notes:
Example:

defaultTmp

Syntax: \texttt{defaultTmp = "<dir>"}

Description: Directory appended to base directory.

Type: String
Optionality: Allowed:
Default: "tmp"
Notes:
Example: \texttt{defaultTmp = "tmp"}

ipAccountManagerIORFile

Syntax: \texttt{ipAccountManagerIORFile = <name>}

Description: Account Manager IOR file location.

Type: String
Optionality: Allowed:
Default: 
Notes:
Example: ipAccountManagerIORFile =
/export/home/jlb/ior/chamdev/ipAccountManagerIOR.txt

ipChargingManagerIORFile
Syntax: ipChargingManagerIORFile = <name>
Description: Charging Manager IOR file location.
Type: String
Optionality: 
Allowed: 
Default: 
Notes: 
Example: ipChargingManagerIORFile =
/export/home/jlb/ior/chamdev/ipAccountManagerIOR.txt

maxChargingSession
Syntax: maxChargingSession = <num>
Description: Maximum number of concurrent charging sessions.
Type: Integer
Optionality: 
Allowed: 
Default: 100
Notes: 
Example: maxChargingSession = 100

maxFileCycles
Syntax: maxFileCycles = <num>
Description: Maximum number of times the template file will be read in a soak test.
Type: Integer
Optionality: 
Allowed: 
Default: 10
Notes: 
Example: maxFileCycles = 10

maxUsers
Syntax: maxUsers = <num>
Description: Maximum number of users in a queryBalanceReq query.
Type: Integer
Optionality: 
Allowed: 
Default: 10
Notes: 
Example: maxUsers = 10
msisdn
Syntax: \texttt{msisdn = <num>}
Description: Default MSISDN to be used in:
- the Charged Party parameter for reserveAmountReq,
directCreditAmountReq, and directDebitAmountReq, reserveUnitReq,
directCreditUnitReq, and directDebitUnitReq queries and
- the Address string parameter in the queryBalanceReq and
createChargingSession queries.
Type:
Optionality: Allowed:
Default: 361
Notes:
Example: \texttt{msisdn = 361}

Queries
The eight parameters below set the defaults used in the Charged Party parameter for the following
queries:
- reserveAmountReq
- directCreditAmountReq
- directDebitAmountReq
- reserveUnitReq
- directCreditUnitReq, and
- directDebitUnitReq.

billingEngineId
Syntax: \texttt{billingEngineId = <id>}
Description: Default billing engine ID to be used in the Charged Party parameter.
Type: Integer
Optionality: Allowed:
Default: 9999999
Notes:
Example: \texttt{billingEngineId = 9999999}

bucketId
Syntax: \texttt{bucketId = <id>}
Description: Default bucket ID to be used in the Charged Party parameter.
Type: Integer
Optionality: Allowed:
Default: 2000
Notes:
Example: \texttt{billingEngineId = 2000}
calledNumber
Syntax: calledNumber = <num>
Description: Default called number to be used in the Charged Party parameter.
Type: 
Optionality: 
Allowed: 
Default: 
Notes: 
Example: calledNumber = 07870

callingNumber
Syntax: callingNumber = <num>
Description: Default calling number to be used in the Charged Party parameter.
Type: 
Optionality: 
Allowed: 
Default: 
Notes: 
Example: callingNumber = 07790

discount
Syntax: discount = <num>
Description: Default discount value to be used in the Charged Party parameter.
Type: Integer
Optionality: 
Allowed: 
Default: 10
Notes: 
Example: discount = 10

event
Syntax: event = <classname>
Description: Default event to be used in the Charged Party parameter.
Type: String
Optionality: 
Allowed: 
Default: My EventClass/My Event Name
Notes: 
Example: event = My Event Class/My Event Name

lifeTimeHint
Syntax: lifeTimeHint = <val>
Description: Default reservation LifeTimeHint value to be used in the Charged Party parameter.
Type: Integer
Optionality:
Allowed:
Default: 180
Notes:
Example: lifeTimeHint = 180

walletId
Syntax: walletId = <id>
Description: Default wallet ID to be used in the Charged Party parameter.
Type:
Optionality:
Allowed:
Default: 1000
Notes:
Example: walletId = 1000

sendSMSFlag
Syntax: sendSMSFlag = <tf>
Description: Default send SMS flag value to be used in the Charged Party parameter for the following queries:
- reserveAmountReq
- directCreditAmountReq
- directDebitAmountReq
- reserveUnitReq
- directCreditUnitReq
- directDebitUnitReq
Type: Boolean
Optionality:
Allowed:
Default: true
Notes:
Example: sendSMSFlag = true

Types
The following two parameters define the default wallet type and balance type name to be used in the Charged Party parameter for the following queries:
- reserveAmountReq
- directCreditAmountReq
- directDebitAmountReq
- reserveUnitReq
- directCreditUnitReq
- directDebitUnitReq
These parameters also define the default wallet type and balance type name to be use for the SubAddress string parameter in the following queries:
- queryBalanceReq
createChargingSession

balanceType

**Syntax:**
\[
\text{balanceType} = <\text{btype}>
\]

**Description:**
Default balance type name to be used in the Charged Party parameter.

**Type:** String

**Optionality:**
Allowed:

**Default:** General Cash

**Notes:**

**Example:**
\[
\text{balanceType} = \text{Promotional Cash}
\]

walletType

**Syntax:**
\[
\text{walletType} = <\text{wtype}>
\]

**Description:**
Default wallet type name to be used in the Charged Party parameter.

**Type:** String

**Optionality:**
Allowed:

**Default:** Personal

**Notes:**

**Example:**
\[
\text{walletType} = \text{Promotional Cash}
\]

**jacob.properties**

The osaChamClient is configured in the jacob.properties file on the host machine.

```
jacob.orb.print_version=on
jacob.verbosity=2
jacob.use_appligator=off
jacob.use_imr=off
jacob.security.support_ssl=off
jacob.poa.monitoring=off
OAIAddr=xxx.xxx.xxx.xxx(IP address of client machine)
```
Overview

Introduction

This chapter explains the alarms which may be generated by the application, probable causes and recommended responses.

In this chapter

This chapter contains the following topics.

Alarm Topic Description 47
OSA Client Interface Alarms 49

Alarm Topic Description

Introduction

The following table lists the alarms that will be generated by the client interface during execution. These messages are logged using the system syslog functionality.

The exact action taken for each logged message is dependent on the hosts systems configuration, generally set in the /etc/syslog.conf configuration file.

Severity levels

This table describes the alarms severity levels.

<table>
<thead>
<tr>
<th>Level</th>
<th>Abbr</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critical</td>
<td>C</td>
<td>These alarms are raised when the application has encountered an error which indicates that the system is unable to function.</td>
</tr>
<tr>
<td>Error</td>
<td>E</td>
<td>These alarms indicate the application has encountered a serious problem completing a necessary task and could not complete the task.</td>
</tr>
<tr>
<td>Warning</td>
<td>W</td>
<td>Warnings are raised to indicate the application encountered a problem completing a non-mission critical task.</td>
</tr>
<tr>
<td>Notice</td>
<td>N</td>
<td>Notices are raised to indicate that the application has completed a task successfully.</td>
</tr>
</tbody>
</table>

Alarm text and variables

The %d and %s symbols represent variables within the alarm text. These values are generated by the subsystem and added to the message when the alarm is raised.

Usually the %d is a number and the %s is text in the context of the message to complete the alarm message. Occasionally other % symbols are also used (for example, %u) for different variables.
Further information

For more information about system alarms, see the following Solaris manual pages:

<table>
<thead>
<tr>
<th>Command</th>
<th>Manual Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>openlog</td>
<td>syslog (3c)</td>
<td>control system log</td>
</tr>
<tr>
<td>syslog</td>
<td>syslog (3c)</td>
<td>control system log</td>
</tr>
<tr>
<td>syslog.conf</td>
<td>syslog.conf (4)</td>
<td>configuration file for syslogd system log daemon</td>
</tr>
</tbody>
</table>
## OSA Client Interface Alarms

### Critical errors

This table defines the Critical error messages for the OSA Client Interface.

<table>
<thead>
<tr>
<th>Number</th>
<th>Alarm Text</th>
<th>Reason</th>
<th>Remedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1004</td>
<td>CDR logger ended while charging sessions still active!</td>
<td>This is an internal error. The CDR logger has been ended while OSA charging sessions were still active. The OSA Gateway should have aborted all active sessions before ending the CDR logger.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1005</td>
<td>Asked to release unknown session ...!</td>
<td>This is an internal error. The CDR logger has been informed that a session has been released. However, it didn’t know that the session had been created in the first place.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1006</td>
<td>Asked to expire unknown session ...!</td>
<td>This is an internal error. The CDR logger has been informed that a session has expired. However, it didn’t know that the session had been created in the first place.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1007</td>
<td>Asked to abort unknown session ...!</td>
<td>This is an internal error. The CDR logger has been informed that a session has been aborted. However, it didn’t know that the session had been created in the first place.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1008</td>
<td>Session ... is not in the list of active sessions!</td>
<td>This is an internal error. An active session has been deleted that was not in the list of active sessions.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1009</td>
<td>Asked to send subsequent reservation without initial reservation!</td>
<td>This is an internal error. The OSA Gateway was about to send a SubsequentAmountReservationRequest to the BE without having sent an InitialAmountReservationRequest.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Details</td>
<td>Support Contact</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>1011</td>
<td>Received unknown error code from InitialAmountReservationNAck!</td>
<td>This is an internal error. The OSA Gateway received an error code as part of a InitialAmountReservationNAck message that it doesn’t recognise.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1014</td>
<td>Received response to unknown request ...!</td>
<td>This is an internal error. The OSA Gateway has received a response from the BE to a request that it thinks it has never sent.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1015</td>
<td>Received InitialAmountReservationAck in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a InitialAmountReservationAck from the Billing Engine but the original request was not a InitialAmountReservationRequest.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1016</td>
<td>Received InitialAmountReservationNAck in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a InitialAmountReservationNAck from the Billing Engine but the original request was not a InitialAmountReservationRequest.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1017</td>
<td>Received SubsequentAmountReservationAck in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a SubsequentAmountReservationAck from the Billing Engine but the original request was not a SubsequentAmountReservationRequest.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1018</td>
<td>Received SubsequentAmountReservationNAck in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a SubsequentAmountReservationNAck from the Billing Engine but the original request was not a SubsequentAmountReservationRequest.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1019</td>
<td>Received ReservationConfirmationAck in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a ReservationConfirmationAck from the Billing Engine but the original request was not a ReservationConfirmationRequest.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>Alarm Code</td>
<td>Description</td>
<td>Error Message</td>
<td>Contact Support</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>---------------</td>
<td>----------------</td>
</tr>
<tr>
<td>1020</td>
<td>Received ReservationConfirmationNAck in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a ReservationConfirmationNAck from the Billing Engine but the original request was not a ReservationConfirmationRequest.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1021</td>
<td>Received ReservationCancellationAck in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a ReservationCancellationAck from the Billing Engine but the original request was not a ReservationCancellationRequest.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1022</td>
<td>Received ReservationCancellationNAck in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a ReservationCancellationNAck from the Billing Engine but the original request was not a ReservationCancellationRequest.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1023</td>
<td>Received PerformDirectDeltaAck in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a PerformDirectDeltaAck from the Billing Engine but the original request was not a PerformDirectDeltaRequest.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1024</td>
<td>Received PerformDirectDeltaNAck in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a PerformDirectDeltaNAck from the Billing Engine but the original request was not a PerformDirectDeltaRequest.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1025</td>
<td>Received unknown message from BE!</td>
<td>This is an internal error. The OSA Gateway received a message from the BE that it does not recognise.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1026</td>
<td>Session timeout expired but session never expires!</td>
<td>This is an internal error. The session timeout has expired while the charging session should never expire since to reservation had been made against it.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1027</td>
<td>Session timeout expired before actual expiry time!</td>
<td>This is an internal error. The session timeout has expired before the charging session has actually due to expire.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1028</td>
<td>Could not initialise ORBit: ...</td>
<td>The OSA Gateway was unable to initialise ORBit (the ORB it is using internally).</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Message</td>
<td>Action</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>1029</td>
<td>No ORBit connection associated with file descriptor!</td>
<td>This is an internal error. The OSA Gateway thought that ORBit was using a particular file descriptor while ORBit claims that it has no connection associated with that file descriptor.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1030</td>
<td>Inconsistent file descriptor!</td>
<td>This is an internal error. ORBit thinks that a particular connection is associated with a file descriptor that is different from the one that we originally read the GIOP request from.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1032</td>
<td>Received response to unknown request ...!</td>
<td>This is an internal error. The OSA Gateway has received a response from the BE to a request that it thinks it has never sent.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1033</td>
<td>Received unexpected message from BE!</td>
<td>This is an internal error. The OSA Gateway received a message from the BE that it didn't expect because it was not QueryAccountAck nor a QueryAccountNAck.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1034</td>
<td>Unknown request type '...'!</td>
<td>This is an internal error. The OSA Gateway has sent a request to the BE that it does not even recognise itself.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1038</td>
<td>Received unknown error code from PerformDirectDeltaNAck!</td>
<td>This is an internal error. The OSA Gateway received an error code as part of a PerformDirectDeltaNAck message that it doesn't recognise.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1039</td>
<td>Asked to send deferred DII requests with empty queue!</td>
<td>This is an internal error. The OSA Gateway thinks there are deferred DII requests to send to the IpAppChargingSession but its internal queue is empty.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1043</td>
<td>Received unknown error code from SubsequentAmountReservationNAck!</td>
<td>This is an internal error. The OSA Gateway received an error code as part of a SubsequentAmountReservationNAck message that it doesn't recognise.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1045</td>
<td>Unable to create IOR file!</td>
<td>The OSA Gateway was unable to generate the IOR file for either the IpAccountManager or the IpChargingManager.</td>
<td>Please check the configuration file.</td>
</tr>
<tr>
<td>Alarm Code</td>
<td>Description</td>
<td>Error Message</td>
<td>Contact</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>1047</td>
<td>Attempted to create existing charging session ...</td>
<td>This is an internal error. The OSA Gateway tried to create a new charging session with the same ID as an existing charging session.</td>
<td>Support</td>
</tr>
<tr>
<td>1048</td>
<td>Asked to send a ReservationConfirmationRequest without initial reservation!</td>
<td>This is an internal error. The OSA Gateway tried to confirm a reservation that has never been made.</td>
<td>Support</td>
</tr>
<tr>
<td>1050</td>
<td>Amount confirmed is invalid!</td>
<td>The BE confirmed a reservation with an amount different from the amount it was given by the OSA Gateway.</td>
<td>Support</td>
</tr>
<tr>
<td>1052</td>
<td>Inconsistent reserved amount from BE!</td>
<td>In a SubsequentAmountReservationAck, the Billing Engine reported an inconsistent reserved amount.</td>
<td>Support</td>
</tr>
<tr>
<td>1068</td>
<td>Received unknown error code from IR_NAck(InitialUnitSecondsReservationAck)!</td>
<td>This is an internal error. The OSA Gateway received an error code as part of a IR_Ack message that it doesn’t recognise.</td>
<td>Support</td>
</tr>
<tr>
<td>1069</td>
<td>Received IR_Ack(InitialUnitSecondsReservationAck) in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a IR_Ack from the Billing Engine but the original request was not a InitialUnitSecondsReservationRequest.</td>
<td>Support</td>
</tr>
<tr>
<td>1070</td>
<td>Received IR_Nack(InitialUnitSecondsReservationAck) in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a IR_Nack from the Billing Engine but the original request was not a InitialUnitSecondsReservationRequest.</td>
<td>Support</td>
</tr>
<tr>
<td>1071</td>
<td>Received unknown error code from INER_NAck(InitialUnitNamedEventsReservationAck)!</td>
<td>This is an internal error. The OSA Gateway received an error code as part of a INER_Ack message that it doesn’t recognise.</td>
<td>Support</td>
</tr>
<tr>
<td>1072</td>
<td>Received INER_Ack(InitialUnitNamedEventsReservationAck) in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a INER_Ack from the Billing Engine but the original request was not an InitialUnitNamedEventsReservationRequest.</td>
<td>Support</td>
</tr>
<tr>
<td>Page</td>
<td>Description</td>
<td>Comment</td>
<td>Action</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>1073</td>
<td>Received INER_Nack(InitialUnitNamedEventsReservationNAck) in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received an INER_NAck from the Billing Engine but the original request was not a InitialUnitNamedEventsReservationRequest.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1074</td>
<td>Received unknown error code from SR_NAck(SubsequentUnitSecondsReservationNAck)!</td>
<td>This is an internal error. The OSA Gateway received an error code as part of a SR_NAck message that it doesn’t recognise.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1075</td>
<td>Received SR_Ack(SubsequentUnitSecondsReservationAck) in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a SR_Ack from the Billing Engine but the original request was not a SubsequentUnitSecondsReservationRequest.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1076</td>
<td>Received SR_Nack(SubsequentUnitSecondsReservationNAck) in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a SR_NAck from the Billing Engine but the original request was not a SubsequentUnitSecondsReservationRequest.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1078</td>
<td>Received unknown error code from SNER_Nack(SubsequentUnitNamedEventsReservationNAck)!</td>
<td>This is an internal error. The OSA Gateway received an error code as part of a SNER_NAck message that it doesn’t recognise.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1079</td>
<td>Received SNER_Ack(SubsequentUnitNamedEventsReservationAck) in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a SNER_Ack from the Billing Engine but the original request was not a SubsequentUnitNamedEventsReservationRequest.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1080</td>
<td>Received SNER_Nack(SubsequentUnitNamedEventsReservationNAck) in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a SNER_NAck from the Billing Engine but the original request was not a SubsequentUnitNamedEventsReservationRequest.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1082</td>
<td>Received CR_Ack(UnitSecondsReservationConfirmationAck) in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a CR_Ack from the Billing Engine but the original request was not a UnitSecondsReservationConfirmationRequest.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>#</td>
<td>Description</td>
<td>Explanation</td>
<td>Action</td>
</tr>
<tr>
<td>----</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>1083</td>
<td>Received CNER_Ack(UnitNamedEventsReservationConfirmationAck) in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a CNER_Ack from the Billing Engine but the original request was not a UnitNamedEventsReservationConfirmationRequest.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1084</td>
<td>Received unknown error code from ATC_Nack(DirectDeltaUnitSecondsNAck)!</td>
<td>This is an internal error. The OSA Gateway received an error code as part of a ATC_Nack message that it doesn’t recognise.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1085</td>
<td>Received ATC_Ack(DirectDeltaUnitSecondsAck) in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a ATC_Ack from the Billing Engine but the original request was not a DirectDeltaUnitSecondsRequest.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1086</td>
<td>Received ATC_NAck(DirectDeltaUnitSecondsNAck) in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a ATC_NAck from the Billing Engine but the original request was not a DirectDeltaUnitSecondsRequest.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1088</td>
<td>Received unknown error code from NE_Nack(DirectDeltaUnitNamedEventsNAck)!</td>
<td>This is an internal error. The OSA Gateway received an error code as part of a NE_Nack message that it doesn’t recognise.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1089</td>
<td>Received NE_Ack(DirectDeltaUnitNamedEventsAck) in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a NE_Ack from the Billing Engine but the original request was not a DirectDeltaUnitNamedEventsRequest.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1090</td>
<td>Received NE_Nack(DirectDeltaUnitNamedEventsNAck) in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a NE_Nack from the Billing Engine but the original request was not a DirectDeltaUnitNamedEventsRequest.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1092</td>
<td>Asked to send a UnitSecondsReservationConfirmationRequest without initial reservation!</td>
<td>This is an internal error. The OSA Gateway tried to confirm a reservation that has never been made.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
<td>Resolution</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>------------</td>
<td></td>
</tr>
<tr>
<td>1093</td>
<td>Asked to send a UnitNamedEventsReservationConfirmationRequest without initial reservation!</td>
<td>This is an internal error. The OSA Gateway tried to confirm a reservation that has never been made. Contact Oracle Support</td>
<td></td>
</tr>
<tr>
<td>1094</td>
<td>Received unknown error code from RateUnitSecondsNAck!</td>
<td>This is an internal error. The OSA Gateway received an error code as part of a RateUnitSecondsNAck message that it doesn’t recognise. Contact Oracle Support</td>
<td></td>
</tr>
<tr>
<td>1095</td>
<td>Received RateUnitSecondsNAck in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a RateUnitSecondsNAck from the Billing Engine but the original request was not a RateUnitSecondsRequest. Contact Oracle Support</td>
<td></td>
</tr>
<tr>
<td>1096</td>
<td>Received RateUnitSecondsAck in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a RateUnitSecondsAck from the Billing Engine but the original request was not a RateUnitSecondsRequest. Contact Oracle Support</td>
<td></td>
</tr>
<tr>
<td>1098</td>
<td>Received unknown error code from RateUnitNamedEventsNAck!</td>
<td>This is an internal error. The OSA Gateway received an error code as part of a RateUnitNamedEventsNAck message that it doesn’t recognise. Contact Oracle Support</td>
<td></td>
</tr>
<tr>
<td>1099</td>
<td>Received RateUnitNamedEventsNAck in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a RateUnitNamedEventsNAck from the Billing Engine but the original request was not a RateUnitNamedEventsRequest. Contact Oracle Support</td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td>Received RateUnitNamedEventsAck in response to a ...!</td>
<td>This is an internal error. The OSA Gateway has received a RateUnitNamedEventsAck from the Billing Engine but the original request was not a RateUnitNamedEventsRequest. Contact Oracle Support</td>
<td></td>
</tr>
<tr>
<td>1111</td>
<td>Could not read configuration file</td>
<td>The OSA Gateway was unable to read its configuration file Contact Oracle Support</td>
<td></td>
</tr>
</tbody>
</table>
1113  The configuration file must contain either the beLocationPluginLib option or the billingEngines list to specify the path of a BE location plugin library, or specify a mapping between billing engines ID and billing engines location.  Contact Oracle Support

1114  OSA loading database currency code to currency map check that CCS balance type currency codes are setup correctly.  Balance Type currency was not found.  Check that CCS balance type currency codes are setup correctly in CCS Resources - Balance Type tab. Each cash Balance Type must have Currency One setup using the in the Balance Type screen.

1115  OSA loading CCS balance units found no rows - ensure <string> is replicated correctly - this process is now aborting to produce a core file for analysis.  A table has not been replicated correctly. The following three tables all need to be replicated to all OSA SCP's: CCS_BALANCE_UNIT_SCP CCS_BAL_CURR_ANN_SSCP CCS_CURRENCY_SCP  Contact Oracle Support

2001  Invalid command line option <invalidOption>  The given command line option is not understood. Check startup script invocation entry for errors.  Contact Oracle Support

2013  Caught cmn::Exception: what <string> / reason <string>  An internal error occurred. The process will terminate with error code 2.  Contact Oracle Support

2014  Caught std::exception: what <string>  An internal error occurred. The process will terminate with error code 3.  Contact Oracle Support

2015  Caught something  An internal error occurred. The process will terminate with error code 4.  Contact Oracle Support

Errors

This table defines the Error messages for the OSA Client Interface.

<table>
<thead>
<tr>
<th>Number</th>
<th>Alarm Text</th>
<th>Reason</th>
<th>Remedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>stat(...) failed with ... (...) will use ... instead</td>
<td>The CDR logger could not find the directory specified in the CDRDirectory entry of the configuration file.</td>
<td>It will use a default directory instead.</td>
</tr>
<tr>
<td>Code</td>
<td>Message</td>
<td>Description</td>
<td>Reference</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>1001</td>
<td>... is not a directory, will use ... instead</td>
<td>The path specified in the CDRDirectory entry of the configuration file is actually a file.</td>
<td></td>
</tr>
<tr>
<td>1002</td>
<td>Could not create CDR directory</td>
<td>The CDR logger could not create the directory specified in the CDRDirectory entry of the configuration file. It will use a default directory instead.</td>
<td></td>
</tr>
<tr>
<td>1003</td>
<td>Could not create CDR file!</td>
<td>The CDR logger was unable to create a CDR file. CDRs will be kept in memory until a file can be created.</td>
<td></td>
</tr>
<tr>
<td>1010</td>
<td>Could not invoke reserveAmountRes() on the IpAppChargingSession: ...</td>
<td>The OSA Gateway could not send a CORBA request to the IpAppChargingSession, probably because of a communication failure. Contact Oracle Support</td>
<td></td>
</tr>
<tr>
<td>1012</td>
<td>Could not invoke ...Err() on the IpAppChargingSession: ...</td>
<td>The OSA Gateway could not send a CORBA request to the IpAppChargingSession, probably because of a communication failure. Contact Oracle Support</td>
<td></td>
</tr>
<tr>
<td>1013</td>
<td>Received ChamComsException on session ...: ...</td>
<td>The OSA Gateway received a &quot;communication failure&quot; exception from the BE client interface, meaning that it has lost connection with the Billing Engine. This may be a temporary error.</td>
<td></td>
</tr>
<tr>
<td>1031</td>
<td>Received ChamComsException on assignment ID ...: ...!</td>
<td>The OSA Gateway received a &quot;communication failure&quot; exception from the BE client interface, meaning that it has lost connection with the Billing Engine. This may be a temporary error.</td>
<td></td>
</tr>
<tr>
<td>1035</td>
<td>Could not invoke extendLifeTimeRes() on the IpAppChargingSession: ...</td>
<td>The OSA Gateway could not send a CORBA request to the IpAppChargingSession, probably because of a communication failure. Contact Oracle Support</td>
<td></td>
</tr>
<tr>
<td>1036</td>
<td>Could not invoke directCreditAmountRes() on the IpAppChargingSession: ...</td>
<td>The OSA Gateway could not send a CORBA request to the IpAppChargingSession, probably because of a communication failure. Contact Oracle Support</td>
<td></td>
</tr>
<tr>
<td>Alarm ID</td>
<td>Alarm Description</td>
<td>Description</td>
<td>Resolution</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------</td>
<td>-------------</td>
<td>------------</td>
</tr>
<tr>
<td>1037</td>
<td>Could not invoke directDebitAmountRes() on the IpAppChargingSession: failed with CORBA system exception: ...</td>
<td>The OSA Gateway could not send a CORBA request to the IpAppChargingSession, probably because of a communication failure.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1040</td>
<td>Could not invoke queryBalanceRes() on the IpAppAccountManager: failed with CORBA system exception: ...</td>
<td>The OSA Gateway could not send a CORBA request to the IpAppAccountManager, probably because of a communication failure.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1041</td>
<td>Could not invoke queryBalanceErr() on the IpAppAccountManager: failed with CORBA system exception: ...</td>
<td>The OSA Gateway could not send a CORBA request to the IpAppAccountManager, probably because of a communication failure.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1044</td>
<td>Received 'no context data' error code from SubsequentAmountReservationNack!</td>
<td>The E2BE could not locate the context data for the reservation.</td>
<td>This is a temporary error.</td>
</tr>
<tr>
<td>1046</td>
<td>Could not invoke sessionEnded() on the IpAppChargingSession: failed with CORBA system exception: ...</td>
<td>The OSA Gateway could not send a CORBA request to the IpAppChargingSession, probably because of a communication failure.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1049</td>
<td>Invoked ...Err() with invalid OSA error code ...!</td>
<td>The OSA Gateway invoked a ...Err() method on the IpAppChargingSession with an invalid OSA error code.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1051</td>
<td>Could not invoke sessionAborted() on the IpAppChargingManager: failed with CORBA system exception: ...</td>
<td>The OSA Gateway could not send a CORBA request to the IpAppChargingManager, probably because of a communication failure.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1053</td>
<td>Could not invoke retrieveTransactionHistoryRes() on the IpAppAccountManager: failed with CORBA system exception: ...</td>
<td>The OSA Gateway could not send a CORBA request to the IpAppAccountManager, probably because of a communication failure.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Error Message</td>
<td>Support</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>---------------</td>
<td>---------</td>
</tr>
<tr>
<td>1054</td>
<td>Could not invoke retrieveTransaction HistoryErr() on the IpAppAccountManager: failed with CORBA system exception: ...</td>
<td>The OSA Gateway could not send a CORBA request to the IpAppAccountManager, probably because of a communication failure.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1055</td>
<td>Could not invoke creditUnitRes() on the IpAppAccountManager: failed with CORBA system exception: ...</td>
<td>The OSA Gateway could not send a CORBA request to the IpAppChargingSession, probably because of a communication failure.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1056</td>
<td>Could not invoke creditUnitErr() on the IpAppChargingSession: failed with CORBA system exception: ...</td>
<td>The OSA Gateway could not send a CORBA request to the IpAppChargingSession, probably because of a communication failure.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1057</td>
<td>Could not invoke debitUnitRes() on the IpAppChargingSession: failed with CORBA system exception: ...</td>
<td>The OSA Gateway could not send a CORBA request to the IpAppChargingSession, probably because of a communication failure.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1058</td>
<td>Could not invoke debitUnitErr() on the IpAppChargingSession: failed with CORBA system exception: ...</td>
<td>The OSA Gateway could not send a CORBA request to the IpAppChargingSession, probably because of a communication failure.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1059</td>
<td>Could not invoke directCreditUnitRes() on the IpAppChargingSession: failed with CORBA system exception: ...</td>
<td>The OSA Gateway could not send a CORBA request to the IpAppChargingSession, probably because of a communication failure.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1060</td>
<td>Could not invoke directCreditUnitErr() on the IpAppChargingSession: failed with CORBA system exception: ...</td>
<td>The OSA Gateway could not send a CORBA request to the IpAppChargingSession, probably because of a communication failure.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>Page</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1061</td>
<td>Could not invoke directDebitUnitRes() on the IpAppChargingSession: failed with CORBA system exception: ...</td>
<td>The OSA Gateway could not send a CORBA request to the IpAppChargingSession, probably because of a communication failure.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1062</td>
<td>Could not invoke directDebitUnitErr() on the IpAppChargingSession: failed with CORBA system exception: ...</td>
<td>The OSA Gateway could not send a CORBA request to the IpAppChargingSession, probably because of a communication failure.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1063</td>
<td>Could not invoke rateRes() on the IpAppChargingSession: failed with CORBA system exception: ...</td>
<td>The OSA Gateway could not send a CORBA request to the IpAppChargingSession, probably because of a communication failure.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1064</td>
<td>Could not invoke rateErr() on the IpAppChargingSession: failed with CORBA system exception: ...</td>
<td>The OSA Gateway could not send a CORBA request to the IpAppChargingSession, probably because of a communication failure.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1065</td>
<td>Could not invoke reserveUnitRes() on the IpAppChargingSession: failed with CORBA system exception: ...</td>
<td>The OSA Gateway could not send a CORBA request to the IpAppChargingSession, probably because of a communication failure.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1066</td>
<td>Could not invoke reserveUnitErr() on the IpAppChargingSession: failed with CORBA system exception: ...</td>
<td>The OSA Gateway could not send a CORBA request to the IpAppChargingSession, probably because of a communication failure.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1067</td>
<td>Could not invoke reportNotification() on the IpAppAccountManager: failed with CORBA system exception: ...</td>
<td>The OSA Gateway could not send a CORBA request to the IpAppAccountManager, probably because of a communication failure.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1077</td>
<td>Received 'no context data' error code from SR_Nack(SubsequentUnitSecondsReservationNack)!</td>
<td>The E2BE could not locate the context data for the reservation.</td>
<td>This is a temporary error.</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1081</td>
<td>Received 'no context data' error code from SNER_Nack(SubsequentUnitNamedEventsReservationNAck)!</td>
<td>The E2BE could not locate the context data for the reservation. This is a temporary error.</td>
<td></td>
</tr>
<tr>
<td>1087</td>
<td>Received 'no context data' error code from ATC_NAck!</td>
<td>The E2BE could not locate the context data for the reservation. This is a temporary error.</td>
<td></td>
</tr>
<tr>
<td>1091</td>
<td>Received 'no context data' error code from NE_NAck!</td>
<td>The E2BE could not locate the context data for the reservation. This is a temporary error.</td>
<td></td>
</tr>
<tr>
<td>1097</td>
<td>Received 'no context data' error code from RateUnitSecondsNAck!</td>
<td>The E2BE could not locate the context data for the reservation. This is a temporary error.</td>
<td></td>
</tr>
<tr>
<td>1101</td>
<td>Received 'no context data' error code from RateUnitNamedEventsNAck!</td>
<td>The E2BE could not locate the context data for the reservation. This is a temporary error.</td>
<td></td>
</tr>
<tr>
<td>1102</td>
<td>Received osa::RetrieveNotificationNAck from BE!</td>
<td>A request for triggered notifications resulted in a NAck. Contact Oracle Support</td>
<td></td>
</tr>
<tr>
<td>1103</td>
<td>Timeout waiting for osa::RetrieveNotificationAck</td>
<td>A request for triggered notifications was sent, but no reply was received. Contact Oracle Support</td>
<td></td>
</tr>
<tr>
<td>1104</td>
<td>Invalid eventName X received in reportNotification. Can not send this notification.</td>
<td>A notification was received from the billing engine but the event type was not valid. Contact Oracle Support</td>
<td></td>
</tr>
<tr>
<td>1105</td>
<td>Can't open ccsSSMDispatcher file for write. Aborting SMS send.</td>
<td>Attempting to open the fifo for writing the SMS data has failed. The permissions for this file may be wrong or the file may not exist. The SMS will not be sent. Contact Oracle Support</td>
<td></td>
</tr>
<tr>
<td>1106</td>
<td>The length of the MSISDN exceeds the defined maximum. Aborting SMS send.</td>
<td>Invalid MSISDN length has been determined. The SMS will not be sent. Contact Oracle Support</td>
<td></td>
</tr>
<tr>
<td>1107</td>
<td>The length of the language for the SMS exceeds the defined maximum. Aborting SMS send.</td>
<td>Invalid language length has been determined. The SMS will not be sent. Contact Oracle Support</td>
<td></td>
</tr>
<tr>
<td>1108</td>
<td>The length of the amount field for the SMS exceeds the defined maximum. Aborting SMS send.</td>
<td>Invalid amount length has been determined. The SMS will not be sent. Contact Oracle Support</td>
<td></td>
</tr>
<tr>
<td>Alarms ID</td>
<td>Description</td>
<td>Details</td>
<td>Resolution</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>1109</td>
<td>Can’t write to the ccssSSMDispatcher. Aborting SMS send.</td>
<td>Attempting to write to the fifo used for sending the SMS data has failed.</td>
<td>The permissions for this file may be wrong or the file may not exist. The SMS will not be sent.</td>
</tr>
<tr>
<td>1110</td>
<td>Could not increase statistic for &lt;name&gt;.</td>
<td>The statistic for &lt;name&gt; could not be increased.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>1115</td>
<td>OSA balance Type To Currency Map adding CODE %s, found Duplicate (balance type id %d, acsCustID %d).</td>
<td>When adding entries to the balance type map a duplicate entry has been found.</td>
<td>Contact Oracle support to resolve the Error.</td>
</tr>
<tr>
<td>1300</td>
<td>Could not create remote database link.</td>
<td>The OSA gateway was unable to create a remote link to the SMF database.</td>
<td>Make sure the correct hostname and Oracle SID is specified in the remoteDbLinkData section of the configuration file.</td>
</tr>
<tr>
<td>2002</td>
<td>Select failed with &lt;errorCode&gt; (&lt;errorCodeDescription&gt;)</td>
<td>The c-library function select(), used to poll open file descriptors for activity failed, with the given error.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2004</td>
<td>Caught cmn::Exception: what &lt;errorComponent&gt; / reason &lt;errorDescription&gt;</td>
<td>The given component gave the given error whilst the new configuration was being applied.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2005</td>
<td>Due to errors updating configuration, current state indecisive</td>
<td>The process was unable to roll back to previous configuration following error.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2006</td>
<td>Caught cmn::Exception: what &lt;string&gt; / reason &lt;string&gt;</td>
<td>The given component gave the given error whilst the updated configuration was being parsed. Internal configuration remains unchanged.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2007</td>
<td>Due to errors during parsing, configuration has not been updated</td>
<td>Informational output indicating that the configuration file could not be parsed, and internal configuration remains unchanged.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2050</td>
<td>Failed to send InitialAmountReservationRequest due to invalid ChargingParty &lt;string&gt;</td>
<td>An invalid charging party was detected when attempting to send the given request to the billing engine.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Details</td>
<td>Contact Support</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>2051</td>
<td>Failed to send InitialAmountReservationRequest due to problems with BeClientIF</td>
<td>It was not possible to deliver the given request to either billing engine pair. This indicates an internal logic problem.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2052</td>
<td>Attempt to send SubsequentAmountReservationRequest on non-existant session with id &lt;value&gt;</td>
<td>Program attempted to send billing engine requests &quot;out of order&quot;. This indicates an internal logic problem.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2053</td>
<td>Failed to send SubsequentAmountReservationRequest due to problems with BeClientIF</td>
<td>It was not possible to deliver the given request to either billing engine pair.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2054</td>
<td>Attempt to send AmountReservationConfirmationRequest on non-existant session with id &lt;value&gt;</td>
<td>Program attempted to send billing engine requests &quot;out of order&quot;. This indicates an internal logic problem.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2055</td>
<td>Failed to send AmountReservationConfirmationRequest due to problems with BeClientIF</td>
<td>It was not possible to deliver the given request to either billing engine pair.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2056</td>
<td>Attempt to send ReservationCancellationRequest on non-existant session with id &lt;value&gt;</td>
<td>Program attempted to send billing engine requests &quot;out of order&quot;. This indicates an internal logic problem.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2057</td>
<td>Failed to send ReservationCancellationRequest due to problems with BeClientIF</td>
<td>It was not possible to deliver the given request to either billing engine pair.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2058</td>
<td>Failed to send PerformDirectDeltaRequest due to invalid ChargingParty &lt;string&gt;</td>
<td>An invalid charging party was detected when attempting to send the given request to the billing engine.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2059</td>
<td>Failed to send PerformDirectDeltaRequest due to problems with BeClientIF</td>
<td>It was not possible to deliver the given request to either billing engine pair.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2060</td>
<td>Failed to send QueryAccountRequest due to invalid ChargingParty &lt;string&gt;</td>
<td>An invalid charging party was detected when attempting to send the given request to the billing engine.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>Code</td>
<td>Alarm Description</td>
<td>Description</td>
<td>Contact Information</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>2061</td>
<td>Failed to send QueryAccountRequest due to problems with BeClientIF</td>
<td>It was not possible to deliver the given request to either billing engine pair.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2062</td>
<td>Attempt to abort non-existant session with id of &lt;value&gt;</td>
<td>This indicates an internal logic problem.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2063</td>
<td>Received response message for non-existant session with id of &lt;value&gt;</td>
<td>Unexpected message received from the billing engine, message not related to any current operations.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2064</td>
<td>Received unexpected response message for session with id &lt;value&gt; whilst idle</td>
<td>Billing engine sent unknown response message in reply to request.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2065</td>
<td>Received unknown Ack message for session with id &lt;value&gt;: action &lt;string&gt;, type &lt;string&gt;</td>
<td>Billing engine replied with unknown response message.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2066</td>
<td>Received unknown NAck message for session with id &lt;value&gt;: action &lt;string&gt;, type &lt;string&gt;</td>
<td>Billing engine replied with unknown response message.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2067</td>
<td>Received unknown response message for session with id &lt;value&gt;: action &lt;string&gt;, type &lt;string&gt;</td>
<td>Billing engine replied with unknown response message.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2068</td>
<td>Failed to send queued request due to problems with BeClientIF</td>
<td>Process was unable to send next queued request, following receipt of a billing engine response.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2069</td>
<td>Received com error exception for non-existant session with id of &lt;value&gt;: &lt;string&gt;</td>
<td>Given communication error occurred whilst waiting for response message.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2070</td>
<td>Received com error exception for session with id &lt;value&gt;</td>
<td>Received given communication error, not related to any outstanding request.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2071</td>
<td>Failed to send InitialUnitSecondsReservationRequest due to invalid ChargingParty &lt;string&gt;</td>
<td>An invalid charging party was detected when attempting to send the given request to the billing engine.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>Code</td>
<td>Message</td>
<td>Error Description</td>
<td>Contact Support</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>2072</td>
<td>Failed to send InitialUnitSecondsReservationRequest due to problems with BeClientIF</td>
<td>It was not possible to deliver the given request to either billing engine pair.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2073</td>
<td>Failed to send InitialUnitNamedEventsReservationRequest due to invalid ChargingParty &lt;string&gt;</td>
<td>An invalid charging party was detected when attempting to send the given request to the billing engine.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2074</td>
<td>Failed to send InitialUnitNamedEventsReservationRequest due to invalid NamedEvent &lt;string&gt;</td>
<td>An invalid named event was detected when attempting to send the given request to the billing engine.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2075</td>
<td>Failed to send InitialUnitNamedEventsReservationRequest due to problems with BeClientIF</td>
<td>It was not possible to deliver the given request to either billing engine pair.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2076</td>
<td>Failed to send SubsequentUnitSecondsReservationRequest due to invalid ChargingParty &lt;string&gt;</td>
<td>An invalid charging party was detected when attempting to send the given request to the billing engine.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2077</td>
<td>Failed to send SubsequentUnitSecondsReservationRequest due to problems with BeClientIF</td>
<td>It was not possible to deliver the given request to either billing engine pair.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2078</td>
<td>Failed to send SubsequentUnitNamedEventsReservationRequest due to invalid ChargingParty &lt;string&gt;</td>
<td>An invalid charging party was detected when attempting to send the given request to the billing engine.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2079</td>
<td>Failed to send SubsequentNamedEventsReservationRequest due to problems with BeClientIF</td>
<td>It was not possible to deliver the given request to either billing engine pair.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2080</td>
<td>Attempt to send UnitSecondsReservationConfirmationRequest on non-existant session with id &lt;value&gt;</td>
<td>Program attempted to send billing engine requests “out of order”. This indicates an internal logic problem.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>Alarm ID</td>
<td>Description</td>
<td>Message</td>
<td>Contact Support</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>2081</td>
<td>Failed to send UnitSecondsReservationConfirmationRequest due to invalid confirmedAmount &lt;value&gt;</td>
<td>It was not possible to deliver the given request to either billing engine pair.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2082</td>
<td>Failed to send UnitSecondsReservationConfirmationRequest due to problems with BeClientIF</td>
<td>It was not possible to deliver the given request to either billing engine pair.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2083</td>
<td>Attempt to send UnitNamedEventsReservationConfirmationRequest on non-existent session with id &lt;value&gt;</td>
<td>Program attempted to send billing engine requests &quot;out of order&quot;. This indicates an internal logic problem.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2084</td>
<td>Failed to send UnitNamedEventsReservationConfirmationRequest due to invalid confirmedAmount &lt;value&gt;</td>
<td>It was not possible to deliver the given request to either billing engine pair.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2085</td>
<td>Failed to send UnitNamedEventsReservationConfirmationRequest due to problems with BeClientIF</td>
<td>It was not possible to deliver the given request to either billing engine pair.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2086</td>
<td>Failed to send DirectDeltaUnitSecondsReservationRequest due to invalid ChargingParty &lt;string&gt;</td>
<td>An invalid charging party was detected when attempting to send the given request to the billing engine.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2087</td>
<td>Failed to send DirectDeltaUnitSecondsReservationRequest due to problems with BeClientIF</td>
<td>It was not possible to deliver the given request to either billing engine pair.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2088</td>
<td>Failed to send DirectDeltaUnitNameEventsReservationRequest due to invalid ChargingParty &lt;string&gt;</td>
<td>An invalid charging party was detected when attempting to send the given request to the billing engine.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2089</td>
<td>Failed to send DirectDeltaUnitNameEventsReservationRequest due to invalid NamedEvent &lt;string&gt;</td>
<td>An invalid named event was detected when attempting to send the given request to the billing engine.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>Code</td>
<td>Error Description</td>
<td>Reason</td>
<td>Action</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>2090</td>
<td>Failed to send DirectDeltaUnitName dEventsReservationRequest due to problems with BeClientIF</td>
<td>It was not possible to deliver the given request to either billing engine pair.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2091</td>
<td>Failed to send UnitSecondsRateRequest due to invalid ChargingParty &lt;string&gt;</td>
<td>An invalid charging party was detected when attempting to send the given request to the billing engine.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2092</td>
<td>Failed to send UnitSecondsRateRequest due to problems with BeClientIF</td>
<td>It was not possible to deliver the given request to either billing engine pair.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2093</td>
<td>Failed to send UnitNamedEventsRateRequest due to invalid ChargingParty &lt;string&gt;</td>
<td>An invalid charging party was detected when attempting to send the given request to the billing engine.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2094</td>
<td>Failed to send UnitNamedEventsRateRequest due to invalid NamedEvent &lt;string&gt;</td>
<td>An invalid named event was detected when attempting to send the given request to the billing engine.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2095</td>
<td>Failed to send UnitNamedEventsRateRequest due to problems with BeClientIF</td>
<td>It was not possible to deliver the given request to either billing engine pair.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2102</td>
<td>Asked to decode charging party of unknown type &lt;value&gt;</td>
<td>Client has sent invalid charging party.</td>
<td>Check parameters.</td>
</tr>
<tr>
<td>2103</td>
<td>Charging party &lt;string&gt; doesn't consist of 3 elements</td>
<td>Client has sent invalid charging party.</td>
<td>Check parameters.</td>
</tr>
<tr>
<td>2104</td>
<td>Failed to establish connection to Oracle using login/password of &lt;login&gt;&lt;password&gt;</td>
<td>Failed to connect to Oracle.</td>
<td>Contact Oracle Support</td>
</tr>
<tr>
<td>2150</td>
<td>Attempt to add overlapping range (&lt;value&gt;, &lt;value&gt;) conflicted with existing (&lt;value&gt;, &lt;value&gt;)</td>
<td>Configuration file error. The defined MSISDN ranges contain an overlapping region.</td>
<td>Check configuration file and restart.</td>
</tr>
<tr>
<td>3000</td>
<td>Cannot recreate db link: Not defined in remoteSMF1Name configuration</td>
<td>Runtime, the db link has failed and the remoteSMF1Name configuration item is not set.</td>
<td>Check configuration file and restart.</td>
</tr>
</tbody>
</table>
### Chapter 3

#### System Alarms

<table>
<thead>
<tr>
<th>Number</th>
<th>Alarm Text</th>
<th>Reason</th>
<th>Remedy</th>
</tr>
</thead>
</table>
| 3001   | Cannot recreate link to `<configured remoteSMF1Name>`: Unknown name given in remoteSMF1Name configuration | Runtime, if the db link has failed and the remoteSMF1Name configuration item does not match an entry in tnsnames.ora.  
  - `<configured remoteSMF1Name>` is replaced by the configuration entry. | Check configuration file and restart |
| 3002   | Cannot recreate db link to `<configured remoteSMF1Name>`: `<error code>`: `<error text>` | Runtime, the db link has failed and an SQL error occurs.  
  - `<configured remoteSMF1Name>` is replaced by the configuration entry.  
  - `<error code>` is replaced by the oracle error code.  
  - `<error text>` is replaced by the oracle error text. | Contact Oracle Support |
| 3003   | DB link to SMF failed | Runtime, the db link has failed and could not be re-established even though the configuration remoteSMF1Name is correct | Contact Oracle Support |

### Warnings

This table defines the Warning messages for the OSA Client Interface.

<table>
<thead>
<tr>
<th>Number</th>
<th>Alarm Text</th>
<th>Reason</th>
<th>Remedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1112</td>
<td><code>beLocationPluginLib</code> option in configuration file will be ignored</td>
<td>This is because the configuration file also contains a <code>billingEngines</code> list, which will take precedence.</td>
<td>No action required</td>
</tr>
</tbody>
</table>

### Notices

This table defines the Notice messages for the OSA Client Interface.

<table>
<thead>
<tr>
<th>Number</th>
<th>Alarm Text</th>
<th>Reason</th>
<th>Remedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1042</td>
<td>Can write CDR files again</td>
<td>The OSA Gateway is able to create CDR files again.</td>
<td>No action required</td>
</tr>
<tr>
<td>Code</td>
<td>Message</td>
<td>Action</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------------------------------------------------------</td>
<td>-----------------</td>
<td></td>
</tr>
<tr>
<td>1116</td>
<td>remoteSMF1Name not set in configuration. Recovery from dropped DB link will not be possible</td>
<td>On startup, the remoteSMF1Name configuration item is not set. No action required</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>Exiting with status &lt;exitCode&gt;</td>
<td>The osaChamScs process is about to exit, returning the given status code &lt;value&gt;. No action required</td>
<td></td>
</tr>
<tr>
<td>2003</td>
<td>Rereading config, due to receipt of SIGHUP</td>
<td>Informational output indicating the configuration file will be reread. No action required</td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td>Turning debug on, due to receipt of SIGUSR1</td>
<td>Informational output indicating debugging information has been turned on. No action required</td>
<td></td>
</tr>
<tr>
<td>2009</td>
<td>Send SIGUSR2 to turn debug off</td>
<td>Informational output describing how to turn debugging information off. No action required</td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td>Turning debug off, due to receipt of SIGUSR2</td>
<td>Informational output indicating debugging information has been turned off. No action required</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td>Shutting down when idle, due to receipt of SIGUSR2</td>
<td>Informational output indicating process will now refuse new connections, and will shutdown when existing operations have finished. No action required</td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td>Process is not idle, about to exit</td>
<td>Informational output indicating all existing operations have finished, and process is shutting down. No action required</td>
<td></td>
</tr>
</tbody>
</table>
Overview

Introduction

This chapter explains the background processes.

In this chapter

This chapter contains the following topics.

osaServerNotificationHandlers  71
osaVWARSQueryHandlers  72
osaVWARSNotificationPlugins  72
libnotificationQM  72
osaChamScs  73
runOsaChamScs  73
ccsSSMDispatcher  73
signalOsaChamScs  73
OSA CHAM SCS Gateway  74
Billing Engine EDRs  74
Statistics  76
Alarms  79

osaServerNotificationHandlers

Purpose

The osaServerNotificationHandlers library provides the OSA gateway specific notification functionality on the UBE. The OSA notifications are specific to the OSA gateway and are not related to CCS real-time or short message notifications.

Location

This binary is located on VWSs.

Configuration

This binary has no specific configuration.
osaVWARSQueryHandlers

Purpose
The osaVWARSQueryHandlers library provides the OSA gateway query functionality on the UBE, returning one of an individual specified balance type, all the balance types in a specified balance type cascade, or all the balance types in the subscriber's wallet; for any balance type returned the sum of all buckets and the earliest expiring bucket will be returned.

Location
This binary is located on VWSs.

Configuration
This binary has no specific configuration.

osaVWARSNotificationPlugins

Purpose
The osaVWARSNotificationPlugins library provides the OSA gateway specific notification functionality on the UBE. The OSA notifications are specific to the OSA gateway and are not related to CCS real-time or short message notifications.

Location
This binary is located on VWSs.

Configuration
This binary has no specific configuration.

libnotificationQM

Purpose
The libnotificationQM library provides the OSA gateway specific notification functionality on the UBE. The OSA notifications are specific to the OSA gateway and are not related to CCS real-time or short message notifications.

Location
This binary is located on VWSs.

Configuration
This binary has no specific configuration.
osaChamScs

Purpose
The osaChamScs binary provides the OSA gateway functionality on the SCS.

Location
This binary is located on the SCS.

runOsaChamScs

Purpose
The runOsaChamScs shell script is used to execute the gateway. See Execution (on page 224).

Location
This binary is located on the SCS.

ccsSSMDDispatcher

Purpose
The ccsSSMDDispatcher shell script executes ccsSSMDispetcher binary on the machine. Refer to the CCS Technical Guide for details of this binary.

Location
This binary is located on the SCS.

signalOsaChamScs

Purpose
The signalOsaChamScs shell script provides the following:

```bash
usage() {
    echo "usage: `basename $0` ( int | term | usr1 | usr2 | hup | kill | abrt)"
    echo "  hup - reread config"
    echo "  usr1 - turn debug on"
    echo "  usr2 - turn debug off"
    echo "  int - shutdown cleanly"
    echo "  term - shutdown cleanly"
    echo "  kill - end process immediately (cannot be trapped)"
    echo "  abrt - abort and produce core"
}
```
OSA CHAM SCS Gateway

Events

OSA CHAM Gateway EDRs are generated on the SCS for any of the following events:

- Charging session is created. (Type 20)
- Reservation is confirmed. (Type 21)
- Reservation is canceled. (Type 22)
- Direct debit/credit is invoked. (Type 23)

Refer to the Event Detail Record Guide for a description of OSA CHAM SCS Gateway EDRs.

Example EDRs

Here are some example OSA EDRs.

- Example amount-based confirmation OSA gateway EDR:
  21|Session=New ChargingSession|Merchant=Merchant Id/1|
  Account=912233289900/Personal/General Cash|ReservedAmount=50|
  ConfirmedAmount=20|Reservation=20031106133119|
  Confirmed=20031106133147|RES=50;LIF=180;DEB=20|
  ApplicationDescription=ReserveAmountReq:DebitAmountReq|
  UnitType=amount|DiscountOverride=0|N

- For amount-based direct charges an EDR will always be written on the SCS. No Billing Engine Rate request is required as the amount is already known. Example direct debit/credit amount-based OSA Gateway EDR:
  23|Session=New ChargingSession|Merchant=Merchant Id/1|
  Account=912233289900/Personal|Amount=-10|Postpaid|
  Time=20031106133230|ApplicationDescription=DirectDebitAmountReq|
  UnitType=amount|DiscountOverride=0|N

- Example direct debit/credit unit seconds-based OSA gateway EDR:
  23|Session=New ChargingSession|Merchant=Merchant Id/1|
  Account=912233289900/Personal|Amount=-10|Postpaid|
  Time=20031106133230|ApplicationDescription=DirectDebitUnitReq|
  UnitType=tariff.91.44|DiscountOverride=0|N

Billing Engine EDRs

Creation

Billing Engine EDRs are written whenever an amount or an expiry date changes on a user’s Account (for a specific Wallet and Balance Type).

Refer to the Event Detail Record Guide for a description of Billing Engine EDRs.

Events

Billing Engine EDRs are generated for any of the following events:

- An amount-based reservation is confirmed. (Type 21)
- An amount-based direct debit/credit is made. As this is the means through which Wallet type and Balance Type expiry dates are also changed, a Billing Engine EDR will also be created for this transaction. (Type 23)
- A unit seconds-based reservation is confirmed. (Type 24)
A unit Named Event-based reservation is confirmed. (Type 25)
A unit seconds-based direct debit/credit is made. (Type 26)
A unit Named Event-based direct debit/credit is made. (Type 27)

Example EDRs

Here are some example BE EDRs.

- **Example amount-based reservation BE EDR:**
  
  | BILLING_ENGINE_ID=1 | SCP_ID=198248872 | SEQUENCE_NUMBER=4494 | CDR_TYPE=21 |
  | RECORD_DATE=20031119140101 | ACCT_ID=2146 | ACCT_REF_ID=2123 | WALLET_TYPE=23 |
  | BALANCE_TYPE=49 | BALANCES=9270 | COSTS=10 |
  | APPLICATION_DESC=ReserveAmountReq:DebitAmountReq:ReserveAmountReq(subsequent)

- **Example amount-based direct debit/credit BE EDR:**
  
  | BILLING_ENGINE_ID=1 | SCP_ID=198248872 | SEQUENCE_NUMBER=4495 | CDR_TYPE=23 |
  | RECORD_DATE=20031119140115 | ACCT_ID=2146 | ACCT_REF_ID=2123 | WALLET_TYPE=23 |
  | BALANCE_TYPE=49 | BALANCES=9260 | COSTS=-10 |
  | APPLICATION_DESC=DirectDebitAmountReq | OLD_BALANCE_EXPIRIES=0 | NEW_BALANCE_EXPIRIES=0 | BONUS_TYPE=CUSTOM | REFERENCE=OSA Bonus

- **Example unit-seCONDS based reservation BE EDR:**
  
  | BILLING_ENGINE_ID=1 | SCP_ID=198248872 | SEQUENCE_NUMBER=4572 | CDR_TYPE=24 |
  | RECORD_DATE=20031119155328 | ACCT_ID=2146 | ACCT_REF_ID=2123 | CLI=01473 | ACS_CUST_ID=22 |
  | BALANCE_TYPES=49 | BALANCES=60 | COSTS=83 | ACCOUNT_TYPE=22 | CASCADE_ID=21 |
  | RATES=20,20,20,20,20,10,10,10,10,10 | LENGTHS=60,60,60,60,60,60,60,60,60,60,60,60,-1 |
  | DISCOUNTS=250000,250000,250000,250000,250000,250000,250000,250000,250000,250000,250000,500000 |
  | MAX_CHARGE=-1 | DURATION=360 | TN=01394 | TCS=20031119154923 | TCE=20031119154423 | CS=S |
  | DISCOUNT_TYPE=CUMULATIVE | APPLICATION_DESC=ReserveUnit(Seconds)Req:DebitUnitReq

- **Example unit-named event based reservation BE EDR:**
  
  | BILLING_ENGINE_ID=1 | SCP_ID=198248872 | SEQUENCE_NUMBER=4487 | CDR_TYPE=25 |
  | RECORD_DATE=20031119135502 | ACCT_ID=2146 | ACCT_REF_ID=2123 |
  | APPLICATION_DESC=ReserveUnit(NamedEvents)Req:DebitUnitReq:CREDITUnitReq |
  | ACS_CUST_ID=22 | CS=S | TCS=20031119135121 | BALANCE_TYPES=49 | BALANCES=8985 | COSTS=700 |
  | ACCOUNT_TYPE=22 | EVENT_CLASS=OSA2BE | EVENT_NAME=event1 | EVENT_COST=100 |
  | EVENT_COUNT=7 | DISCOUNT=0 | CASCADE=21

- **Example unit-seCONDS based direct credit/debit BE EDR:**
  
  | BILLING_ENGINE_ID=1 | SCP_ID=198248872 | SEQUENCE_NUMBER=4490 | CDR_TYPE=26 |
  | RECORD_DATE=20031119135814 | ACCT_ID=2146 | ACCT_REF_ID=2123 | CLI=01473 |
  | ACS_CUST_ID=22 | BALANCE_TYPES=49 | BALANCES=8285 | COSTS=15 | ACCOUNT_TYPE=22 |
  | CASCADE_ID=21 | RATES=20,20,20,20,20,10,10,10,10,10 |
  | LENGTHS=60,60,60,60,60,60,60,60,60,60,60,60,-1 |
  | DISCOUNTS=250000,250000,250000,250000,250000,250000,250000,250000,250000,500000 |
  | MAX_CHARGE=100000 | DURATION=10 | TN=01394 | TCS=20031119135432 | TCE=20031119135442 |
  | CS=S | DISCOUNT_TYPE=CUMULATIVE | APPLICATION_DESC=DirectCreditUnit(Seconds)Req

- **Example unit-named event based direct credit/debit BE EDR:**
Statistics

Introduction

The OSA gateway generates statistics for every defined and supported CORBA method.

The statistics are a simple count of each method, named the same as the method they are monitoring.

The OSA gateway uses the standard Oracle statistics functions provided by the smsStatsDaemon.

Refer to the SMS Technical Guide for details.

The statistics can be replicated to the SMS at a configurable rate.
Chapter 4

Diagram

Here is the Statistics process.

List of statistics

Here is a list of the statistics produced by OSA CHAM processes.

- setCallback (Account Manager)
- createNotification
- destroyNotification
- queryBalanceReq
- changeNotification
- getNotification
- retrieveTransactionHistoryReq
- createChargingSession
- creditAmountReq
- creditUnitReq
- debitAmountReq
Chapter 4

- debitUnitReq
- directCreditAmountReq
- directCreditUnitReq
- directDebitAmountReq
- directDebitUnitReq
- extendLifeTimeReq
- getAmountLeft
- getLifeTimeLeft
- getUnitLeft
- rateReq
- release
- reserveAmountReq
- reserveUnitReq
- reportNotification
- queryBalanceRes
- queryBalanceErr
- retrieveTransactionHistoryRes
- retrieveTransactionHistoryErr
- sessionAborted
- creditAmountErr
- creditAmountRes
- creditUnitErr
- creditUnitRes
- debitAmountErr
- debitAmountRes
- debitUnitErr
- debitUnitRes
- directCreditAmountErr
- directCreditAmountRes
- directCreditUnitErr
- directCreditUnitRes
- directDebitAmountErr
- directDebitAmountRes
- directDebitUnitErr
- directDebitUnitRes
- extendLifeTimeErr
- extendLifeTimeRes
- rateErr
- rateRes
- reserveAmountErr
- reserveAmountRes
- reserveUnitErr
- reserveUnitRes
- sessionEnded
Alarms

Introduction
The OSA gateway generates alarms. The OSA gateway writes them to the system log on the target platform:
/var/adm/messages

Target location
The OSA gateway alarms are logged to a configurable location on the target platform such as the following:
/IN/service_packages/OSA/tmp/runOsaChamScs.log

Functionality
The OSA gateway uses the standard Oracle alarm functionality provided by the smsAlarmDaemon, a process that parses the syslog in /var/adm/messages and replicates these alarms to the smsMaster on the SMS. The alarms can be replicated to the SMS at a configurable rate.

Refer to the SMS Technical Guide for details.
Diagram

Here is a diagram of the Alarms process.

---

**Diagram:**

Here is a diagram of the Alarms process.

[Diagram of the Alarms process is shown here.]
Chapter 5

Tools and Utilities

Overview

Introduction

This chapter explains the tools used in OSA Gateway.

In this chapter

This chapter contains the following topics.

Running the OSA Gateway 81
Running the Test Client Application 82
signalOsaChamScs.sh 82

Running the OSA Gateway

Starting up a gateway instance

Follow these steps to start up a gateway instance.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>As osa_oper on the SCS type: /IN/service_packages/OSA/bin/runOsaChamScsStartup.sh</td>
</tr>
<tr>
<td>2</td>
<td>Check the /IN/service_packages/OSA/tmp/runOsaChamScs.log on the SCS, to ensure no errors are reported and that the OSA gateway is accepting connections.</td>
</tr>
<tr>
<td>3</td>
<td>The OSA gateway will generate CORBA IOR location files during startup. These must be copied onto the client machine to allow the two CORBA objects (Client and Gateway) to communicate.</td>
</tr>
</tbody>
</table>

Stopping a gateway instance

Follow these steps to terminate all instances of the OSA gateway currently running.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>As osa_oper on the SCS type: /IN/service_packages/OSA/bin/signalOsaChamScs.sh INT</td>
</tr>
</tbody>
</table>
Running the Test Client Application

Starting up a client instance

Follow these steps to start up a client instance.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Go to the installation directory for the osaChamClient package on the host machine.</td>
</tr>
<tr>
<td>2</td>
<td>When in osaChamClient_v2.X.X\bin, double click on runOsaChamClient.bat.</td>
</tr>
</tbody>
</table>

*Note:* The client will fail to start if there are any errors while reading the CORBA IOR files or the configuration file.

Stopping the client instance

To stop the client instance, double click *Exit* in any panel.

signalOsaChamScs.sh

**Purpose**

signalOsaChamScs.sh enables you to send instructions to all the osaChamScs processes on a host.

**Configuration**

signalOsaChamScs.sh supports the following command line parameters:

```
signalOsaChamScs.sh <int | term | usr1 | usr2 | hup | kill | abrt>
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>Shutdown cleanly</td>
</tr>
<tr>
<td>term</td>
<td>Shutdown cleanly</td>
</tr>
<tr>
<td>usr1</td>
<td>Turn debug on</td>
</tr>
<tr>
<td>usr2</td>
<td>Turn debug off</td>
</tr>
<tr>
<td>hup</td>
<td>Reread config</td>
</tr>
<tr>
<td>kill</td>
<td>End process immediately (cannot be trapped).</td>
</tr>
<tr>
<td>abrt</td>
<td>Abort and produce core file.</td>
</tr>
</tbody>
</table>
Overview

Introduction

This section is intended to give examples of the OSA CHAM SCS gateway and third-party client application usage through providing example message flows for a small selection of possible transactions.

In this chapter

This chapter contains the following topics.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Balance/Account Info Queries</td>
<td>83</td>
</tr>
<tr>
<td>Creation of Charging Session</td>
<td>84</td>
</tr>
<tr>
<td>Transaction History</td>
<td>84</td>
</tr>
<tr>
<td>Notifications</td>
<td>85</td>
</tr>
<tr>
<td>Amount Based Reservations</td>
<td>86</td>
</tr>
<tr>
<td>Unit Based Reservations (Named Events)</td>
<td>91</td>
</tr>
</tbody>
</table>

Balance/Account Info Queries

Success

![Message Flow Diagram]

Message Details

<table>
<thead>
<tr>
<th>Step</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The IpAppAccountManager invokes a queryBalanceReq() on the IpAccountManager containing 1 or more user account details.</td>
</tr>
<tr>
<td>2</td>
<td>After a successful query on the billing engine the queryBalanceRes() method is invoked on the IpAppAccountManager containing result details.</td>
</tr>
</tbody>
</table>
### Failure

<table>
<thead>
<tr>
<th>Step</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The IpAppAccountManager invokes a queryBalanceReq() on the IpAccountManager containing 1 or more user account details.</td>
</tr>
<tr>
<td>2</td>
<td>After an unsuccessful query on the billing engine the queryBalanceRes() method is invoked on the IpAppAccountManager containing details of the specific error which occurred.</td>
</tr>
</tbody>
</table>

### Creation of Charging Session

**Message Flow**

1. The IpAppChargingManager invokes a createChargingSession() on the IpChargingManager. The IpChargingManager creates an instance of the IpChargingSession object and returns an object reference for the new object.

**Message Details**

<table>
<thead>
<tr>
<th>Step</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The IpAppChargingManager invokes a createChargingSession() on the IpChargingManager. The IpChargingManager creates an instance of the IpChargingSession object and returns an object reference for the new object.</td>
</tr>
</tbody>
</table>

### Transaction History

**Transaction History Retrieval**

1. retrieveTransactionHistoryReq()
### Message Details

<table>
<thead>
<tr>
<th>Step</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The IpAppAccountManager invokes a retrieveTransactionHistoryReq() on the IpAccountManager containing 1 or more user account details.</td>
</tr>
<tr>
<td>2</td>
<td>After a successful query on the SMS the retrieveTransactionHistoryRes() method is invoked on the IpAppAccountManager containing result details.</td>
</tr>
</tbody>
</table>

### Notifications

#### Message Flow

**IpAppAccountManager**

1. createNotification()
2. reportNotification()
3. getNotification()
4. changeNotification()
5. reportNotification()
6. destroyNotification()

**IpAccountManager**

The IpAppAccountManager invokes a createNotification() message on the IpAccountManager for an account, asking for notification of a low credit.

2. The charging event criteria is met and the IpAppAccountManager is informed.
3. The IpAppAccountManager retrieves information on current event criteria set.
4. The IpAppAccountManager changes the event criteria for the account (for example: asking for zero credit).
5. The charging event criteria is met and the IpAppAccountManager is informed.
6. The IpAppAccountManager is no longer interested in this account and removes its event criteria.
Amount Based Reservations

Single Reservations with Confirmation

1. At the beginning of this scenario the subscriber has a balance of 100 billing engine monetary units. The IpAppChargingSession invokes a reserveAmountReq() on the IpChargingSession containing details of the account to reserve from, the wallet/balance type, the amount to reserve (20 BE units), any discount to be applied, and the lifetime of the reservation. The subscriber’s account balance will remain at 100 billing engine monetary units, however there will only be 80 billing engine monetary units available for alternative reservations.

2. The IpAppChargingSession object will query the billing engine and in this instance successfully reserves the requested amount. It informs the IpAppChargingSession object by calling the reserveAmountRes() method.

3. The application wants to debit and confirm the use of 10 of the reserved BE units against the subscriber’s account. It uses the debitAmountReq() method (with the closeReservation flag set to ‘true’). The current reservation is now closed and the subscriber’s account would be reduced by the effective cost of 10 BE units. The 10 unconfirmed BE units, those which were initially reserved but not confirmed, would be available for other transactions. The subscriber’s balance will now be equivalent to the cost of 90 billing engine monetary units.

4. The request is successful and the application is informed with the debitAmountRes() method.

<table>
<thead>
<tr>
<th>Step</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>At the beginning of this scenario the subscriber has a balance of 100 billing engine monetary units. The IpAppChargingSession invokes a reserveAmountReq() on the IpChargingSession containing details of the account to reserve from, the wallet/balance type, the amount to reserve (20 BE units), any discount to be applied, and the lifetime of the reservation. The subscriber’s account balance will remain at 100 billing engine monetary units, however there will only be 80 billing engine monetary units available for alternative reservations.</td>
</tr>
<tr>
<td>2</td>
<td>The IpAppChargingSession object will query the billing engine and in this instance successfully reserves the requested amount. It informs the IpAppChargingSession object by calling the reserveAmountRes() method.</td>
</tr>
<tr>
<td>3</td>
<td>The application wants to debit and confirm the use of 10 of the reserved BE units against the subscriber’s account. It uses the debitAmountReq() method (with the closeReservation flag set to ‘true’). The current reservation is now closed and the subscriber’s account would be reduced by the effective cost of 10 BE units. The 10 unconfirmed BE units, those which were initially reserved but not confirmed, would be available for other transactions. The subscriber’s balance will now be equivalent to the cost of 90 billing engine monetary units.</td>
</tr>
<tr>
<td>4</td>
<td>The request is successful and the application is informed with the debitAmountRes() method.</td>
</tr>
</tbody>
</table>
Multiple Reservations With Confirmation

1. At the beginning of this scenario the subscriber has a balance of 100 billing engine monetary units. The IpAppChargingSession invokes a reserveAmountReq() on the IpChargingSession containing details of the account to reserve from, the wallet/balance type, the amount to reserve (20 BE units), any discount to be applied, and the lifetime of the reservation. The subscriber’s account balance will remain at 100 billing engine monetary units, however there will only be 80 billing engine monetary units available for alternative reservations.

2. The IpAppChargingSession object will query the billing engine and in this instance successfully reserves the requested amount. It informs the IpAppChargingSession object by calling the reserveAmountRes() method.

3. The application wants to debit the use of 10 of the reserved BE units against the reserved BE units on the OSA CHAM SCS gateway. It uses the debitAmountReq() method (with the closeReservation flag set to ‘false’). The subscriber’s account balance of 100 billing engine monetary units. The current reservation on the OSA CHAM SCS gateway now has 10 reserved BE units, and 10 BE units waiting to be confirmed through the subscriber sending a release() method (or using the closeReservation flag of a subsequent debit(credit)UnitReq() method).

4. The reservation alteration is successful – the debitAmountRes() method is used to inform the application.
The application needs a subsequent reservation and uses the reserveAmountReq() method to request 10 more BE units. The subscriber’s account balance of 100 billing engine monetary units, however, there will only be the equivalent cost of 70 BE units available for alternative transactions once this transaction has completed. These unreserved units in the user’s account are available for alternative reservations from this or other third-party client applications. The current reservation on the OSA CHAM SCS gateway will now have 20 reserved BE units, and 10 BE units waiting to be confirmed through the subscriber sending a release() method (or using the closeReservation flag of a subsequent debit(credit)AmountReq() method).

The IpAppChargingSession object will query the billing engine and in this instance successfully reserves the requested amount. It informs the IpAppChargingSession object by calling the reserveAmountRes() method.

The application wants to credit some from the debited amount back to the reservation held on the OSA CHAM SCS gateway. It invokes the creditAmountReq() method for 5 BE units. The subscriber’s account balance of 100 billing engine monetary units. The current reservation on the OSA CHAM SCS gateway now has 25 reserved BE units, and 5 BE units waiting to be confirmed through the subscriber sending a release() method (or using the closeReservation flag of a subsequent debit(credit)AmountReq() method)

If no more messages were received before the session times out, the subscriber’s balance on the billing engine would not be changed.

The application uses the debitAmountReq() method to debit and confirm an additional 2 BE units against the reservation and have these units confirmed on the billing engine. The debitAmountReq() method is invoked with the closeReservation flag to ‘true’. The reservation is confirmed (7 BE units) through the IpChargingSession informing the billing engine. The current reservation is now closed, and the subscriber’s account would be reduced by 7 billing engine monetary units. The 23 unconfirmed BE units, those that were initially reserved but not confirmed, would be available for other reservations. The subscriber will now have a balance of 93 billing engine monetary units.

The request is successful and the application is informed with the debitUnitRes() method.

Note: This charging session may now be used for subsequent reservations.

Reservation Released
Messages

<table>
<thead>
<tr>
<th>Step</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>At the beginning of this scenario the subscriber has a balance of 100 billing engine monetary units. The IpAppChargingSession invokes a reserveAmountReq() on the IpChargingSession containing details of the account to reserve from, the wallet/balance type, the amount to reserve (10 BE units), any discount to be applied, and the lifetime of the reservation.</td>
</tr>
<tr>
<td>2</td>
<td>The IpAppChargingSession object will query the billing engine and in this instance successfully reserves the requested amount. It informs the IpAppChargingSession object by calling the reserveAmountRes() method.</td>
</tr>
<tr>
<td>3</td>
<td>The application cancels the reservation using the release() method. The IpAppChargingSession does not wait for a confirmation. The charging session is now closed and cannot be used for any other transactions.</td>
</tr>
</tbody>
</table>

Direct Debit

 Messages

<table>
<thead>
<tr>
<th>Step</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>At the beginning of this scenario the subscriber has a balance of 100 billing engine monetary units. The IpAppChargingSession invokes a directDebitAmountReq() on the IpChargingSession containing details of the account to reserve from, the wallet/balance type, the amount to reserve (10 BE units), any discount to be applied. The IpChargingSession object will attempt to debit the account on the billing engine and in this instance it is successful.</td>
</tr>
<tr>
<td>2</td>
<td>The IpAppChargingSession object will query the billing engine and in this instance successfully debits the requested amount. It informs the IpAppChargingSession object by calling the directDebitAmountRes() method. The subscriber will now have a balance of 90 billing engine monetary units.</td>
</tr>
<tr>
<td>3</td>
<td>At the beginning of this scenario the subscriber has a balance of 100 billing engine monetary units. The IpAppChargingSession invokes a directDebitAmountReq() on the IpChargingSession containing details of the account to reserve from, the wallet/balance type, the amount to reserve (10 BE units), any discount to be applied, and the lifetime of the reservation. The IpChargingSession object will attempt to debit the account on the billing engine and in this instance it is unsuccessful.</td>
</tr>
<tr>
<td>4</td>
<td>The IpAppChargingSession object will query the billing engine and in this instance fails to debit the requested amount. It informs the IpAppChargingSession object by calling the directDebitAmountErr() method with the appropriate error cause. The subscriber will still have a balance of 100 billing engine monetary units.</td>
</tr>
</tbody>
</table>

Note: This transaction did not invoke a reservation on this charging session.
Direct Credit

1. At the beginning of this scenario the subscriber has a balance of 100 billing engine monetary units. The IpAppChargingSession invokes a directCreditAmountReq() on the IpChargingSession containing details of the account to reserve from, the wallet/balance type, the amount to reserve (10 BE units), any discount to be applied. The IpChargingSession object will attempt to credit the account on the billing engine and in this instance it is successful.

2. The IpAppChargingSession object will query the billing engine and in this instance successfully credits the requested amount. It informs the IpAppChargingSession object by calling the directCreditAmountRes() method. The subscriber will now have a balance of 110 billing engine monetary units.

3. At the beginning of this scenario the subscriber has a balance of 100 billing engine monetary units. The IpAppChargingSession invokes a directCreditAmountReq() on the IpChargingSession containing details of the account to reserve from, the wallet/balance type, the amount to reserve (10 BE units), any discount to be applied, and the lifetime of the reservation. The IpChargingSession object will attempt to credit the account on the billing engine and in this instance it is unsuccessful.

4. The IpAppChargingSession object will query the billing engine and in this instance fails to debit the requested amount. It informs the IpAppChargingSession object by calling the directCreditAmountErr() method with the appropriate error cause. The subscriber will still have a balance of 100 billing engine monetary units.

Note: This transaction did not invoke a reservation on this charging session.
Unit Based Reservations (Named Events)

Single Reservation with Confirmation

<table>
<thead>
<tr>
<th>Step</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>At the beginning of this scenario the subscriber has a balance equivalent to the cost of 100 named events. The IpAppChargingSession invokes a reserveUnitReq() on the IpChargingSession containing details of the account to reserve from, the wallet/balance type, the amount to reserve (20 named events), any discount to be applied, the event class and event name to be used, and the lifetime of the reservation. The subscriber’s account balance will remain equivalent to the cost of 100 named events, however there will only be the equivalent cost of 80 named events available for alternative reservations.</td>
</tr>
<tr>
<td>2</td>
<td>The IpAppChargingSession object will query the billing engine and in this instance successfully reserves the requested number of named events. It informs the IpAppChargingSession object by calling the reserveUnitRes() method.</td>
</tr>
<tr>
<td>3</td>
<td>The application wants to debit and confirm the use of 10 of the reserved named events against the subscriber’s account. It uses the debitUnitReq() method (with the closeReservation flag set to ‘true’). The current reservation is now closed and the subscriber’s account would be reduced by the effective cost of 10 named events. The 10 unconfirmed named events, those that were initially reserved but not confirmed, would be available for other transactions. The subscriber’s balance will now be equivalent to the cost of 90 named events.</td>
</tr>
<tr>
<td>4</td>
<td>The request is successful and the application is informed with the debitUnitRes() method.</td>
</tr>
</tbody>
</table>
Multiple Reservations With Confirmation

1. At the beginning of this scenario the subscriber has a balance equivalent to the cost of 100 named events. The IpAppChargingSession invokes a reserveUnitReq() on the IpChargingSession containing details of the account to reserve from, the wallet/balance type, the amount to reserve (20 named events), any discount to be applied, the event class and event name to be used, and the lifetime of the reservation. The subscriber’s account balance will remain equivalent to the cost of 100 named events, however there will only be the equivalent cost of 80 named events available for alternative reservations.

2. The IpAppChargingSession object will query the billing engine and in this instance successfully reserves the requested number of named events. It informs the IpAppChargingSession object by calling the reserveUnitRes() method.

3. The application wants to debit the use of 10 of the reserved named events against the reserved named events. It uses the debitUnitReq() method (with the closeReservation flag set to ‘false’). The subscriber’s account balance will remain equivalent to the cost of 100 named events. The current reservation on the OSA CHAM SCS gateway now has 10 reserved named events, and 10 named events waiting to be confirmed through the subscriber sending a release() method (or using the closeReservation flag of a subsequent debit(credit)UnitReq() method).

4. The reservation alteration is successful – the debitUnitRes() method is used to inform the application.
<table>
<thead>
<tr>
<th>Step</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>The application needs a subsequent reservation and uses the reserveUnitReq() method to request 10 more named events. The subscriber's account balance will remain equivalent to the cost of 100 named events, however, there will only be the equivalent cost of 70 named events available for alternative transactions once this transaction has completed. These unreserved units in the user's account are available for alternative reservations from this or other third-party client applications. The current reservation on the OSA CHAM SCS gateway will now have 20 reserved named events, and 10 named events waiting to be confirmed through the subscriber sending a release() method (or using the closeReservation flag of a subsequent debit(credit)UnitReq() method).</td>
</tr>
<tr>
<td>6</td>
<td>The IpAppChargingSession object will query the billing engine and in this instance successfully reserves the requested amount. It informs the IpAppChargingSession object by calling the reserveUnitRes() method.</td>
</tr>
<tr>
<td>7</td>
<td>The application wants to credit some from the debited amount back to the reservation held on the OSA CHAM SCS gateway. It invokes the creditUnitReq() method for 5 named events. The subscriber's account balance will remain equivalent to the cost of 100 named events. The current reservation on the OSA CHAM SCS gateway now has 25 reserved named events, and 5 named events waiting to be confirmed through the subscriber sending a release() method (or using the closeReservation flag of a subsequent debit(credit)UnitReq() method). If no more messages were received before the session times out, the subscriber's balance on the billing engine would not be changed.</td>
</tr>
<tr>
<td>8</td>
<td>The reservation alteration is successful – the creditUnitRes() method is used to inform the application.</td>
</tr>
<tr>
<td>9</td>
<td>The application uses the debitUnitReq() method to debit and confirm an additional 2 named events against the reservation and have these named events confirmed on the billing engine. The debitUnitReq() method is invoked with the closeReservation flag to 'true'. The reservation is confirmed (7 named events) through the IpChargingSession informing the billing engine. The current reservation is now closed and the subscriber's account would be reduced by the effective cost of 7 named events. The 23 unconfirmed named events, those that were initially reserved but not confirmed, would be available for other reservations. The subscriber will now have a balance equivalent to 93 named events.</td>
</tr>
<tr>
<td>10</td>
<td>The request is successful and the application is informed with the debitUnitRes() method.</td>
</tr>
</tbody>
</table>

**Note:** This charging session may now be used for subsequent reservations.

**Reservation Released**

![Diagram of reservation process](image-url)
Messages

<table>
<thead>
<tr>
<th>Step</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>At the beginning of this scenario the subscriber has a balance equivalent to the cost of 100 named events. The IpAppChargingSession invokes a <code>reserveUnitReq()</code> on the IpChargingSession containing details of the account to reserve from, the wallet/balance type, the amount to reserve (10 named events), any discount to be applied, the event class and event name to be used, and the lifetime of the reservation.</td>
</tr>
<tr>
<td>2</td>
<td>The IpAppChargingSession object will query the billing engine and in this instance successfully reserves the requested number of named events. It informs the IpAppChargingSession object by calling the <code>reserveUnitRes()</code> method.</td>
</tr>
<tr>
<td>3</td>
<td>The application cancels the reservation using the <code>release()</code> method. The IpAppChargingSession does not wait for a confirmation. The charging session is now closed and cannot be used for any other transactions.</td>
</tr>
</tbody>
</table>

Direct Debit (Named Events)

Direct Debit (Named Events)

Messages

<table>
<thead>
<tr>
<th>Step</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>At the beginning of this scenario the subscriber has a balance equivalent to the cost of 100 named events. The IpAppChargingSession invokes a <code>directDebitUnitReq()</code> on the IpChargingSession containing details of the account to debit, the wallet/balance type, the amount to debit (10 named events), the event class and event name to be used, and any discount to be applied. The IpChargingSession object will attempt to debit the account on the billing engine and in this instance it is successful.</td>
</tr>
<tr>
<td>2</td>
<td>The IpAppChargingSession object will query the billing engine and in this instance successfully debits the account by the equivalent cost of the requested named events. It informs the IpAppChargingSession object by calling the <code>directDebitAmountRes()</code> method. The subscriber will now have a balance equivalent to the cost of 90 named events.</td>
</tr>
<tr>
<td>3</td>
<td>At the beginning of this scenario the subscriber has a balance equivalent to the cost of 100 named events. The IpAppChargingSession invokes a <code>directDebitUnitReq()</code> on the IpChargingSession containing details of the account to debit, the wallet/balance type, the amount to debit (10 named events), the event class and event name to be used, and any discount to be applied. The IpChargingSession object will attempt to debit the account on the billing engine.</td>
</tr>
</tbody>
</table>
Step | Details
---|---
4 | The IpChargingSession object will query the billing engine and in this instance fails to debit the account by the equivalent cost of the requested named events. It informs the IpAppChargingSession object by calling the directDebitAmountErr() method with the appropriate error cause. The subscriber will still have a balance equivalent to the cost of 100 named events.

**Direct Credit**

<table>
<thead>
<tr>
<th>Step</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>At the beginning of this scenario the subscriber has a balance equivalent to the cost of 100 named events. The IpAppChargingSession invokes a directCreditUnitReq() on the IpChargingSession containing details of the account to credit, the wallet/balance type, the amount to credit (10 named events), the event class and event name to be used, and any discount to be applied. The IpChargingSession object will attempt to credit the account on the billing engine and in this instance it is successful.</td>
</tr>
<tr>
<td>2</td>
<td>The IpAppChargingSession object will query the billing engine and in this instance successfully credits the account by the equivalent cost of the requested named events. It informs the IpAppChargingSession object by calling the directDebitAmountRes() method. The subscriber will now have a balance equivalent to the cost of 110 named events.</td>
</tr>
<tr>
<td>3</td>
<td>At the beginning of this scenario the subscriber has a balance equivalent to the cost of 100 named events. The IpAppChargingSession invokes a directCreditUnitReq() on the IpChargingSession containing details of the account to credit, the wallet/balance type, the amount to credit (10 named events), the event class and event name to be used, and any discount to be applied. The IpChargingSession object will attempt to credit the account on the billing engine and in this instance it is unsuccessful.</td>
</tr>
<tr>
<td>4</td>
<td>The IpChargingSession object will query the billing engine and in this instance fails to credit the account by the equivalent cost of the requested named events. It informs the IpAppChargingSession object by calling the directDebitAmountErr() method with the appropriate error cause. The subscriber will still have a balance equivalent to the cost of 100 named events.</td>
</tr>
</tbody>
</table>
Multiple Reservations with Confirmation - OSA Plus

At the beginning of this scenario the subscriber has a balance of 100 billing engine monetary units. The IpAppChargingSession of Fusion Works instance A invokes a reserveAmountWithApplicationReferenceReq( ) on the IpChargingSession containing details of the account to reserve from, the wallet/balance type, the amount to reserve (20 BE units), any discount to be applied, and the lifetime of the reservation. The subscriber’s account balance will remain at 100 billing engine monetary units, however there will only be 80 billing engine monetary units available for alternative reservations.

The IpAppChargingSession object will query the billing engine and in this instance successfully reserves the requested amount. It informs the IpAppChargingSession of Fusion Works instance A object by calling the reserveAmountRes( ) method.
### Step 3 Details

The application wants to debit 10 of the reserved BE units against the reservation held on the OSA CHAM SCS gateway. It uses the `debitAmountWithApplicationReferenceReq()` method (with the `closeReservation` flag set to 'false'). The current reservation on the OSA CHAM SCS gateway now has 10 reserved BE units and 90 BE units that can still be used through sending a subsequent `debitUnitReq()`. The `debit(credit)UnitReq()` message only modifies the reserved funds on the OSA CHAM SCS gateway, it does not does not send a request to the BE. If the subscriber wants to confirm the funds debited or credited from the reservation held on the OSA CHAM SCS gateway then either a `release()` message can be used or alternatively the 'closeReservation' flag for a subsequent `debit(credit)UnitReq()` message can be used. The funds confirmed will be those that have been debited from the reservation on the OSA CHAM SCS gateway.

For example:

- `reserveAmountReq` for 100 BE units
- `debitAmountReq` for 10 BE units (closeReservation = 'false')
- `debitAmountReq` for 10 BE units (closeReservation = 'true')

This will reserve 20 BE units on the subscriber’s account and close the reservation on both the OSA CHAM SCS gateway and the BE.

### Step 4 Details

The reservation alteration is successful – the `debitAmountRes()` method is used to inform the application.

### Step 5 Details

The application needs a subsequent reservation and uses the `reserveAmountWithApplicationReferenceReq()` method to request 10 more BE units. The `FusionWorksinstance` to be used in the call back is modified. The subscriber’s account balance of 100 billing engine monetary units, however, there will only be the equivalent cost of 70 BE units available for alternative transactions once this transaction has completed. These unreserved units in the user’s account are available for alternative reservations from this or other third-party client applications. The current reservation on the OSA CHAM SCS gateway will now have 20 reserved BE units, and 10 BE units waiting to be confirmed through the subscriber sending a `release()` method (or using the `closeReservation` flag of a subsequent `debit(credit)AmountWithApplicationReferenceReq()` method).

### Step 6 Details

The `IpAppChargingSession` object will query the billing engine and in this instance successfully reserves the requested amount. It informs the `IpAppChargingSession` object of Fusion Works instance B by calling the `reserveAmountRes()` method.

### Step 7 Details

The application wants to credit some from the debited amount back to the reservation held on the OSA CHAM SCS gateway. It invokes the `creditAmountWithApplicationReferenceReq()` method for 5 BE units. This registers Fusion Works instance A to receive the call-backs from the CHAM. The subscriber’s account balance of 100 billing engine monetary units. The current reservation on the OSA CHAM SCS gateway now has 25 reserved BE units, and 5 BE units waiting to be confirmed through the subscriber sending a `release()` method (or using the `closeReservation` flag of a subsequent `debit(credit)AmountWithApplicationReferenceReq()` method).

If no more messages were received before the session times out, the subscriber’s balance on the billing engine would not be changed.

### Step 8 Details

The reservation alteration is successful – the `creditUnitRes()` method is used to inform the application of Fusion Works instance A.
<table>
<thead>
<tr>
<th>Step</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>The application uses the debitAmountWithApplicationReferenceReq() method to debit and confirm an additional 2 BE units against the reservation and have these units confirmed on the billing engine. The debitAmountWithApplicationReferenceReq() method is invoked with the closeReservation flag to 'true'. This registers Fusion Works instance A to receive the call-backs from the CHAM. The reservation is confirmed (7 BE units) through the IpChargingSession informing the billing engine. The current reservation is now closed and the subscriber’s account would be reduced by 7 billing engine monetary units. The 23 unconfirmed BE units, those that were initially reserved but not confirmed, would be available for other reservations. The subscriber will now have a balance of 93 billing engine monetary units.</td>
</tr>
<tr>
<td>10</td>
<td>The request is successful and the application is informed with the debitUnitRes() method of Fusion Works instance B.</td>
</tr>
</tbody>
</table>

**Note:** This charging session may now be used for subsequent reservations.
Chapter 7

Account Manager Functionality

Overview

Introduction

This chapter explains the methods that are available using the IpAccountManager. The IpAccountManager object resides on the OSA CHAM SCS gateway and provides the following methods that can be invoked by the third-party client application.

In this chapter

This chapter contains the following topics.

- SetCallBack ()
- QueryBalanceReq()
- RetrieveTransactionHistoryReq()
- CreateNotification()
- ChangeNotification()
- GetNotification()
- DestroyNotification()
- Account Manager Parameters

SetCallBack ()

Description

This method may be invoked on the IpAccountManager to pass the reference to the IpAppAccountManager object to be used by the SCF for call-backs.

Parameters

Here is the parameter for this method.

- appInterface (on page 105)

Returns

Nothing.

QueryBalanceReq()

Description

This method may be invoked on the IpAccountManager and cause a billing engine query for each account in the TpAddressSet and the call-back of either the queryBalanceRes() or queryBalanceErr() method on the IpAppAccountManager.
Parameters

Here is the parameter for this method.

- users (on page 106)

TpAddress structure

The treatment and expected type of each element within each TpAddress structure is:

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Type</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plan</td>
<td>TpAddressPlan</td>
<td>Ignored</td>
</tr>
<tr>
<td>AddrString</td>
<td>TpString</td>
<td>Defines the Account Reference CLI (e.g.: MSISDN) in the format stored in CCS/BE database tables.</td>
</tr>
<tr>
<td>Name</td>
<td>TpString</td>
<td>Ignored</td>
</tr>
<tr>
<td>Presentation</td>
<td>TpAddressPresentation</td>
<td>Ignored</td>
</tr>
<tr>
<td>Screening</td>
<td>TpAddressScreening</td>
<td>Ignored</td>
</tr>
<tr>
<td>SubAddressString</td>
<td>TpString</td>
<td>This parameter is used to define the following parameters:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- wallet type name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- balance type name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- balance type cascade name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- additional subscriber parameters that will be listed separately below.</td>
</tr>
</tbody>
</table>

The wallet type name, balance type name and balance cascade name and additional subscriber parameters should be separated by a ';'. If no wallet type name or balance type name is given then the default wallet type/balance type for this account will be used. If the balance type cascade name is provided then the balance type name will be ignored.

When a balance type name is specified then only the details for this balance type will be returned.

If the balance type name ‘ALL’ is specified then the system value of all buckets in the balance type and the earliest expiring bucket for each balance type will be returned.

Where a balance type cascade name is specified then the resulting Balance Info Set will contain an entry for each balance type unit within the balance type cascade.
<table>
<thead>
<tr>
<th>Element Name</th>
<th>Type</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>The list of additional subscriber parameters that may be queried are as follows (the specific syntax for each parameter will be matched in the OSA gateway so the OSA client must adhere to this syntax):</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IA -&gt; Initial Activation Date</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LRA -&gt; Last Recharge Amount</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LBBR -&gt; Last Balance Before Recharge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AC -&gt; Account Creation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FDS -&gt; FFFD Status</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFN -&gt; FF Numbers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FDN -&gt; FD Number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LR -&gt; Last Recharge Date</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PBE -&gt; Previous Balance Expiry Date</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PWE -&gt; Previous Wallet Expiry Date</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEA -&gt; Least Expired Amount</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEA -&gt; Total Expired Amount</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBA -&gt; Initial Balance Amount</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The additional subscriber parameters should be ‘|’ separated and separated from the initial charging details by a ‘;’.

For example, specifying a wallet type name, balance type name and friends & family numbers would have the following syntax:

“Personal;General Cash;;FFN”

**Returns**

TpAssignmentID: a unique id (unsigned integer) within the instance of IpAccountManager for the period of transaction.

**RetrieveTransactionHistoryReq()**

**Description**

This method may be invoked on the IpAccountManager to request transaction details for the specified account for the provided time interval. Implementation of this method will involve a query to the SMS to retrieve CDR details and therefore relies on this information remaining available, as per standard Oracle Global architecture.

**Parameters**

Here is the parameter for this method.

- *user* (on page 106)
- *transactionInterval* (on page 105)
TpAddress structure

The treatment and expected type of each element within each TpAddress structure is:

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Type</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plan</td>
<td>TpAddressPlan</td>
<td>Ignored</td>
</tr>
<tr>
<td>AddrString</td>
<td>TpString</td>
<td>Defines the Account Reference CLI (for example: MSISDN) in the format stored in CCS/BE database tables.</td>
</tr>
<tr>
<td>Name</td>
<td>TpString</td>
<td>Ignored</td>
</tr>
<tr>
<td>Presentation</td>
<td>TpAddressPresentation</td>
<td>Ignored</td>
</tr>
<tr>
<td>Screening</td>
<td>TpAddressScreening</td>
<td>Ignored</td>
</tr>
<tr>
<td>SubAddressString</td>
<td>TpString</td>
<td>This parameter is used to define the following parameters:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• wallet type name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• balance type name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The wallet type name and balance type name should be separated by a ';' :</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If no wallet type name or balance type name is given then the default wallet type/balance type for this account will be used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When a balance type name is specified then only the details for this balance type will be returned.</td>
</tr>
</tbody>
</table>

Returns

TpAssignment ID – a unique id (unsigned integer) within the instance of IpAccountManager for the period of transaction

CreateNotification()

Description

This method may be invoked on the IpAccountManager and cause triggers to be set in the billing engine for each account specified. This enables notification of events for the account specified to be sent back to the third-party client application. As a reference to the IpAppAccountManager object is passed to the IpAccountManager, this method is unique because an alternative IpAppAccountManager object can be used by the SCF for call-backs.

Note: A specified event criterion applies to all users specified in the request. As there is no way to specify a unique user in the reportNotification() method, it is recommended that only a single user is specified in the createNotification() request.

Parameters

Here are the parameters for this method.

- AppAccountManager (on page 105)
- ChargingEventCriteria (on page 105)
TpAddress structure

The treatment and expected type of each element within each TpAddress structure is:

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Type</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plan</td>
<td>TpAddressPlan</td>
<td>Ignored</td>
</tr>
<tr>
<td>AddrString</td>
<td>TpString</td>
<td>Defines the Account Reference CLI (e.g.: MSISDN) in the format stored in CCS/BE database tables.</td>
</tr>
<tr>
<td>Name</td>
<td>TpString</td>
<td>Ignored</td>
</tr>
<tr>
<td>Presentation</td>
<td>TpAddressPresentation</td>
<td>Ignored</td>
</tr>
<tr>
<td>Screening</td>
<td>TpAddressScreening</td>
<td>Ignored</td>
</tr>
<tr>
<td>SubAddressString</td>
<td>TpString</td>
<td>This parameter is used to define the following parameters:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• wallet type name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• balance type name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The wallet type name and balance type name should be separated by a ';'. If no wallet type name or balance type name is given then the default wallet type/balance type for this account will be used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When a balance type name is specified then only the details for this balance type will be returned.</td>
</tr>
</tbody>
</table>

Returns

TpAssignment ID – a unique id (unsigned integer) within the instance of IpAccountManager for the period of transaction.

ChangeNotification()

Description

This method is invoked on the IpAccountManager and cause triggers to be updated in the billing engine for each account specified. This enables notification of events for the account specified to be sent back to the third-party client application. This method will overwrite any notifications of events (for the id supplied here) that were set using the createNotification() method.

Note: A specified event criterion applies to all users specified in the request. As there is no way to specify a unique user in the reportNotification() method, it is recommended that only a single user is specified in the createNotification() request.

Parameters

Here is a list of the parameters for this method.

- assignmentID (on page 105)
- ChargingEventCriteria (on page 105)
**TpAddress structure**

The treatment and expected type of each element within each TpAddress structure is:

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Type</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plan</td>
<td>TpAddressPlan</td>
<td>Ignored</td>
</tr>
<tr>
<td>AddrString</td>
<td>TpString</td>
<td>Defines the Account Reference CLI (e.g.: MSISDN) in the format stored in CCS/BE database tables.</td>
</tr>
<tr>
<td>Name</td>
<td>TpString</td>
<td>Ignored</td>
</tr>
<tr>
<td>Presentation</td>
<td>TpAddressPresentation</td>
<td>Ignored</td>
</tr>
<tr>
<td>Screening</td>
<td>TpAddressScreening</td>
<td>Ignored</td>
</tr>
<tr>
<td>SubAddressString</td>
<td>TpString</td>
<td>This parameter is used to define the following parameters:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- wallet type name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- balance type name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The wallet type name and balance type name should be separated by a ':'. If no wallet type name or balance type name is given then the default wallet type/balance type for this account will be used. When a balance type name is specified then only the details for this balance type will be returned.</td>
</tr>
</tbody>
</table>

**Returns**

Nothing.

**GetNotification()**

**Description**

This method may be invoked on the IpAccountManager to query current charging event criteria that was introduced by the third-party client application using the createNotification() method or modified using the changeNotification() method.

**Note:** This method may return a large amount of data.

**Parameters**

None.

**Returns**

TpChargingEventCriteriaResultSet – this is a set containing details of every open notification for this session. The set contains a numbered set of TpChargingEventName values and the associated TpAssignmentID value that was returned to the third-party client application when the notification trigger was invoked using the createNotification() method. (See standard [4] for TpChargingEventName encoding)
DestroyNotification()

Description
This method may be invoked on the IpAccountManager to disable all current notifications associated with the id supplied. The id was returned to the third-party client application when the createNotification() method was invoked. This will affect all notifications that were later modified using the changeNotification() method.

Parameter
Here is the parameter used for this method.

- **assignmentID** (on page 105)

Returns
Nothing.

Account Manager Parameters

Introduction
Here are the parameters used for this functionality.

**AppAccountManager**

Description: Reference to an IpAppAccountManager object. (This may be different to the current IpAppAccountManager object making the request)

**appInterface**

Description: Reference to an IpAppAccountManager object.

**assignmentID**

Description: The TpAssignmentID returned by the IpAccountManager when the createNotification() method was invoked. This unique id identifies the set of triggers that the IpAccountManager has already set using the createNotification() method.

**ChargingEventCriteria**

Description: A TpChargingEventCriteria, which contains a set of each of TpAddress structures and TpChargingEventName values. (See standard [4] for TpChargingEventName encoding.)

**transactionInterval**

Description: A TpTimeInterval structure which specifies the time interval (start and stop time) for which the third-party client application is to be retrieved. Each time interval is expressed as a TpDateAndTime structure. (See standard [2] for TpDateAndTime encoding)
user
Description: A TpAddress containing only a single TpAddress structure for the specified user.

users
Description: A TpAddressSet containing a numbered set of TpAddress structures.
Overview

Introduction

This chapter explains the methods available using the IpAppAccountManager. The IpAppAccountManager object resides on the third-party client application and provides the following methods that can be invoked by the OSA CHAM SCS gateway.

In this chapter

This chapter contains the following topics.

- QueryBalanceRes() 107
- QueryBalanceErr() 113
- RetrieveTransactionHistoryRes() 113
- RetrieveTransactionHistoryErr() 114
- ReportNotification() 114
- Application Account Manager Parameters 116

QueryBalanceRes()

Description

This method will be invoked on the IpAppAccountManager as a result of a successful queryBalanceReq(). The information returned will be a summary of the subscriber’s wallet details for a single wallet type and balance type, a sum of all the available, i.e. non-expired balances, for each balance type unit within the entered balance type cascade or all the balance types (balance type name == ‘ALL’) and the expiry date of the earliest expiring bucket in the balance type for all the available, i.e. non-expired balances in the subscriber’s wallet. If the wallet type name, balance type name or balance type cascade name were not specified in the initial request, then the account query will continue using the user’s default wallet type and balance type. The response data will include a set of account balances, the account type, the wallet state, and a list of available dates (if they exist) that may include the following: the account expiry, the earliest wallet expiry, the date of the last use of the account, and the date when the last reference to the account was made. Additional subscriber details may be returned if any of the additional subscriber parameters listed below are added to the SubAddressString field in the queryBalanceReq(). The following list is all the additional subscriber details that can be returned in the queryBalanceRes():

- InitialActivationDate
- LastRechargeAmount
- LastBalanceBeforeRecharge
- AccountCreationDate
- FFFDStatus
- FFNumbers
Chapter 8

- FDNumber
- LastRechargeDate
- PreviousBalanceExpiryDate
- PreviousWalletExpiryDate
- LastExpiredAmount
- ServiceProviderName
- TotalExpiredAmount
- InitialBalanceAmount

**Note:** The balance amount will be returned as major currency (i.e. exponent of 2) if a real currency is specified and with exponent of 0 if the currency is specified as 'XXX'

**Parameters**

Here is a list of the parameters for this method.

- *queryId* (on page 117)
- *balances* (on page 116)

**TpAddress structure**

The treatment and expected type of each element within each TpAddress structure is:

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Type</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plan</td>
<td>TpAddressPlan</td>
<td>Ignored (field will contain the default P_ADDR_PLAN_UNDEFINED string)</td>
</tr>
<tr>
<td>AddrString</td>
<td>TpString</td>
<td>Defines the Account Reference CLI (e.g.: MSISDN) in the format stored in CCS/BE database tables.</td>
</tr>
<tr>
<td>Name</td>
<td>TpString</td>
<td>Ignored (Empty)</td>
</tr>
<tr>
<td>Presentation</td>
<td>TpAddressPresentation</td>
<td>Ignored (field will contain the default P_ADDRESS_PRESENTATION_UNDEFINED string)</td>
</tr>
<tr>
<td>Screening</td>
<td>TpAddressScreening</td>
<td>Ignored (field will contain the default P_ADDRESS_SCREENING_UNDEFINED string)</td>
</tr>
<tr>
<td>SubAddressString</td>
<td>TpString</td>
<td>This parameter is used to define the following parameters:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- wallet type name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- balance type name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- balance type cascade name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- additional subscriber parameters that will be listed separately below.</td>
</tr>
</tbody>
</table>

The wallet type name, balance type name and balance cascade name and additional subscriber parameters should be separated by a ';'. If no wallet type name or balance type name is given then the default wallet type/balance type for this account will be used. If the balance type cascade name is provided then the balance type name will be ignored.
<table>
<thead>
<tr>
<th>Element Name</th>
<th>Type</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>When a balance type name is specified then only the details for this balance type will be returned.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If the balance type name ‘ALL’ is specified then the system value of all buckets in the balance type and the earliest expiring bucket for each balance type will be returned.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Where a balance type cascade name is specified then the resulting Balance Info Set will contain an entry for each balance type unit within the balance type cascade.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The list of additional subscriber parameters that may be queried are as follows (the specific syntax for each parameter will be matched in the OSA gateway so the OSA client must adhere to this syntax):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• IA -&gt; Initial Activation Date</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• LRA -&gt; Last Recharge Amount</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• LBBR -&gt; Last Balance Before Recharge</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• AC -&gt; Account Creation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• FDS -&gt; FFFD Status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• FFN -&gt; FF Numbers</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• FDN -&gt; FD Number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• LR -&gt; Last Recharge Date</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• PBE -&gt; Previous Balance Expiry Date</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• PWE -&gt; Previous Wallet Expiry Date</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• LEA -&gt; Least Expired Amount</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• TEA -&gt; Total Expired Amount</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• IBA -&gt; Initial Balance Amount</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The additional subscriber parameters should be ‘</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For example, specifying a wallet type name, balance type name and friends &amp; family numbers would have the following syntax:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“Personal;General Cash;;FFN”</td>
</tr>
</tbody>
</table>

**TpBalanceSet Additional Information**

The AdditionalInfo field in the TpBalanceInfo structure contains the following information encoded as follows:
Where:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wallet State</td>
<td>The 'wallet state' of the TpBalanceInfo AdditionalInfo string is encoded as a single character as follows.</td>
</tr>
<tr>
<td>Last Wallet Use</td>
<td>The 'last use date' of the TpBalanceInfo Additional Info string is encoded as a date format providing the last use of the wallet as taken from the OSA VWS QueryAccountAck message.</td>
</tr>
<tr>
<td>Last Reference Use</td>
<td>The 'last reference use' of the TpBalanceInfo Additional Info string is encoded as a date format providing the last use of the account reference as taken from the CCS_ACCT_REFERENCE table using the last_accessed column. The account reference is a user of an account, i.e. home phone CLI, MSISDN, or calling card.</td>
</tr>
<tr>
<td>Balance Expiry Date</td>
<td>The 'balance expiry date' of the TpBalanceInfo Additional Info string is encoded as a date format providing the date at which the earliest bucket in the current balance expires wallet as taken from the OSA VWS QueryAccountAck message.</td>
</tr>
<tr>
<td>Wallet Expiry Date</td>
<td>The 'wallet expiry date' of the TpBalanceInfo Additional Info string is encoded as a date format providing the date at which the current wallet expires wallet as taken from the OSA VWS QueryAccountAck message.</td>
</tr>
<tr>
<td>Service Provider</td>
<td>The 'service provider' of the TpBalanceInfo Additional Info string is encoded as a string corresponding to the name of the subscriber's service provider for the current wallet as taken from the ACS_CUSTOMER table using the name column.</td>
</tr>
</tbody>
</table>

**Parameter** | **Description**
--- | ---
Wallet State | The 'wallet state' of the TpBalanceInfo AdditionalInfo string is encoded as a single character as follows.
<table>
<thead>
<tr>
<th>Character</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Active</td>
</tr>
<tr>
<td>D</td>
<td>Dormant</td>
</tr>
<tr>
<td>F</td>
<td>Frozen</td>
</tr>
<tr>
<td>T</td>
<td>Terminated</td>
</tr>
<tr>
<td>P</td>
<td>Pre-Use</td>
</tr>
<tr>
<td>X</td>
<td>Unsupported</td>
</tr>
</tbody>
</table>

Last Wallet Use | The 'last use date' of the TpBalanceInfo Additional Info string is encoded as a date format providing the last use of the wallet as taken from the OSA VWS QueryAccountAck message. |
If this parameter cannot be determined, or is not yet provided then the parameter will remain empty.

Last Reference Use | The 'last reference use' of the TpBalanceInfo Additional Info string is encoded as a date format providing the last use of the account reference as taken from the CCS_ACCT_REFERENCE table using the last_accessed column. The account reference is a user of an account, i.e. home phone CLI, MSISDN, or calling card. |
If this parameter cannot be determined, or is not yet provided then the parameter will remain empty.

Balance Expiry Date | The 'balance expiry date' of the TpBalanceInfo Additional Info string is encoded as a date format providing the date at which the earliest bucket in the current balance expires wallet as taken from the OSA VWS QueryAccountAck message. |
If this parameter cannot be determined, is not yet provided, or the balance does not expire then the parameter will remain empty.

Wallet Expiry Date | The 'wallet expiry date' of the TpBalanceInfo Additional Info string is encoded as a date format providing the date at which the current wallet expires wallet as taken from the OSA VWS QueryAccountAck message. |
If this parameter cannot be determined, is not yet provided, or the account does not expire then the parameter will remain empty.

Service Provider | The 'service provider' of the TpBalanceInfo Additional Info string is encoded as a string corresponding to the name of the subscriber's service provider for the current wallet as taken from the ACS_CUSTOMER table using the name column. |
If this parameter cannot be determined, or is not yet provided then the parameter will remain empty.
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account Type</td>
<td>The ‘account type’ of the TpBalanceInfo Additional Info string is encoded as a string corresponding to the name of the subscriber's account type for the current wallet as taken from the CCS_ACCT_TYPE table using the name column. This parameter is used to identify the service configuration for this call where the name of the account type will be agreed between CCS and the billing system. If this parameter cannot be determined, or is not yet provided then the parameter will remain empty.</td>
</tr>
<tr>
<td>Subscriber’s Preferred Language</td>
<td>The ‘subscribers preferred language’ of the TpBalanceInfo Additional Info string is encoded as a string corresponding to the name of the subscriber’s preferred language for the current account as taken from the ACS_LANGUAGE table using the name column. The numeric value for the preferred language is agreed between CCS and the billing system. If this parameter cannot be determined, or is not yet provided then the parameter will remain empty.</td>
</tr>
<tr>
<td>Balance Type Name</td>
<td>The 'balance type name' of the TpBalanceInfo Additional Info string is encoded as a string corresponding to the name of the balance type as taken from the CCS_BALANCE_UNIT table using the name column. This parameter will only be present if the balance type name = ‘ALL’ is specified in the incoming request.</td>
</tr>
<tr>
<td>Balance Unit Type Name</td>
<td>The 'balance unit type name' of the TpBalanceInfo Additional Info string is encoded as a string corresponding to the name of the balance type unit for this balance type as taken from the CCS_BALANCE_UNIT table using the name column. If this parameter cannot be determined, or is not yet provided then the parameter will remain empty.</td>
</tr>
<tr>
<td>Account Creation Date</td>
<td>The ‘account creation date’ of the TpBalanceInfo Additional Info string is encoded as a date format providing the creation date of the wallet as taken from the CCS_ACCT_HIST_INFO using the creation_date column accessed through a remote database link to the SMS. If this parameter cannot be determined, or is not yet provided, then the parameter will remain empty.</td>
</tr>
<tr>
<td>Initial Balance Amount</td>
<td>The ‘initial balance amount’ of the TpBalanceInfo Additional Info string is encoded as an integer providing the initial balance of the wallet in VWS small currency as taken from the CCS_ACCT_REFERENCE table using the initial_balance_amount column. If this parameter cannot be determined, or is not yet provided then the parameter will remain empty.</td>
</tr>
<tr>
<td>Initial Activation Date</td>
<td>The ‘initial activation date’ of the TpBalanceInfo Additional Info string is encoded as a date format providing the initial activation date of the wallet as taken from the CCS_ACCT_HIST_INFO using the initial_activation_date column accessed through a remote database link to the SMS. If this parameter cannot be determined, is not yet provided, or the wallet has not yet been activated, then the parameter will remain empty.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Last Recharge Amount</td>
<td>The 'last recharge amount' of the TpBalanceInfo Additional Info string is encoded as an integer providing the last recharge amount of the wallet in VWS small currency as taken from the CCS_ACCT_HIST_INFO using the last_recharge_amount column accessed through a remote database link to the SMS. If this parameter cannot be determined, is not yet provided, or the wallet has not yet been recharged, then the parameter will remain empty.</td>
</tr>
<tr>
<td>Last Recharge Date</td>
<td>The 'last recharge date' of the TpBalanceInfo Additional Info string is encoded as a date format providing the last recharge date of the wallet as taken from the CCS_ACCT_HIST_INFO using the last_recharge_date column accessed through a remote database link to the SMS. If this parameter cannot be determined, is not yet provided, or the wallet has not yet been recharged, then the parameter will remain empty.</td>
</tr>
<tr>
<td>Last Balance Before Recharge</td>
<td>The 'last balance before recharge' of the TpBalanceInfo Additional Info string is encoded as an integer providing the last balance before recharge amount of the wallet. The amount will be in VWS small currency as taken from the CCS_ACCT_HIST_INFO using the last_balance_before_recharge column accessed through a remote database link to the SMS. If this parameter cannot be determined, is not yet provided, or the wallet has not yet been recharged, then the parameter will remain empty.</td>
</tr>
<tr>
<td>Previous Balance Expiry Date</td>
<td>The 'previous balance expiry date' of the TpBalanceInfo Additional Info string is encoded as a date format providing the date when a balance previously expired in the wallet as taken from the CCS_ACCT_HIST_INFO using the last_balance_expiry_period column accessed through a remote database link to the SMS. If this parameter cannot be determined, or is not yet provided then the parameter will remain empty.</td>
</tr>
<tr>
<td>Previous Wallet Expiry Date</td>
<td>The 'previous wallet expiry date' of the TpBalanceInfo Additional Info string is encoded as a date format providing the date when the wallet previously expired as taken from the CCS_ACCT_HIST_INFO using the last_wallet_expiry_period column accessed through a remote database link to the SMS. If this parameter cannot be determined, or is not yet provided then the parameter will remain empty.</td>
</tr>
<tr>
<td>Last Expired Amount</td>
<td>The 'last expired amount' of the TpBalanceInfo Additional Info string is encoded as an integer providing the amount of the last expired balance (for 'Cash' balances only) in the wallet. The amount is in VWS small currency as taken from the CCS_ACCT_HIST_INFO using the last_expired_amount column accessed through a remote database link to the SMS. If this parameter cannot be determined, or is not yet provided then the parameter will remain empty.</td>
</tr>
<tr>
<td>Total Expired Amount</td>
<td>The 'total expired amount' of the TpBalanceInfo Additional Info string is encoded as an integer providing the sum of all expired balances (for 'Cash' balances only) in the wallet. The amount will be in VWS small currency as taken from the CCS_ACCT_HIST_INFO using the total_expired_amount column accessed through a remote database link to the SMS. If this parameter cannot be determined, or is not yet provided then the parameter will remain empty.</td>
</tr>
<tr>
<td>Friends &amp; Family, Friends &amp; Destination Status</td>
<td>The 'friends &amp; family, friends and destination status' of the TpBalanceInfo Additional Info string is encoded as a string providing the current F&amp;F / F&amp;D status as taken from the CCS_ACCT_REFERENCE table using the PT_FF_FD_SERVICE_ACTIVE profile tag in the profile column. The string</td>
</tr>
</tbody>
</table>
## Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>returned will be one of:</td>
</tr>
<tr>
<td></td>
<td>• &quot;Not Active&quot; (not subscribed),</td>
</tr>
<tr>
<td></td>
<td>• &quot;FF Active&quot; (subscribed to the friends &amp; family service), or</td>
</tr>
<tr>
<td></td>
<td>• &quot;FD Active&quot; (subscribed to the friends &amp; destination service).</td>
</tr>
<tr>
<td>Friends &amp; Family Numbers</td>
<td>The 'friends &amp; family numbers' of the TpBalanceInfo Additional Info string is encoded as a ',' separated string providing the current F&amp;F numbers as taken from the CCS_ACCT_REFERENCE table using the PT_FF_NUMBERS profile tag in the profile column. The string returned is a list of the currently defined friends &amp; family numbers. If this parameter cannot be determined, or is not yet provided then the parameter will remain empty.</td>
</tr>
<tr>
<td>Friends &amp; Destination Number</td>
<td>The 'friends &amp; destination number' of the TpBalanceInfo Additional Info string is encoded as a string providing the current F&amp;D number as taken from the CCS_ACCT_REFERENCE table using the PT_FD_NUMBERS profile tag in the profile column. The string returned is the currently defined friends &amp; destination number. If this parameter cannot be determined, or is not yet provided then the parameter will remain empty.</td>
</tr>
</tbody>
</table>

### Notes:
- Dates are encoded as YYYYMMDDHHMMSS. for example, 20020215201500
- All balance types consist of 0 or more buckets. An individual bucket may or may not have an individual expiry date. The expiry date of an individual balance type is the earliest expiry date for an individual bucket within the respective balance type. If none of the buckets expire, then the balance type will not expire. The total for a balance type is a sum of all the balances for the individual buckets.

---

### QueryBalanceErr()

#### Description

This method is invoked on the IpAppAccountManager as a result of an unsuccessful queryBalanceReq().

#### Parameters

Here is a list of the parameters for this method.

- `queryId` (on page 117)
- `cause` (on page 116)

---

### RetrieveTransactionHistoryRes()

#### Description

This method is invoked on the IpAppAccountManager as a result of a successful retrieveTransactionHistoryReq(). The information returned is a summary of the user’s account transaction details for a single wallet and balance type over a given interval. If the wallet and balance type were not specified in the initial request, then the account transaction history query will continue using the user’s default wallet and balance type.
Parameters
Here the parameters for this method.

- retrievalId (on page 117)
- transactionHistory (on page 117)

Additional information
The AdditionalInfo field in the TpTransactionHistory structure contains the string returned from querying the 'EXTRA_INFORMATION' field of the CCS_BE_CDR table for a particular MSISDN. This string is returned unchanged to the third-party client application. The following is a sample of this CDR field:

```
CLI=48495100100|ACS_CUST_ID=1|BALANCE_TYPES=1|BALANCES=1991000|COSTS=1000|ACCOUNT_TY
PE=1|CASCADE_ID=1|RATES=1000|LENGTHS=-1|DISCOUNTS=0|MAX_CHARGE=5000|
DURATION=60|TN=48495100100|TCS=20030528144351|TCE=20030528144451|CS=S|DISCOUNT_TYPE=
BASIC|WALLET_TYPE=1|SMSMO_MSG_ID=1
```

RetrieveTransactionHistoryErr()

Description
This method is invoked on the IpAppAccountManager as a result of an unsuccessful retrieveTransactionHistoryReq().

Parameters
Here is a list of the parameters for this method.

- queryId (on page 117)
- transactionHistoryError (on page 117)

Note: The error return codes are an explicit encoding predefined by the standard in [4], therefore the error returned to the OSA CHAM SCS gateway may not have a suitable mapped error definition. When this occurs P_AM_TRANSACTION_ERROR_UNSPECIFIED will be returned to the third-party client application.

ReportNotification()

Description
This method is invoked on the IpAppAccountManager as a result of a successful createNotification(). This method will notify the third-party client application of any charging event criteria that have been triggered on any of the user’s accounts established by the createNotification() method. If multiple user accounts were specified in the initial request then any of the individual account triggers will cause this method to be invoked.

Parameters
Here is a list of the parameters for this method.

- queryId (on page 117)
- chargingEventInfo (on page 116)

Note: The balance amount will be returned as major currency (i.e. exponent of 2) if a real currency is specified and with exponent of 0 if the currency is specified as 'XXX'.

114 NCC OSA CHAM Technical Guide
### Additional information

The AdditionalInfo field in the TpBalanceInfo structure will contain the following information encoded as follows:-

\[ S=<\text{wallet state}>;L=<\text{last wallet use}>;M=<\text{last reference use}>;B=<\text{balance expiry date}>;C=<\text{wallet expiry date}>;T=<\text{account type}>;A=<\text{subscribers preferred language}>;\text{MSISDN}=<\text{MSISDN}>\]

Where:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wallet State</td>
<td>The ‘wallet state’ of the TpBalanceInfo AdditionalInfo string is encoded as a single character as follows.</td>
</tr>
<tr>
<td></td>
<td><strong>Character</strong></td>
</tr>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>F</td>
</tr>
<tr>
<td></td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>P</td>
</tr>
<tr>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Last Wallet Use</td>
<td>The ‘last use date’ of the TpBalanceInfo AdditionalInfo string is encoded as a date format providing the last use of the account. If this parameter cannot be determined or is not yet provided then the parameter will remain empty.</td>
</tr>
<tr>
<td>Last Reference Use</td>
<td>The ‘last reference use’ of the TpBalanceInfo AdditionalInfo string is encoded as a date format providing the last use of the account reference. The account reference is a user of an account ie home phone CLI, MSISDN, or calling card. If this parameter cannot be determined or is not yet provided then the parameter will remain empty.</td>
</tr>
<tr>
<td>Balance Expiry Date</td>
<td>The ‘balance expiry date’ of the TpBalanceInfo AdditionalInfo string is encoded as a date format providing the date at which the earliest bucket in the current balance expires. If this parameter cannot be determined, is not yet provided, or the balance does not expire then the parameter will remain empty.</td>
</tr>
<tr>
<td>Account Expiry Date</td>
<td>The ‘account expiry date’ of the TpBalanceInfo AdditionalInfo string is encoded as a date format providing the date at which the current account expires. If this parameter cannot be determined, is not yet provided, or the account does not expire then the parameter will remain empty.</td>
</tr>
<tr>
<td>Account Type</td>
<td>The ‘account type’ of the TpBalanceInfo AdditionalInfo string is encoded as a string corresponding to the name of the subscriber’s account type for the current account. This parameter is used to identify the service configuration for this call where the name of the account type is agreed between CCS and the billing system. If this parameter cannot be determined or is not yet provided then the parameter will remain empty.</td>
</tr>
<tr>
<td>Subscribers Preferred Language</td>
<td>The ‘subscribers preferred language’ of the TpBalanceInfo AdditionalInfo string is encoded as a string corresponding to the name of the subscriber’s preferred language for the current account. The numeric value for the preferred language is agreed between CCS and the billing system. If this parameter cannot be determined or is not yet provided then the parameter will remain empty.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>MSISDN</td>
<td>The ‘MSISDN’ of the TpBalanceInfo AdditionalInfo string is encoded as a string corresponding to the MSISDN for the current account. This parameter is added to identify the account for which the charging event criteria have been triggered which has resulted in this report notification being generated on the third-party client application.</td>
</tr>
</tbody>
</table>

Notes:
- Dates are encoded as YYYYMMDDHHMMSS. e.g. 20020215201500
- All balance types consist of 0 or more buckets. An individual bucket may or may not have an individual expiry date. The expiry date of an individual balance type is the earliest expiry date for an individual bucket within the respective balance type. If none of the buckets expire, then the balance type will not expire. The total for a balance type is a sum of all the balances for the individual buckets.

Application Account Manager Parameters

Introduction
Here are the parameters used for this functionality.

balances

Description: A TpBalanceSet which is a set of numbered elements containing a:
- tpAddress structure,
- TpBalanceQueryError value (should be P_BALANCE_QUERY_OK), and
- TpBalanceInfo structure.
(See standard [4] for alternative TpBalanceQueryError and for TpBalanceInfo encodings)

cause

Description: A TpBalanceQueryError value which is either a mapping of the error returned from the billing engine or the error resolved at the OSA CHAM SCS gateway for the invoked request. (See standard [4] for the TpBalanceQueryError encoding.)

Note: The error return codes are an explicit encoding predefined by the standard in [4], therefore the error returned to the OSA CHAM SCS gateway may not have a suitable mapped error definition. When this occurs P_BALANCE_QUERY_ERROR_UNDEFINED is returned to the third-party client application.

chargingEventInfo


Note: The balance amount is returned as major currency (i.e. exponent of 2) if a real currency is specified and with exponent of 0 if the currency is specified as 'XXX'
queryId
Description: The TpAssignmentID returned by the IpAccountManager when the matching method, i.e.:
- createNotification()
- queryBalanceReq()
- retrieveTransactionHistoryReq()
was invoked.

retrievalId
Description: The TpAssignmentID returned by the IpAccountManager when the retrieveTransactionHistoryReq() method was invoked.

transactionHistory
Description: A TpTransactionHistorySet which is a set of numbered elements containing a TpAssignmentID (a local transactionID on the OSA CHAM SCS gateway – may or may not be the same as the retrievalId), a TpDateAndTime structure, and an AdditionalInfo string. (See standard [2] for TpDateAndTime encoding.)

transactionHistoryError
Description: A TpTransactionHistoryStatus value which is either a mapping of the error returned from the query or the error resolved at the OSA CHAM SCS gateway for the invoked request. (See standard [4] for the TpTransactionHistoryStatus encoding.)

Note: The error return codes are an explicit encoding predefined by the standard in [4], therefore the error returned to the OSA CHAM SCS gateway may not have a suitable mapped error definition. When this occurs P_AM_TRANSACTION_ERROR_UNSPECIFIED is returned to the third-party client application.
Overview

Introduction

This chapter explains the methods that are available using the IpChargingManager. The IpChargingManager object resides on the OSA CHAM SCS gateway and provides the following methods that can be invoked by the third-party client application.

In this chapter

This chapter contains the following topics.

SetCallBack() 119
CreateChargingSession() 120
CreateChargingSessionWithSessionID() 120
getChargingSession() 121
Charging Manager Parameters 122

SetCallBack()

Description

This method is invoked on the IpChargingManager to pass the reference to the IpAppChargingManager object to be used by the SCF for call-backs.

Parameters

Here is the parameter for this method.

- appInterface (on page 122)

Returns

Nothing.
CreateChargingSession()

Description

This method is invoked on the IpChargingManager creating a new charging session. This involves the OSA CHAM SCS gateway allocating a unique session ID that is stored in an internal table of active sessions. The third-party client application passes a reference to the IpAppChargingSession object to be used by the IpChargingSession charging methods when making call-backs. An explanation for all the charging methods will follow in subsequent sections of this document. Although there may be many simultaneous active sessions, there may be only a single IpAppChargingSession object and a single IpChargingSession object created on the third-party client application and the OSA CHAM SCS gateway respectively.

Parameters

Here is a list of the parameters for this method.

- **appChargingSession** (on page 122)
- **sessionDescription** (on page 123)
- **merchantAccount** (on page 123)
- **user** (on page 123)
- **correlationID** (on page 122)

Returns

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TpChargingSessionID</td>
<td>a TpChargingSessionID structure.</td>
</tr>
</tbody>
</table>

The treatment and expected type of each element within each TpChargingSessionID structure is:

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Type</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChargingSessionReference</td>
<td>IpChargingSessionRef</td>
<td>This element specifies a reference to the IpChargingSession object. (used by all the IpAppChargingSession charging methods)</td>
</tr>
<tr>
<td>ChargingSessionID</td>
<td>TpSessionID</td>
<td>This element specifies the unique session ID for the charging session. (used by all the IpAppChargingSession charging methods)</td>
</tr>
<tr>
<td>RequestNumberFirstRequest</td>
<td>TpInt32</td>
<td>This element specifies the request number to use for the next request. (used by all the IpAppChargingSession charging methods)</td>
</tr>
</tbody>
</table>

CreateChargingSessionWithSessionID()

Description

This method will be invoked on the IpChargingManager creating a new charging session. This will involve the OSA CHAM SCS gateway allocating a unique session ID that is stored in an internal table of active sessions. The third-party client application will pass a reference to the IpAppChargingSession object to be used by the IpChargingSession charging methods when making call-backs and in addition a session ID which may be used to reference this charging session.
Parameters

Here is a list of the parameters for this method.

- `appChargingSession` (on page 122)
- `sessionDescription` (on page 123)
- `merchantAccount` (on page 123)
- `user` (on page 123)
- `correlationID` (on page 122)
- `sessionID` (on page 123)

Returns

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TpChargingSessionID</td>
<td>a TpChargingSessionID structure.</td>
</tr>
</tbody>
</table>

The treatment and expected type of each element within each TpChargingSessionID structure is:

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Type</th>
<th>Usage Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChargingSessionReference</td>
<td>IpChargingSessionRef</td>
<td>This element specifies a reference to the IpChargingSession object. (used by all the IpAppChargingSession charging methods)</td>
</tr>
<tr>
<td>ChargingSessionID</td>
<td>TpSessionID</td>
<td>This element specifies the unique session ID for the charging session. (used by all the IpAppChargingSession charging methods)</td>
</tr>
<tr>
<td>RequestNumberFirstRequest</td>
<td>TpInt32</td>
<td>This element specifies the request number to use for the next request. (used by all the IpAppChargingSession charging methods)</td>
</tr>
</tbody>
</table>

Note: The session ID is still returned by this message to ensure maximum possible OSA compatibility.

`getChargingSession( )`

Description

This method is invoked on the IpChargingManager and returns the details of a reference to a previously created charging session from the free shared pool (no outstanding reservations and no outstanding UBE responses). The details returned the client will be a TpChargingSessionID structure containing the IpChargingSessionReference, ChargingSessionID and RequestNumberFirstRequest (to be used in any subsequent requests for the charging session). When invoked, the third-party client application will pass a reference to the IpAppChargingSession object to be used by the IpChargingSession charging methods when making call-backs.

If there are no:

- free IpChargingSession objects, the definition of a free IpChargingSession object is one with no outstanding reservations and no outstanding UBE responses, then the IpChargingSession reference of an IpChargingSession object with an outstanding reservation will be returned but with a -1 value for both the ChargingSessionID and RequestNumberFirstRequest fields.
- free IpChargingSession objects, and all the IpChargingSession objects have outstanding UBE responses then a PRESOURCE_UNAVAILABLE exception will be returned to the client.
• previously created charging session objects then a PRESOURCE_UNAVAILABLE exception will be returned to the client.

Parameters

Here is the parameter for this method.

• \textit{appChargingSession} (on page 122)

Returns

\begin{table}[h]
\centering
\begin{tabular}{|l|l|l|}
\hline
Parameter & Description \\
\hline
TpChargingSessionID & a TpChargingSessionID structure. \\
\hline
\end{tabular}
\end{table}

The treatment and expected type of each element within each TpChargingSessionID structure is:

\begin{table}[h]
\centering
\begin{tabular}{|l|l|l|}
\hline
Element Name & Type & Usage Notes \\
\hline
ChargingSessionReference & IpChargingSessionRef & This element specifies a reference to the IpChargingSession object. (used by all the IpAppChargingSession charging methods) \\
\hline
ChargingSessionID & TpSessionID & This element specifies the unique session ID for the charging session. (used by all the IpAppChargingSession charging methods) \\
\hline
RequestNumberFirstRequest & TpInt32 & This element specifies the request number to use for the next request. (used by all the IpAppChargingSession charging methods) \\
\hline
\end{tabular}
\end{table}

\textbf{Note:} The session ID is still returned by this message to ensure maximum possible OSA compatibility.

Charging Manager Parameters

\textbf{Introduction}

Here are the parameters used for this functionality.

\begin{itemize}
\item \textit{appChargingSession} \\
\textit{Description:} The reference to an IpAppChargingSession object.
\item \textit{appInterface} \\
\textit{Description:} The reference to an IpAppChargingManager object.
\item \textit{correlationID} \\
\textit{Description:} A TpCorrelationID value used to correlate the charging to network activity such as voice calls, data sessions, or multi-media sessions. This is included in the CDR (Type 20) that is generated by the OSA CHAM SCS gateway. Otherwise the parameter is ignored by this method. (See standard [5] for TpCorrelationID encoding.)
\end{itemize}
merchantAccount
Description: A TpMerchantAccountID structure composed of the merchant ID (string) and account ID (integer) to be included in all CDRS (Types 20 – 23) that is generated by the OSA CHAM SCS gateway. Otherwise the parameter is ignored by this method.

sessionDescription
Description: A string to be included in the CDR (Type 20) that is generated by the OSA CHAM SCS gateway. Otherwise the parameter is ignored by this method.

sessionID
Description: The ID of the charging session to be created by the OSA CHAM SCS.

user
Description: A TpAddress structure that is ignored by this method.
Overview

Introduction

This chapter explains the methods that are available using the IpAppChargingManager. The IpAppChargingManager object resides on the third-party client application and provides the following methods that can be invoked by the OSA CHAM SCS gateway.

In this chapter

This chapter contains the following topics.

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SessionAborted()</td>
<td>125</td>
</tr>
<tr>
<td>Application Charging Manager Parameters</td>
<td>125</td>
</tr>
</tbody>
</table>

SessionAborted()

Description

This method is invoked on the IpAppChargingManager to indicate an abnormal termination of a charging session. Charging methods can no longer be called using this charging session. The OSA CHAM SCS gateway will have removed this session from the internal table of active charging sessions. This method can only be invoked by OSA CHAM SCS gateway if the setCallback() method for the IpAccountManager has been previously invoked by the third-party client application.

Parameters

Here is the parameter for this method.

- sessionID (on page 125)

Application Charging Manager Parameters

Introduction

Here are the parameters used for this functionality.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sessionID</td>
<td>The ID of the charging session that has been aborted by the OSA CHAM SCS gateway.</td>
</tr>
</tbody>
</table>
Overview

Introduction

This chapter explains the methods available using the IpChargingSession. The IpChargingSession object resides on the OSA CHAM SCS gateway and provides the following methods that can be invoked by the third-party client application.

In this chapter

This chapter contains the following topics.

CreditAmountReq() 127
CreditUnitReq() 128
DebitAmountReq() 128
DebitUnitReq() 129
DirectCreditAmountReq() 129
DirectCreditUnitReq() 137
DirectDebitAmountReq() 143
DirectDebitUnitReq() 146
ExtendLifeTimeReq() 153
GetAmountLeft() 153
GetLifeTimeLeft() 154
GetUnitLeft() 154
RateReq() 155
Release() 157
ReserveAmountReq() 157
ReserveUnitReq() 160
Charging Session Parameters 166

CreditAmountReq()

Description

This method is invoked on the IpChargingSession to perform one of the following:

- This method is invoked on the IpAccountManager to credit monies towards an existing reservation already made on the OSA CHAM SCS gateway. The incoming request will specify the number of UBE small currency monies to be credited.
- This method is invoked on the IpAccountManager to credit time towards an existing reservation already made on the OSA CHAM SCS gateway. The incoming request will specify the number of seconds to be credited, the internal UBE time balance units is 100ths seconds so a conversion will be required.
- This method is invoked on the IpAccountManager to credit non cash or time towards an existing reservation already made on the OSA CHAM SCS gateway. The incoming request will specify the amount to be credited; the internal UBE units are mapped directly to the specified amount.
If a reservation does not exist for this user, or the reservation has expired then this request will fail. It is not possible to credit an amount towards a reservation unless debits greater to or equal to the credited amount have already been debited – it is not possible to credit a subscriber’s wallet through the reservation-confirmation process.

Parameters

Here is a list of the parameters for this method.

- `sessionID` (on page 168)
- `applicationDescription` (on page 166)
- `amount` (on page 166)
- `closeReservation` (on page 167)
- `requestNumber` (on page 167)

Returns

Nothing.

CreditUnitReq()

Description

This method is invoked on the IpAccountManager to credit units towards an existing reservation already made on the OSA CHAM SCS gateway. If a reservation does not exist for this user, or the reservation has expired, then this request will fail. It is not possible to credit units towards a reservation unless debits greater to or equal to the credited units have already been debited – it is not possible to credit a user’s account through the reservation-confirmation process.

Parameters

Here is a list of the parameters for this method.

- `sessionID` (on page 168)
- `applicationDescription` (on page 166)
- `volumes` (on page 168)
- `closeReservation` (on page 167)
- `requestNumber` (on page 167)

Returns

Nothing.

DebitAmountReq()

Description

This method will be invoked on the IpChargingSession to perform one of the following:

- This method will be invoked on the IpAccountManager to debit monies towards an existing reservation already made on the OSA CHAM SCS gateway. The incoming request will specify the number of UBE small currency monies to be debited.
This method will be invoked on the IpAccountManager to debit time towards an existing reservation already made on the OSA CHAM SCS gateway. The incoming request will specify the number of seconds to be debited, the internal UBE time balance units is 100ths seconds so a conversion will be required.

- This method will be invoked on the IpAccountManager to debit non cash or time towards an existing reservation already made on the OSA CHAM SCS gateway. The incoming request will specify the amount to be debited; the internal UBE units are mapped directly to the specified amount.
- If a reservation does not exist for this user, or the reservation has expired then this request will fail.

**Parameters**

Here is a list of the parameters for this method.

- `sessionID` (on page 168)
- `applicationDescription` (on page 166)
- `amount` (on page 166)
- `closeReservation` (on page 167)
- `requestNumber` (on page 167)

**Returns**

Nothing.

**DebitUnitReq()**

**Description**

This method will be invoked on the IpAccountManager to debit units from an existing reservation already made on the OSA CHAM SCS gateway. If a reservation does not exist for this user, or the reservation has expired then this request will fail.

**Parameters**

Here is a list of the parameters for this method.

- `sessionID` (on page 168)
- `applicationDescription` (on page 166)
- `volumes` (on page 168)
- `closeReservation` (on page 167)
- `requestNumber` (on page 167)

**Returns**

Nothing.

**DirectCreditAmountReq()**

**Description**

This method is invoked on the IpChargingSession to perform one of the following:
This method is invoked on the IpChargingSession to credit monies towards a subscriber's wallet using cash balance types only. The specified (or default) balance type or the specified balance type cascade (the UBE will credit the first balance type in the balance type cascade) is used. The UBE will create the specified balance type or first balance type in the balance type cascade if the balance type does not already exist in the subscriber's wallet. If the balance type cascade name is specified then the balance type name is ignored. The incoming request will specify the number of UBE small currency monies to be credited. Typically this method is used for balance types where the unit type of the balance type is 'Cash'; if the balance type cascade name is specified in the incoming request then only balance types where the unit type of the balance type is 'Cash' is available in the balance type cascade.

This method is invoked on the IpChargingSession to credit time towards a subscriber's wallet using time balance types. The incoming request will specify the number of seconds to be credited, the internal UBE time balance units is 100ths seconds so a conversion is required. The balance type name must be specified in the incoming request where the unit type of the balance type is 'Time' and the allowNonCashBalanceForAmountMethods (on page 18) configuration parameter must be 'true' for time-based reservations to be supported. If the balance type cascade name is specified in the incoming request then the amount specified is assumed to be monies and only balance types where the unit type of the balance type is 'Cash' is available in the balance type cascade.

This method is invoked on the IpChargingSession to credit an amount towards a subscriber's wallet using non-cash or time balance types. The incoming request will specify the amount to be credited; the internal UBE units are mapped directly to the specified amount. The balance type name must be specified in the incoming request where the unit type of the balance type is non 'Cash' or 'Time' and the allowNonCashBalanceForAmountMethods configuration parameter must be 'true' for non cash or time-based reservations to be supported. If the balance type cascade name is specified in the incoming request then the amount specified is assumed to be monies and only balance types where the unit type of the balance type is 'Cash' is available in the balance type cascade.

Extend the expiry date of the specified (or default) balance type by a specified number of days based on the rules detailed in Extending balance and wallet expiry dates (on page 131). The number of days is taken from the incoming P_CHS_BALANCE_VALIDITY_MODIFICATION parameter or if not specified and the PI_XDATE_EXTENSION table is installed then determine the appropriate balance extension from this table. If the number of days is not specified in the incoming P_CHS_BALANCE_VALIDITY_MODIFICATION parameter and the PI_XDATE_EXTENSION table is not installed then no default value is used. Specifying the balance type cascade is not supported in conjunction with specifying the balance expiry extension parameter. A configuration item (defaulting to 'false') will cause the request to fail if the balance extension parameter exceeds the wallet extension parameter or if the result of extending the balance expiry date exceeds the current or extended wallet expiry date; this expiry indicator is passed through to the UBE. The P_CHS_OVERRIDE_BALANCE_VALUE is not supported when the balance type cascade name is specified.

Extend the expiry date of the specified (or default) wallet type by a specified number of days based on the rules detailed in Extending balance and wallet expiry dates. The number of days is taken from the incoming P_CHS_WALLET_VALIDITY_MODIFICATION parameter or if not specified and the PI_YDATE_EXTENSION table is installed then determine the appropriate wallet extension from this table. If the number of days is not specified in the incoming P_CHS_WALLET_VALIDITY_MODIFICATION parameter and the PI_YDATE_EXTENSION table is not installed then no default value is used. A configuration item (defaulting to 'false') will cause the request to fail if the wallet extension parameter is less than the balance extension parameter or if the result of extending the wallet expiry date is less than the extended balance expiry date; this expiry indicator is passed through to the UBE. If the balance expiry date is not being extended then this check cannot be performed.

Set the value of the specified (or default) balance type to a specific value where the UBE will create the specified balance type if the balance type does not already exist in the subscriber's wallet. The incoming P_CHS_OVERRIDE_BALANCE_VALUE parameter will indicate the value supplied is an override and not to be used to credit an existing balance type; this override indicator is passed through to the UBE. The P_CHS_OVERRIDE_BALANCE_VALUE is not supported when the balance type cascade name is specified.
• Set the expiry date of the specified (or default) balance type to a specific date (value taken from the incoming P_CHS_BALANCE_VALIDITY_DATE parameter). A configuration item (defaulting to 'false') will cause the request to fail if the balance specific date parameter exceeds the wallet specific date parameter; this expiry indicator is passed through to the UBE.

• Set the expiry date of the specified (or default) wallet type to a specific date (value taken from the incoming P_CHS_WALLET_VALIDITY_DATE parameter). A configuration item (defaulting to 'false') will cause the request to fail if the wallet specific date parameter is less than the balance specific date parameter; this expiry indicator is passed through to the UBE.

• Recharge a wallet using a voucher (voucher number and voucher pin taken from the incoming P_CHS_VOUCHER parameter). If the EDR details contain both the BONUS_TYPE and REFERENCE tags then an applicable bonus value may be incorporated into the amount credited to subscriber's wallet subject to the rules detailed in Applying bonuses (on page 133).

• Recharge a wallet using a voucher type (voucher type name taken from the incoming P_CHS_VOUCHER_TYPE_NAME parameter). If the EDR details contain both the BONUS_TYPE and REFERENCE tags then an applicable bonus value may be incorporated into the amount credited to subscriber's wallet subject to the rules detailed in Applying bonuses.

• Bonuses may be applied to the value specified in the request; these will be calculated on the UBE using the P_CHS_BONUS_TYPE and P_CHS_REFERENCE parameters when crediting a subscriber's wallet using a specified value, recharging a subscriber using a voucher or recharging a subscriber using a voucher type. The bonus type and reference parameters will be passed through to the UBE.

If the specified balance type, or default balance type if none is specified, does not exist, then the balance type will be created; the expiry date for the existing or new balance type and for the wallet will be set based on the rules detailed in Extending balance and wallet expiry dates (on page 131) or Specifying balance and wallet expiry dates (on page 132) depending on which scenario applies.

If the balance type cascade is specified, then the first balance type in the cascade will be credited and if this balance type does not exist then the balance type will be created; the expiry date for the existing or new balance type and the wallet will be set based on the rules detailed in Extending balance and wallet expiry dates; specifying a specific balance date is not supported in conjunction with specifying a balance type cascade.

If the 'override balance wallet value' flag is specified and the balance type:

• already exists, then the specified balance type will have a single bucket value updated to this value (the bucket with the latest expiry date) and all remaining buckets in the balance type will be deleted; the expiry date for the balance type and wallet will be set based on the rules detailed in Extending balance and wallet expiry dates or Specifying balance and wallet expiry dates depending on which scenario applies. Specifying the balance type cascade is not supported in conjunction with specifying the 'override balance wallet value' flag. A configuration item (defaulting to 'false') will cause the request to fail if the specific balance date parameter exceeds the specific wallet date parameter; this expiry indicator will be passed through to the UBE.

• does not exist then the balance type will be created; the expiry date for the balance type and wallet will be set based on the rules detailed in Extending balance and wallet expiry dates or Specifying balance and wallet expiry dates depending on which scenario applies.

If the EDR details contain both the BONUS_TYPE and REFERENCE tags, then an applicable bonus value may be incorporated into the amount credited to subscriber's wallet subject to the rules detailed in Applying bonuses (on page 133).

The current reservation, if one exists, will not be affected through invoking this request.

Extending balance and wallet expiry dates

Updating existing balances
If the balance extensions parameter is specified and the 'override balance wallet value' flag is:

Chapter 11, Charging Session Functionality 131
not specified, then all the buckets in the balance type will have the expiry date extended by this number of days from today only if extending the expiry date results in extending the existing expiry date. If the result of extending the balance expiry date exceeds the result of extending the wallet expiry date then the balance expiry date will not be extended unless the 'allow wallet balance extension' flag is present in the incoming message.

- specified and the balance extensions parameter is specified then the bucket with the latest expiry date will have the expiry date extended by this number of days from today only if extending the expiry date results in extending the existing expiry date. If the result of extending the balance expiry date exceeds the result of extending the wallet expiry date then the balance expiry date will not be extended unless the 'allow wallet balance extension' flag is present in the incoming message.

Irrespective of whether the 'override balance wallet value' flag is specified, if the:

- bucket expiry date to be extended is set to NEVER_EXPIRES then the bucket expiry date will remain unchanged.
- wallet extensions parameter is specified then the wallet will have the expiry date extended by this number of days from today if the result of extending the current wallet expiry date results in extending the existing wallet expiry date. If the result of extending the wallet expiry date is less than the result of setting the balance expiry date then the wallet expiry date will not be extended unless the 'allow wallet balance extension' flag is present in the incoming message.

Creating new balances
If the balance extensions parameter is specified and the 'override balance wallet value' flag is:

- not specified, then the new balance type will have the expiry date set to this number of days from today irrespective of the wallet expiry date.
- is specified, then the new balance type will have the expiry date set to this number of days from today irrespective of the wallet expiry date.

Irrespective of whether the 'override balance wallet value' flag is specified:

- if the the balance extensions parameter is not specified then the new balance type will have the balance expiry date set to the number of days from today using the existing BE.beVWARS.createBucketExpiryDays configuration parameter (defaulting to 30 days); this is irrespective of the wallet expiry date.
- if the wallet extensions parameter is specified then the wallet will have the expiry date extended by this number of days from today if the result of extending the current wallet expiry date results in extending the existing wallet expiry date. If the result of extending the wallet expiry date is less than the result of setting the balance expiry date then the wallet expiry date will not be extended unless the 'allow wallet balance extension' flag is present in the incoming message.

Specifying balance and wallet expiry dates

Updating existing balances
If the 'override balance wallet value' flag is:

- not specified and the specific balance date parameter or specific wallet date parameter is specified, then this will return an error; specifying the specific balance date parameter or specific wallet date parameter is not supported without specifying the 'override balance wallet value'.
- is specified and the specific balance date parameter is specified, then the bucket with the latest expiry date will have the expiry date set to the specific balance date parameter irrespective of whether the new bucket expiry date is shortened or lengthened. If the result of setting the balance expiry date exceeds the result of setting the wallet expiry date then the balance expiry date will remain unchanged unless the 'allow wallet balance extension' flag is present in the incoming message.

If the 'override balance wallet value' flag is specified and:

- if the bucket expiry date to be set is currently set to NEVER_EXPIRES, then the bucket expiry date will be set to the specific balance date parameter.
the specific balance date is set to NEVER_EXPIRES (i.e. '0'), then the bucket expiry date will be set
to NEVER_EXPIRES.

- the specific wallet date parameter is specified, then the wallet will have the expiry date set to the
  specific wallet date parameter irrespective of whether the new wallet expiry date is shortened or
  lengthened. If the result of setting the wallet expiry date is less than the result of setting the balance
  expiry date then the wallet expiry date will remain unchanged unless the 'allow wallet balance
  extension' flag is present in the incoming message.

- and if the wallet expiry date to be set is currently set to NEVER_EXPIRES, then the wallet expiry
date will be set to the specific wallet date parameter.

- the specific wallet date is set to NEVER_EXPIRES (i.e. '0'), then the wallet expiry date will be set to
  NEVER_EXPIRES.

### Creating new balances

If the 'override balance wallet value' flag is not specified and the specific balance date parameter:

- or specific wallet date parameter is specified, then this will return an error; specifying the specific
  balance date parameter or specific wallet date parameter is not supported without specifying the
  'override balance wallet value'.

- is specified, then the new balance type will have the expiry date set to the specific balance date
  parameter irrespective of the wallet expiry date.

If the 'override balance wallet value' flag is specified and the specific balance date:

- is set to NEVER_EXPIRES (i.e. '0'), then the new balance type will have the expiry date set to
  NEVER_EXPIRES.

- parameter is not specified, then the new balance type will have the balance expiry date set to the
  number of days from today using the existing BE.beVWARS.createBucketExpiryDays configuration
  parameter (defaulting to 30 days); this is irrespective of the wallet expiry date.

If the 'override balance wallet value' flag is specified and the specific wallet:

- date parameter is specified, then the wallet will have the expiry date set to the specific wallet date
  parameter irrespective of whether the new wallet expiry date is shortened or lengthened. If the
  result of setting the wallet expiry date is less than the result of setting the balance expiry date then
  the wallet expiry date will remain unchanged unless the 'allow wallet balance extension' flag is
  present in the incoming message.

- expiry date to be set is currently set to NEVER_EXPIRES, then the wallet expiry date will be set to
  the specific wallet date parameter.

- date is set to NEVER_EXPIRES (i.e. '0'), then the wallet expiry date will be set to
  NEVER_EXPIRES.

### Applying bonuses

If the EDR details contain both the BONUS_TYPE and REFERENCE tags, then an attempt to match the
BONUS_TYPE, REFERENCE, service provider, triggering balance type and applicable bonus range to
the incoming EDR tags, recharge value and subscriber details will be made.

The triggering balance type will always have at least the recharge value credited to the balance type
however as explained below if a matching bonus configuration is found then an applicable bonus will
also be applied. If the triggering balance type is:

- found in the subscriber's wallet and if the promotional balance type also exists in the subscriber's
  wallet then the promotional balance type will be credited by the percentage of the recharge value as
  defined for the matching bonus definition.

- found in the subscriber's wallet, but the promotional balance type does not exist in the subscriber's
  wallet then the triggering balance type will be credited by the percentage of the recharge value as
  defined for the matching bonus definition.
not found in the subscriber's wallet then there will be no bonus applied.

Notes:

- The BONUS_TYPE value must be in upper case. For example CUSTOM.
- It is not possible to combine balance and wallet extensions with balance and wallet specific dates in the incoming request; combining these charging parameters will cause the request to fail.
- The subscriber's wallet may not be available on the local SCS database if the user is not a prepaid subscriber. For post-paid subscribers the user can be 'cold-billed' using the information extracted from the OSA CDR. If the MSISDN is not found when the chargedParty is decoded and the 'AllowPostPaid' flag is set to true, then the request will return success. Otherwise, if the chargedParty fails to decode successfully and the 'AllowPostPaid' flag is set to false, then the request will return failure.

Parameters

Here is a list of the parameters for this method.

- `sessionId` (on page 168)
- `applicationDescription` (on page 166)
- `chargingParameters` (on page 167)
- `amount` (on page 166)
- `requestNumber` (on page 167)

TpChargingParameter structure

The treatment and expected type of each element within each TpChargingParameter structure is:-

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChargedParty</td>
<td>TpString</td>
<td>Name: P_CHS_CHARGE_PARTY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mandatory.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ChargedType 1: The TpString will correspond to the MSISDN to charge.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ChargedType 3: The TpString will correspond to the wallet ID and billing engine ID to charge. The wallet ID and billing engine ID are separated by '/'.</td>
</tr>
<tr>
<td>ChargedType</td>
<td>TpInt32</td>
<td>Name: P_CHS_CHARGE_TYPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mandatory. Value showing type of ChargedParty.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 1 - MSISDN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 2 - reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 3 - Wallet ID</td>
</tr>
<tr>
<td>Discount Override</td>
<td>TpInt32</td>
<td>Name: P_CHS_DISCOUNT_OVERRIDE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Percentage discount to apply to final cost. Valid values are 0 – 10000 where these are in 1/100th of a percent. Therefore, 500 would equate to a 5% discount. The discount applied to the final cost will depend solely on the OSA discount entered here.</td>
</tr>
<tr>
<td>Wallet Type Name</td>
<td>TpString</td>
<td>Name: P_CHS_WALLET_TYPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional. The wallet type to use for Charge Type 1. If not specified, the default wallet type for this account is used.</td>
</tr>
<tr>
<td>Element Name</td>
<td>Element Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>---------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Balance Type Name</td>
<td>TpString</td>
<td>Optional. The balance type to use for Charge Type 1. If not specified, the default balance type for this account is used.</td>
</tr>
</tbody>
</table>
| Balance Type Cascade Name                        | TpString      | Optional. The balance type cascade name to use for Charge Type 1. If:  
  - specified then the balance type name will be ignored.  
  - not specified, the default balance type for this account will be used.  |
| Voucher Number and Voucher PIN                   | TpString      | Optional. The voucher number and voucher pin used when doing a voucher recharge.                                                            |
| Voucher Type Name                                | TpString      | Optional. The voucher type name used when doing a voucher type recharge.                                                                     |
| Bonus Type Name                                  | TpString      | Optional. The bonus type used in conjunction with the bonus reference to determine the applicable bonus to be applied to the value when crediting the subscriber’s wallet. A match will be done against the configured bonus in the database using the CCS_BONUS_TYPE.type column.  
  Valid value is 'C(U)STOM' where 'U' will be stored in the database.  
  Bonuses will be applied subject to the rules detailed in Applying bonuses (on page 133). |
| Reference                                         | TpString      | Optional. The bonus reference used in conjunction with the bonus type to determine the applicable bonus to be applied to the value when crediting the subscriber’s wallet. A match will be done against the configured bonus in the database using the CCS_BONUS_TYPE.component column.  
  This is a freeform text string; maximum of 20 characters.  
  Bonuses will be applied subject to the rules detailed in Applying bonuses (on page 133). |
| Override Balance Wallet Value and Dates Flag     | TpBoolean     | Optional. This can be set to 'TRUE' to override the value in the specified or default balance type with this amount; it cannot be used in conjunction with a balance type cascade name.  
  This can be used in conjunction with the balance and wallet validity date parameters to specify specific expiry dates and must be present to use either of the balance or wallet validity date parameters.  
  Defaults to 'FALSE'. |
<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Balance Validity Modification            | TpInt32      | Name: `P_CHS_BALANCE_VALIDITY_MODIFICATION`
Optional. The amount of days when extending the balance expiry date. The new balance balance expiry date will be set to the current date and time plus the number of days entered here. Only those expiry dates within the determined balance type (entered balance type or default balance type if none entered), which would be increased by this modification, will be affected. Any expiry date within the determined balance type that is set to ‘never expire’ will not be affected by this request.
The new balance expiry date will be set to this value irrespective of the current balance expiry date subject to the rules detailed in *Extending balance and wallet expiry dates* (on page 131).
| Wallet Validity Modification             | TpInt32      | Name: `P_CHS_WALLET_VALIDITY_MODIFICATION`
Optional. The amount of days when extending the wallet expiry date. The new wallet expiry date will be set to the current date and time plus the number of days entered here. The wallet expiry cannot be decreased by this modification. If the expiry date within the determined wallet type is set to ‘never expire’ it will not be affected by this request.
The new wallet expiry date will be set to this value irrespective of the current balance expiry date subject to the rules detailed in *Extending balance and wallet expiry dates* (on page 131).
| Balance Validity Date                    | TpDate       | Name: `P_CHS_BALANCE_VALIDITY_DATE`
Optional. The specific date when setting the balance expiry date. The new balance expiry date will be set to this value irrespective of the current balance expiry date subject to the rules detailed in *Specifying balance and wallet expiry dates* (on page 132).
The override balance wallet value parameter must be present for this parameter to be taken into account.
| Wallet Validity Date                     | TpDate       | Name: `P_CHS_WALLET_VALIDITY_DATE`
Optional. The specific date when setting the wallet expiry date. The new wallet expiry date will be set to this value irrespective of the current wallet expiry date subject to the rules detailed in *Specifying balance and wallet expiry dates* (on page 132).
The override balance wallet value parameter must be present for this parameter to be taken into account.
| Send SMS Flag                            | TpBoolean    | Name: `P_CHS_SEND_SMS_FLAG`
Optional. This can be set to ‘TRUE’ to ensure an SMS is sent when this reservation is confirmed on the billing engine (SMS only invoked if the ChargedType equals ‘1’, otherwise this parameter is ignored).
Defaults to ‘FALSE’.
Returns

Nothing.

**DirectCreditUnitReq()**

**Description**

This method is invoked on the IpChargingSession to perform one of the following:

- Credit units towards a subscriber’s wallet using the specified balance type or the specified balance type cascade (the UBE will credit the first balance type in the balance type cascade) where the UBE will create the specified balance type or first balance type in the balance type cascade if the balance type does not already exist in the subscriber’s wallet. If the balance type cascade name is specified, then the balance type name will be ignored. If neither is specified then the default balance type cascade already defined for either the rating definition or the billable-event definition will be used. Each unit has an associated cost on the billing engine and therefore a specified number of units translate into a fixed amount of monies to be credited towards the subscriber’s wallet. The only supported unit-based reservations are seconds and billable events.

- Extend the expiry date of the specified balance type by a specified number of days based on the rules detailed in *Extending balance and wallet expiry dates* (on page 131), albeit the ‘override balance wallet value’ flag is not supported for this request. The number of days will be taken from the incoming P_CHS_BALANCE_VALIDITY_MODIFICATION parameter. Specifying the balance type cascade is not supported in conjunction with specifying the balance expiry extension parameter. A configuration item (defaulting to ‘false’) will cause the request to fail if the balance extension parameter exceeds the wallet extension parameter, or if the result of extending the balance expiry date exceeds the current or extended wallet expiry date; this expiry indicator will be passed through to the UBE.

- Extend the expiry date of the specified (or default) wallet type by a specified number of days based on the rules detailed in *Extending balance and wallet expiry dates*, albeit the ‘override balance wallet value’ flag is not supported for this request. The number of days will be taken from the incoming P_CHS_WALLET_VALIDITY_MODIFICATION parameter. A configuration item (defaulting to ‘false’) will cause the request to fail if the wallet extension parameter is less than the balance extension parameter; this expiry indicator will be passed through to the UBE.

If the specified balance type does not exist then the balance type will be created; the expiry date for the existing or new balance type and for the wallet will be set based on the rules detailed in *Extending balance and wallet expiry dates*, albeit the ‘override balance wallet value’ flag is not supported for this request.

If the balance type cascade is specified then the first balance type in the cascade will be credited and if this balance type does not exist then the balance type will be created; the expiry date for the existing or new balance type and the wallet will be set based on the rules detailed in *Extending balance and wallet expiry dates*, albeit the ‘override balance wallet value’ flag is not supported for this request; specifying a specific balance date is not supported in conjunction with specifying a balance type cascade.

The current reservation, if one exists, will not be affected through invoking this request.

**Note:** The subscriber’s wallet may not be available on the local SCS database if the user is not a prepaid subscriber. For post-paid subscribers the user can be ‘cold-billed’ using the information extracted from the OSA CDR. If the MSISDN is not found when the chargeParty is decoded and the ‘AllowPostPaid’ flag is set to true and the rating tariff for this subscriber can be determined then the request will return success. Otherwise, if the chargeParty fails to decode successfully and the ‘AllowPostPaid’ flag is set to false and/or the rating tariff for this subscriber can not be determined, then the request will return failure.
Parameters

Here is a list of the parameters for this method.

- sessionID (on page 168)
- applicationDescription (on page 166)
- chargingParameters (on page 167)
- volumes (on page 168)
- requestNumber (on page 167)

Unit-Seconds based credit request

The treatment and expected type of each element within each TpChargingParameter structure for a unit-seconds based direct delta is:

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChargedParty</td>
<td>TpString</td>
<td>Name: P_CHS_CHARGE_PARTY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mandatory.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ChargedType 1: The TpString will correspond to the MSISDN to charge.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ChargedType 3: The TpString will correspond to the wallet ID and billing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>engine ID to charge. The wallet ID and billing engine ID is separated by</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'/'.</td>
</tr>
<tr>
<td>ChargedType</td>
<td>TpInt32</td>
<td>Name: P_CHS_CHARGE_TYPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mandatory. Value showing type of ChargedParty.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 1 - MSISDN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 2 - reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 3 - Wallet ID</td>
</tr>
<tr>
<td>Calling Number</td>
<td>TpString</td>
<td>Name: P_CHS_CALLING_NUMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mandatory. Network Calling Number to be used for rating-tariff calculations.</td>
</tr>
<tr>
<td>Called Number</td>
<td>TpString</td>
<td>Name: P_CHS_CALLED_NUMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mandatory. Network Called Number to be used for rating-tariff calculations.</td>
</tr>
<tr>
<td>Discount Override</td>
<td>TpInt32</td>
<td>Name: P_CHS_DISCOUNT_OVERRIDE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Percentage discount to apply to final cost. Valid values are 0 – 10000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>where these are in 1/100th of a percent. Therefore, 500 would equate to a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5% discount. The discount applied to the final cost will depend on the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'discountRuleType' parameter set in the CCS.ReservationHandler section of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>eserv.config file on the UBE. The following are valid values where the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>resulting discounted cost will be applied:</td>
</tr>
</tbody>
</table>
### Discount Rule Types

<table>
<thead>
<tr>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>DiscountRuleType = 'ServiceOverride' (use only OSA discount)</td>
<td>finalCost = initialCost * (OSA discount)</td>
</tr>
<tr>
<td>DiscountRuleType = 'S<em>R</em>W'</td>
<td>finalCost = initialCost * [OSA discount * CCS rating discount * CCS wallet discount]</td>
</tr>
<tr>
<td>DiscountRuleType = 'S+R+W'</td>
<td>finalCost = initialCost * [OSA discount + CCS rating discount + CCS wallet discount]</td>
</tr>
<tr>
<td>DiscountRuleType = 'S+R+W'</td>
<td>finalCost = initialCost * [OSA discount + CCS rating discount + CCS wallet discount]</td>
</tr>
<tr>
<td>DiscountRuleType = 'S+R+W'</td>
<td>finalCost = initialCost * [OSA discount + CCS rating discount + CCS wallet discount]</td>
</tr>
<tr>
<td>DiscountRuleType = 'S+R+W'</td>
<td>finalCost = initialCost * [OSA discount + CCS rating discount + CCS wallet discount]</td>
</tr>
<tr>
<td>DiscountRuleType = 'S+W*R'</td>
<td>finalCost = initialCost * [OSA discount + CCS wallet discount * CCS rating discount]</td>
</tr>
<tr>
<td>DiscountRuleType = 'S+W*R'</td>
<td>finalCost = initialCost * [OSA discount + CCS wallet discount * CCS rating discount]</td>
</tr>
<tr>
<td>DiscountRuleType = 'R+W*S'</td>
<td>finalCost = initialCost * [CCS rating discount + CCS wallet discount * OSA discount]</td>
</tr>
<tr>
<td>DiscountRuleType = 'R+W*S'</td>
<td>finalCost = initialCost * [CCS rating discount + CCS wallet discount * OSA discount]</td>
</tr>
</tbody>
</table>

### Wallet Type Name

<table>
<thead>
<tr>
<th>Wallet Type Name</th>
<th>TpString</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name: P_CHS_WALLET_TYPE</td>
<td>Optional. The wallet type name to use for Charge Type 1. If not specified, the default wallet type for this account is used.</td>
</tr>
<tr>
<td>Element Name</td>
<td>Element Type</td>
</tr>
<tr>
<td>------------------------------</td>
<td>--------------</td>
</tr>
</tbody>
</table>
| Balance Type Name            | TpString     | **Name:** P_CHS_BALANCE_TYPE  
Optional. The balance type name to use for Charge Type 1. If:  
- not specified, there will be no default balance type for this operation.  
- specified then a balance type cascade containing only this balance type will override the balance type cascade already defined for the rating definition. |
| Balance Type Cascade Name    | TpString     | **Name:** P_CHS_BALANCE_TYPE_CASCADE  
Optional. The balance type cascade name to use for Charge Type 1. If specified, this balance type cascade will override the balance type cascade already defined for the rating definition. |
| Balance Validity Modification| TpInt32      | **Name:** P_CHS_BALANCE_VALIDITY_MODIFICATION  
Optional. The number of days when extending the balance expiry date. The new balance expiry date will be set to the current date and time plus the number of days entered here. Only those expiry dates within the determined balance type (entered balance type), which would be increased by this modification, will be affected. Any expiry date within the determined balance type that is set to 'never expire' will not be affected by this request. The balance type parameter must be present for this parameter to be taken into account. |
| Wallet Validity Modification | TpInt32      | **Name:** P_CHS_WALLET_VALIDITY_MODIFICATION  
Optional. The number of days when extending the wallet expiry date. The new wallet expiry date will be set to the current date and time plus the number of days entered here. The wallet expiry cannot be decreased by this modification. If the expiry date within the determined wallet type is set to 'never expire', it will not be affected by this request. The balance type parameter must be present for this parameter to be taken into account. |
| Send SMS Flag                | TpBoolean    | **Name:** P_CHS_SEND_SMS_FLAG  
Optional. This can be set to 'TRUE' to ensure an SMS is sent when this reservation is confirmed on the billing engine (SMS only invoked if the ChargedType equals '1', otherwise this parameter is ignored). Defaults to 'FALSE'. |
**Unit-Billable event based credit request**

The treatment and expected type of each element within each TpChargingParameter structure for a unit-seconds based direct delta is:

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChargedParty</td>
<td>TpString</td>
<td>Name: P_CHS_CHARGE_PARTY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mandatory. ChargedType 1: The TpString will correspond to the MSISDN to charge.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ChargedType 3: The TpString will correspond to the wallet ID and billing engine ID to charge. The wallet ID and billing engine ID is separated by '/'.</td>
</tr>
<tr>
<td>ChargedType</td>
<td>TpInt32</td>
<td>Name: P_CHS_CHARGE_TYPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mandatory. Value showing type of ChargedParty.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 1 - MSISDN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 2 - reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- 3 - Wallet ID</td>
</tr>
<tr>
<td>Discount Override</td>
<td>TpInt32</td>
<td>Name: P_CHS_DISCOUNT_OVERRIDE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Percentage discount to apply to final cost. Valid values are 0 – 10000 where these are in 1/100th of a percent. Therefore, 500 would equate to a 5% discount. The discount applied to the final cost will depend on the OSA discount entered here.</td>
</tr>
<tr>
<td>Event Name</td>
<td>TpString</td>
<td>Name: P_CHS_EVENT_NAME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mandatory. Event Class and Name for rating tariff calculation. The TpString will correspond to the event class and event name to charge. The event class and event name are separated by '/'.</td>
</tr>
<tr>
<td>Wallet Type Name</td>
<td>TpString</td>
<td>Name: P_CHS_WALLET_TYPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional. The wallet type name to use for Charge Type 1. If not specified, the default wallet type for this account is used.</td>
</tr>
</tbody>
</table>

**Note:** CCS discounts do not apply to billable events.
### Balance Type Name
- **Element Name**: Balance Type Name
- **Element Type**: TpString
- **Description**: Name: P_CHS_BALANCE_TYPE
  - Optional. The balance type name to use for Charge Type 1. If:
    - not specified, there will be no default balance type for this operation.
    - specified then a balance type cascade containing only this balance type will override the balance type cascade already defined for the rating definition.

### Balance Type Cascade Name
- **Element Name**: Balance Type Cascade Name
- **Element Type**: TpString
- **Description**: Name: P_CHS_BALANCE_TYPE_CASCADE
  - Optional. The balance type cascade name to use for Charge Type 1. If specified, then this balance type cascade will override the balance type cascade already defined for the billable-event definition.

### Balance Validity Modification
- **Element Name**: Balance Validity Modification
- **Element Type**: TpInt32
- **Description**: Name: P_CHS_BALANCE_VALIDITY_MODIFICATION
  - Optional. The number of days when extending the balance expiry date. The new balance expiry date will be set to the current date and time plus the number of days entered here. Only those expiry dates within the determined balance type (entered balance type), which would be increased by this modification, will be affected. Any expiry date within the determined balance type that is set to ‘never expire’ will not be affected by this request.
  - The balance type parameter must be present for this parameter to be taken into account.

### Wallet Validity Modification
- **Element Name**: Wallet Validity Modification
- **Element Type**: TpInt32
- **Description**: Name: P_CHS_WALLET_VALIDITY_MODIFICATION
  - Optional. The number of days when extending the wallet expiry date. The new wallet expiry date will be set to the current date and time plus the number of days entered here. The wallet expiry cannot be decreased by this modification. If the expiry date within the determined wallet type is set to ‘never expire’, it will not be affected by this request.
  - The balance type parameter must be present for this parameter to be taken into account.

### Send SMS Flag
- **Element Name**: Send SMS Flag
- **Element Type**: TpBoolean
- **Description**: Name: P_CHS_SEND_SMS_FLAG
  - Optional. This can be set to ‘TRUE’ to ensure an SMS is sent when this reservation is confirmed on the billing engine (SMS only invoked if the ChargedType equals ‘1’, otherwise this parameter is ignored). Defaults to ‘FALSE’.
Returns
Nothing.

DirectDebitAmountReq()

Description
This method is invoked on the IpChargingSession to perform one of the following:

- This method is invoked on the IpChargingSession to debit monies from a subscriber's wallet using cash balance types only. The specified (or default) balance type or the specified balance type cascade is used. If the balance type cascade name is specified, then the balance type name will be ignored. If there are insufficient funds in the specified balance type or across all balance types in the balance type cascade, then based on a parameter in the incoming request, the balance type, or in the case of the balance type cascade, then the final balance type in the balance type cascade will be allowed to be taken negative; this 'allow negative balance' indicator will be passed through to the UBE. The incoming request specifies the number of UBE small currency monies to be credited. Typically this method is used for balance types where the unit type of the balance type is 'Cash'; if the balance type cascade name is specified in the incoming request, then only balance types where the unit type of the balance type is 'Cash' will be available in the balance type cascade.

- This method is invoked on the IpChargingSession to debit time towards a subscriber's wallet using time balance types. The incoming request specifies the number of seconds to be debited, the internal UBE time balance units is 100ths seconds so a conversion will be required. The balance type name must be specified in the incoming request where the unit type of the balance type is 'Time' and the allowNonCashBalanceForAmountMethods (on page 18) configuration parameter must be 'true' for time-based reservations to be supported. If the balance type cascade name is specified in the incoming request then the amount specified will be assumed to be monies and only balance types where the unit type of the balance type is 'Cash' will be available in the balance type cascade.

- This method is invoked on the IpChargingSession to debit an amount towards a subscriber's wallet using non-cash or time balance types. The incoming request specifies the amount to be debited; the internal UBE units are mapped directly to the specified amount. The balance type name must be specified in the incoming request where the unit type of the balance type is non 'Cash' or 'Time' and the allowNonCashBalanceForAmountMethods configuration parameter must be 'true' for non cash or time-based reservations to be supported. If the balance type cascade name is specified in the incoming request then the amount specified will be assumed to be monies and only balance types where the unit type of the balance type is 'Cash' will be available in the balance type cascade.

- Extend the expiry date of the specified (or default) balance type by a specified number of days based on the rules detailed in Extending balance and wallet expiry dates (on page 131), albeit the 'override balance wallet value' flag is not supported for this request. The number of days are taken from the incoming P_CHS_BALANCE_VALIDITY_MODIFICATION parameter. Specifying the balance type cascade is not supported in conjunction with specifying the balance expiry extension parameter. A configuration item (defaulting to 'false') will cause the request to fail if the balance extension parameter exceeds the wallet extension parameter or if the result of extending the balance expiry date exceeds the current or extended wallet expiry date; this expiry indicator will be passed through to the UBE.
Extend the expiry date of the specified (or default) wallet type by a specified number of days based on the rules detailed in *Extending balance and wallet expiry dates*, albeit the 'override balance wallet value' flag is not supported for this request. The number of days are taken from the incoming P_CHS_WALLET_VALIDITY_MODIFICATION parameter. A configuration item (defaulting to 'false') will cause the request to fail if the wallet extension parameter is less than the balance extension parameter or if the result of extending the wallet expiry date is less than the extended balance expiry date; this expiry indicator will be passed through to the UBE. If the balance expiry date is not being extended, then this check cannot be performed.

The specified balance type, or default balance type, if none is specified, will be debited and the expiry date for the balance type and for the wallet will be set based on the rules detailed in *Extending balance and wallet expiry dates*, albeit the 'override balance wallet value' flag is not supported for this request.

If the balance type cascade is specified, then the balances will be debited in order of the cascade; specifying a balance expiry extension is not supported in conjunction with specifying a balance type cascade.

The current reservation, if one exists, will not be affected through invoking this request.

**Note:** The subscriber's wallet may not be available on the local SCS database if the user is not a prepaid subscriber. For post-paid subscribers the user can be 'cold-billed' using the information extracted from the OSA CDR. If the MSISDN is not found when the chargedParty is decoded and the 'AllowDirectPostPaid' flag is set to 'true' then the request will return success. Otherwise, if the chargedParty fails to decode successfully and the 'AllowDirectPostPaid' flag is set to 'false', then the request will return failure.

**Parameters**

Here is a list of the parameters for this method.

- sessionID (on page 168)
- applicationDescription (on page 166)
- chargingParameters (on page 167)
- amount (on page 166)
- requestNumber (on page 167)

**TpChargingParameter structure**

The treatment and expected type of each element within each TpChargingParameter structure is:

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| ChargedParty   | TpString     | Name: P_CHS_CHARGE_PARTY  
Mandatory.  
ChargedType 1: The TpString will correspond to the MSISDN to charge.  
ChargedType 3: The TpString will correspond to the wallet ID and billing engine ID to charge. The wallet ID and billing engine ID are separated by '/'. |
| ChargedType    | TpInt32      | Name: P_CHS_CHARGE_TYPE  
Mandatory. Value showing type of ChargedParty.  
- 1 - MSISDN  
- 2 - reserved  
- 3 - Wallet ID |
<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discount Override</td>
<td>TpInt32</td>
<td>Name: P_CHS_DISCOUNT_OVERRIDE Percentage discount to apply to final cost. Valid values are 0 – 10000 where these are in 1/100th of a percent. Therefore, 500 would equate to a 5% discount. The discount applied to the final cost will depend solely on the OSA discount entered here.</td>
</tr>
<tr>
<td>Wallet Type Name</td>
<td>TpString</td>
<td>Name: P_CHS_WALLET_TYPE Optional. The wallet type name to use for Charge Type 1. If not specified, the default wallet type for this account is used.</td>
</tr>
<tr>
<td>Balance Type Name</td>
<td>TpString</td>
<td>Name: P_CHS_BALANCE_TYPE Optional. The balance type name or balance type cascade name to use for Charge Type 1. If not specified, the default balance type for this account will be used. To specify a balance cascade, a &quot;;&quot; is used to identify the parameter as a cascade. e.g. &quot;;promo_cascade&quot;. When a balance type cascade name is specified only balances containing 'Cash' Balance Units will be included in the result. The additional &quot;;&quot; is maintained when specifying the balance type cascade name for backwards compatibility only, if the balance type cascade parameter is specified separately then this parameter will take priority.</td>
</tr>
</tbody>
</table>
| Balance Type Cascade Name    | TpString     | Name: P_CHS_BALANCE_TYPE_CASCADE Optional. The balance type cascade name to use for Charge Type 1. If:  
- specified then the balance type name will be ignored.  
- not specified, the default balance type for this account will be used. |
<p>| Balance Validity Modification| TpInt32      | Name: P_CHS_BALANCE_VALIDITY_MODIFICATION Optional. The number of days when extending the balance expiry date. The new balance balance expiry date will be set to the current date and time plus the number of days entered here. Only those expiry dates within the determined balance type (entered balance type or default balance type if none entered), which would be increased by this modification, will be affected. Any expiry date within the determined balance type that is set to 'never expire' will not be affected by this request. |</p>
<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wallet Validity Modification</td>
<td>TpInt32</td>
<td>Name: P_CHS_WALLET_VALIDITY_MODIFICATION Optional. The number of days when extending the wallet expiry date. The new wallet expiry date will be set to the current date and time plus the number of days entered here. The wallet expiry cannot be decreased by this modification. If the expiry date within the determined wallet type is set to 'never expire', it will not be affected by this request.</td>
</tr>
<tr>
<td>Send SMS Flag</td>
<td>TpBoolean</td>
<td>Name: P_CHS_SEND_SMS_FLAG Optional. This can be set to 'TRUE' to ensure an SMS is sent when this reservation is confirmed on the billing engine (SMS only invoked if the ChargedType equals ‘1’, otherwise this parameter is ignored). Defaults to ‘FALSE’.</td>
</tr>
<tr>
<td>Allow Negative Balance Flag</td>
<td>TpBoolean</td>
<td>Name: P_CHS_ALLOW_NEGATIVE_BALANCE Optional. This can be set to ‘TRUE’ to set the 'IgnoreWalletBalance' flag in the UBE request message. Defaults to ‘FALSE’.</td>
</tr>
</tbody>
</table>

**Returns**

Nothing.

**DirectDebitUnitReq()**

**Description**

This method is invoked on the IpChargingSession to perform one of the following:

- Debit units from a subscriber's wallet using the specified balance type or the specified balance type cascade. If the balance type cascade name is specified, then the balance type name will be ignored. If there are insufficient funds in the specified balance type or across all balance types in the balance type cascade, then based on a parameter in the incoming request, the balance type, or in the case of the balance type cascade, then the final balance type in the balance type cascade will be allowed to be taken negative; this 'allow negative balance' indicator will be passed through to the UBE. If the 'allow negative balance' indicator is not set, for a seconds based charge if there is insufficient funds then the maximum number of seconds will be returned. Each unit has an associated cost on the billing engine and therefore a specified number of units translate into a fixed amount of monies to be debited from the subscriber's wallet. The only supported unit-based reservations are seconds and billable events.

- Extend the expiry date of the specified balance type by a specified number of days based on the rules detailed in *Extending balance and wallet expiry dates* (on page 131), albeit the 'override balance wallet value' flag is not supported for this request. The number of days will be taken from the incoming P_CHS_BALANCE_VALIDITY_MODIFICATION parameter. Specifying the balance type cascade is not supported in conjunction with specifying the balance expiry extension parameter. A configuration item (defaulting to 'false') will cause the request to fail if the balance extension parameter exceeds the wallet extension parameter, or if the result of extending the balance expiry date exceeds the current or extended wallet expiry date; this expiry indicator will be passed through to the UBE.
• Extend the expiry date of the specified (or default) wallet type by a specified number of days based on the rules detailed in *Extending balance and wallet expiry dates*, albeit the ‘override balance wallet value’ flag is not supported for this request. The number of days will be taken from the incoming P_CHS_WALLET_VALIDITY_MODIFICATION parameter. A configuration item (defaulting to ‘false’) will cause the request to fail if the wallet extension parameter is less than the balance extension parameter or if the result of extending the wallet expiry date is less than the extended balance expiry date; this expiry indicator will be passed through to the UBE. If the balance expiry date is not being extended, then this check cannot be performed.

The specified balance type will be debited and the expiry date for the balance type and for the wallet will be set based on the rules detailed in *Extending balance and wallet expiry dates*, albeit the ‘override balance wallet value’ flag is not supported for this request.

If the balance type cascade is specified, then the balances will be debited in order of the cascade; specifying a balance expiry extension is not supported in conjunction with specifying a balance type cascade.

The current reservation, if one exists, will not be affected through invoking this request.

**Note:** The subscriber’s wallet may not be available on the local SCS database if the user is not a prepaid subscriber. For post-paid subscribers the user can be ‘cold-billed’ using the information extracted from the OSA EDR. If the MSISDN is not found when the chargeParty is decoded and the ‘AllowPostPaid’ flag is set to true and the rating tariff for this subscriber can be determined, then the request will return success. Otherwise, if the chargeParty fails to decode successfully and the ‘AllowPostPaid’ flag is set to false and/or the rating tariff for this subscriber can not be determined, then the request will return failure.

**Parameters**

Here is a list of the parameters for this method.

- sessionID (on page 168)
- applicationDescription (on page 166)
- chargingParameters (on page 167)
- volumes (on page 168)
- requestNumber (on page 167)

**Unit-Seconds based debit request**

The treatment and expected type of each element within each TpChargingParameter structure for a unit-seconds based direct delta is:-

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| ChargedParty   | TpString     | Name: P_CHS_CHARGE_PARTY  
Mandatory.  
ChargedType 1: The TpString will correspond to the MSISDN to charge.  
ChargedType 3: The TpString will correspond to the wallet ID and billing engine ID to charge. The wallet ID and billing engine ID is separated by ‘/’. |
### ChargedType

**Element Name**: ChargedType  
**Element Type**: TpInt32  
**Description**: Name: P_CHS_CHARGE_TYPE  
Mandatory. Value showing type of ChargedParty.  
- 1 - MSISDN  
- 2 - reserved  
- 3 - Wallet ID

### Calling Number

**Element Name**: Calling Number  
**Element Type**: TpString  
**Description**: Name: P_CHS_CALLING_NUMBER  
Mandatory. Network Calling Number to be used for rating-tariff calculations.

### Called Number

**Element Name**: Called Number  
**Element Type**: TpString  
**Description**: Name: P_CHS_CALLED_NUMBER  
Mandatory. Network Called Number to be used for rating-tariff calculations.

### Discount Override

**Element Name**: Discount Override  
**Element Type**: TpInt32  
**Description**: Name: P_CHS_DISCOUNT_OVERRIDE  
Percentage discount to apply to final cost. Valid values are 0 – 10000 where these are in 1/100th of a percent. Therefore, 500 would equate to a 5% discount.  
The discount applied to the final cost will depend on the 'discountRuleType' parameter set in the CCS RESERVATION_HANDLER section of eserv.config file on the UBE. The following are valid values where the resulting discounted cost will be applied:

- **discountRuleType = 'ServiceOverride'**  
  (use only OSA discount)  
  \[
  \text{finalCost} = \text{initialCost} \times (\text{OSA discount})
  \]

- **discountRuleType = 'S*R*W'**  
  \[
  \text{finalCost} = \text{initialCost} \times [\text{OSA discount} \times \text{CCS rating discount} \times \text{CCS wallet discount}]
  \]

- **discountRuleType = 'S+R+W'**  
  \[
  \text{finalCost} = \text{initialCost} \times [\text{OSA discount} + \text{CCS rating discount} + \text{CCS wallet discount}]
  \]

- **discountRuleType = 'S+R*W'**  
  \[
  \text{finalCost} = \text{initialCost} \times [\text{OSA discount} + \text{CCS rating discount} \times \text{CCS wallet discount}]
  \]

- **discountRuleType = 'S*R+W'**  
  \[
  \text{finalCost} = \text{initialCost} \times [\text{OSA discount} \times \text{CCS rating discount} + \text{CCS wallet discount}]
  \]
### Table: Charging Session Functionality

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>discountRuleType = 'S+W*R'</td>
<td></td>
<td>finalCost = initialCost * [OSA discount + CCS wallet discount * CCS rating discount]</td>
</tr>
<tr>
<td>discountRuleType = 'S*W+R'</td>
<td></td>
<td>finalCost = initialCost * [OSA discount * CCS wallet discount + CCS rating discount]</td>
</tr>
<tr>
<td>discountRuleType = 'R+W*S'</td>
<td></td>
<td>finalCost = initialCost * [CCS rating discount + CCS wallet discount * OSA discount]</td>
</tr>
<tr>
<td>discountRuleType = 'R*W+S'</td>
<td></td>
<td>finalCost = initialCost * [CCS rating discount * CCS wallet discount + OSA discount]</td>
</tr>
</tbody>
</table>

#### Wallet Type Name

**Name:** P_CHS_WALLET_TYPE  
Optional. The wallet type name to use for Charge Type 1. If not specified, the default wallet type for this account is used.

#### Balance Type Name

**Name:** P_CHS_BALANCE_TYPE  
Optional. The balance type name to use for Charge Type 1. If:  
- not specified, there will be no default balance type for this operation.  
- specified then a balance type cascade containing only this balance type will override the balance type cascade already defined for the rating definition.

#### Balance Type Cascade Name

**Name:** P_CHS_BALANCE_TYPE_CASCADE  
Optional. The balance type cascade name to use for Charge Type 1. If specified, this balance type cascade will override the balance type cascade already defined for the rating definition.
<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Balance Validity Modification | TpInt32      | Name: P_CHS_BALANCE_VALIDITY_MODIFICATION
Optional. The number of days when extending the balance expiry date. The new balance expiry date will be set to the current date and time plus the number of days entered here. Only those expiry dates within the determined balance type (entered balance type), which would be increased by this modification, will be affected. Any expiry date within the determined balance type that is set to 'never expire' will not be affected by this request.
The balance type parameter must be present for this parameter to be taken into account. |
| Wallet Validity Modification | TpInt32      | Name: P_CHS_WALLET_VALIDITY_MODIFICATION
Optional. The number of days when extending the wallet expiry date. The new wallet expiry date will be set to the current date and time plus the number of days entered here. The wallet expiry cannot be decreased by this modification. If the expiry date within the determined wallet type is set to 'never expire', it will not be affected by this request.
The balance type parameter must be present for this parameter to be taken into account. |
| Send SMS Flag             | TpBoolean    | Name: P_CHS_SEND_SMS_FLAG
Optional. This can be set to 'TRUE' to ensure an SMS is sent when this reservation is confirmed on the billing engine (SMS only invoked if the ChargedType equals '1', otherwise this parameter is ignored). Defaults to 'FALSE'. |
| Allow Negative Balance Flag | TpBoolean    | Name: P_CHS_ALLOW_NEGATIVE_BALANCE
Optional. This can be set to 'TRUE' to set the 'IgnoreWalletBalance' flag in the UBE request message. Defaults to 'FALSE'.
## Unit-Billable event based debit request

The treatment and expected type of each element within each TpChargingParameter structure for a unit-billable based direct delta is:

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChargedParty</td>
<td>TpString</td>
<td>Name: P_CHS_CHARGE_PARTY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mandatory. The TpString will correspond to the MSISDN to charge.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ChargedType 1: The TpString will correspond to the wallet ID and billing engine ID to charge. The wallet ID and billing engine ID is separated by '/'.</td>
</tr>
<tr>
<td>ChargedType</td>
<td>TpInt32</td>
<td>Name: P_CHS_CHARGE_TYPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mandatory. Value showing type of ChargedParty.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 - MSISDN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 2 - reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 3 - Wallet ID</td>
</tr>
<tr>
<td>Discount Override</td>
<td>TpInt32</td>
<td>Name: P_CHS_DISCOUNT_OVERRIDE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Percentage discount to apply to final cost. Valid values are 0 – 10000 where these are in 1/100th of a percent. Therefore, 500 would equate to a 5% discount. The discount applied to the final cost will depend on the OSA discount entered here.</td>
</tr>
<tr>
<td>Event Name</td>
<td>TpString</td>
<td>Name: P_CHS_EVENT_NAME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mandatory. Event Class and Name for rating tariff calculation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The TpString will correspond to the event class and event name to charge. The event class and event name are separated by '/'.</td>
</tr>
<tr>
<td>Wallet Type Name</td>
<td>TpString</td>
<td>Name: P_CHS_WALLET_TYPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional. The wallet type name to use for Charge Type 1. If not specified, the default wallet type for this account is used.</td>
</tr>
<tr>
<td>Element Name</td>
<td>Element Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>--------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Balance Type Name                  | TpString     | Name: P_CHS_BALANCE_TYPE  
Optional. The balance type name to use for Charge Type 1. If:  
- not specified, there will be no default balance type for this operation.  
- specified then a balance type cascade containing only this balance type will override the balance type cascade already defined for the rating definition. |
| Balance Type Cascade Name          | TpString     | Name: P_CHS_BALANCE_TYPECASCADE  
Optional. The balance type cascade name to use for Charge Type 1. If specified, then this balance type cascade will override the balance type cascade already defined for the billable-event definition. |
| Balance Validity Modification      | TpInt32      | Name: P_CHS_BALANCE_VALIDITY_MODIFICATION  
Optional. The number of days when extending the balance expiry date. The new balance expiry date will be set to the current date and time plus the number of days entered here. Only those expiry dates within the determined balance type (entered balance type), which would be increased by this modification, will be affected. Any expiry date within the determined balance type that is set to ‘never expire’ will not be affected by this request.  
The balance type parameter must be present for this parameter to be taken into account. |
| Wallet Validity Modification       | TpInt32      | Name: P_CHS_WALLET_VALIDITY_MODIFICATION  
Optional. The number of days when extending the wallet expiry date. The new wallet expiry date will be set to the current date and time plus the number of days entered here. The wallet expiry cannot be decreased by this modification. If the expiry date within the determined wallet type is set to ‘never expire’, it will not be affected by this request.  
The balance type parameter must be present for this parameter to be taken into account. |
| Send SMS Flag                      | TpBoolean    | Name: P_CHS_SEND_SMS_FLAG  
Optional. This can be set to ‘TRUE’ to ensure an SMS is sent when this reservation is confirmed on the billing engine (SMS only invoked if ChargedType equals ‘1’, otherwise this parameter is ignored). Defaults to ‘FALSE’. |
### Allow Negative Balance Flag

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Allow Negative Balance Flag   | TpBoolean    | Name: P_CHS_ALLOW_NEGATIVE_BALANCE  
Optional. This can be set to 'TRUE' to set the 'IgnoreWalletBalance' flag in the UBE request message. Defaults to 'FALSE'. |

### ExtendLifeTimeReq()

**Description**

This method is invoked on the IpAccountManager to extend the lifetime of the reservation on both the OSA CHAM SCS gateway and on the billing engine for amount-based reservations only. The reservation will be extended by the value currently stored on the OSA CHAM SCS gateway entered as the 'ReservationLifeTimeHint' parameter of the TpChargingParameter structure for this reservation.

**Parameters**

Here is the parameter for this method.
- *sessionID* (on page 168)

**Returns**

Nothing.

### GetAmountLeft()

**Description**

This method is invoked on the IpAccountManager to retrieve the amount stored in the internal table on the OSA CHAM SCS gateway remaining for this reservation. This method can only be invoked for amount-based reservations.

**Parameters**

Here is the parameter for this method.
- *sessionID* (on page 168)
Returns

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TpChargingPrice</td>
<td>This is a structure containing the currency and a TpAmount structure. The currency is returned by the OSA CHAM SCS gateway as an empty string. (See standard [5] for TpAmount encoding. The 'Exponent' of the TpAmount structure will always be set to '0' and the 'Number' of the TpAmount structure will always be the number of small units of currency where the billing engine system currency is used).</td>
</tr>
</tbody>
</table>

GetLifeTimeLeft()

Description

This method is invoked on the IpAccountManager to retrieve the lifetime remaining on the OSA CHAM SCS gateway for this reservation. This method can be invoked for both amount-based and unit-based reservations.

Parameters

Here is the parameter for this method.

- sessionID (on page 168)

Returns

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TpInt32</td>
<td>An unsigned integer corresponding to the number of seconds that this reservation will remain active.</td>
</tr>
</tbody>
</table>

GetUnitLeft()

Description

This method is invoked on the IpAccountManager to retrieve the units stored in the internal table on the OSA CHAM SCS gateway remaining for this reservation. This method can only be invoked for unit-based reservations.

Parameters

Here is the parameter for this method.

- sessionID (on page 168)
Returns

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TpVolumeSet</td>
<td>A TpVolumeSet structure containing a single TpVolume which is composed of a</td>
</tr>
<tr>
<td></td>
<td>TpUnitID value and a TpAmount structure. The TpUnitID value must be either</td>
</tr>
<tr>
<td></td>
<td>P_CHS_UNIT_NUMBER (named events), or P_CHS_UNIT_SECONDS (time-based in</td>
</tr>
<tr>
<td></td>
<td>seconds). (See standard [5] for TpAmount encoding. The ‘Exponent’ of the</td>
</tr>
<tr>
<td></td>
<td>TpAmount structure will always be set to ‘0’ and the ‘Number’ of the TpAmount</td>
</tr>
<tr>
<td></td>
<td>structure will always be the number of units (named events or seconds) ).</td>
</tr>
</tbody>
</table>

RateReq()

Description

This method is invoked on the IpChargingManager to determine the individual rates that are associated with this user’s account. The rates have the associated discount applied prior to returning them to the third-party client application. This method can only be invoked for unit-based reservations.

Parameters

Here is a list of the parameters for this method.

- `sessionId` (on page 168)
- `chargingParameters` (on page 167)

TpChargingParameter structure

The treatment and expected type of each element within each TpChargingParameter structure for rate request is:-

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChargedParty</td>
<td>TpString</td>
<td>Name: P_CHS_CHARGE_PARTY&lt;br&gt;ChargedType 1: The TpString will correspond to the MSISDN to charge.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ChargedType 3: The TpString will correspond to the wallet ID and billing engine ID to charge.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The wallet ID and billing engine ID is separated by ‘/’.</td>
</tr>
<tr>
<td>ChargedType</td>
<td>TpInt32</td>
<td>Name: P_CHS_CHARGE_TYPE&lt;br&gt;Mandatory. Value showing type of ChargedParty.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 - MSISDN&lt;br&gt;• 2 - reserved&lt;br&gt;• 3 - Wallet ID</td>
</tr>
<tr>
<td>Calling Number</td>
<td>TpString</td>
<td>Name: P_CHS_CALLING_NUMBER&lt;br&gt;Optional. Network Calling Number to be used for rating-tariff</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calculations</td>
</tr>
<tr>
<td>Element Name</td>
<td>Element Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Called Number          | TpString     | **Name:** P_CHS_CALLED_NUMBER  
Optional. Network Called Number to be used for rating-tariff calculations. |
| Discount Override      | TpInt32      | **Name:** P_CHS_DISCOUNT OVERRIDE  
Percentage discount to apply to final cost.  
Valid values are 0 – 10000 where these are in 1/100th of a percent.  
Therefore, 500 would equate to a 5% discount.  
The discount applied to the final cost will depend on the ‘overrideDiscountType’ parameter set in the CCS.ReservationHandler section of eserv.config file found on the BE.  
The following are valid values where the resulting discounted cost will be applied:  
**Note:** CCS discounts do not apply to named events  
- overrideDiscountType = ‘Override’ (ignore OSA discount)  
- finalCost = initialCost * (CCS discount for seconds only)  
- overrideDiscountType = ‘Compound’  
finalCost = initialCost * [CCS discount for seconds only * OSA discount]  
- overrideDiscountType = ‘Cumulative’  
finalCost = initialCost * [CCS discount for seconds only + OSA discount]  |
| Event Name             | TpString     | **Name:** P_CHS_EVENT_NAME  
Optional. Event Class and Name for rating tariff calculation.  
The TpString will correspond to the event class and event name to charge. The event class and event name will be separated by ‘/’. |
| Wallet Type Name       | TpString     | **Name:** P_CHS_WALLET_TYPE  
Optional. The wallet type to use for Charge Type 1. If not specified, the default wallet type for this account is used. |

**Returns**

Nothing.
Release()

Description

This method will be invoked on the IpAccountManager to release the specified charging session. This will result in the following events on the OSA CHAM SCS gateway:

1. Any amounts (monies or units) that have been debited from the current reservation will be confirmed on the billing engine.
2. Any amount remaining in the current reservation will be returned to the user’s account and be available for alternative reservations from the third-party client application.
3. If any funds are confirmed on the billing engine during this transaction then an OSA CDR (Type 21) will be created, and if the current reservation is unit-seconds based then a CCS CDR (Type 24) will be created, otherwise, if the current reservation is unit-named event based, then a CCS CDR (Type 25) will be created.
4. If no funds are confirmed on the billing engine during this transaction then an OSA CDR (Type 22) will be created and there will be no corresponding CCS CDRs created.
5. If any funds are confirmed on the billing engine during this transaction, then a SMS may be sent to the user with the confirmation details of this transaction.
6. The charging session on the OSA CHAM SCS gateway will be removed the internal table of active sessions. This charging session will no longer be valid for any requests from the third-party client application.

Parameters

Here is a list of the parameters for this method.

- sessionID (on page 168)
- requestNumber (on page 167)

Returns

Nothing.

ReserveAmountReq()

Description

This method is invoked on the IpChargingSession to perform one of the following:

- This method is invoked on the IpChargingSession to make an initial monies-based reservation. A monies-based reservation will reserve monies from the subscriber’s wallet using cash balance types only and store this amount in an internal table on the OSA CHAM SCS gateway to be debited and/or credited using the credit(debit)AmountReq() methods. The incoming request will specify the number of UBE small currency monies to be reserved. Typically this method is used for balance types where the unit type of the balance type is ‘Cash’; if the balance type cascade name is specified in the incoming request then only balance types where the unit type of the balance type is ‘Cash’ will be available in the balance type cascade.
- This method is invoked on the IpChargingSession to make an initial time-based reservation.
A time-based reservation will reserve time from the subscriber's wallet using time balance types only and store this amount in an internal table on the OSA CHAM SCS gateway to be debited and/or credited using the credit(debit)AmountReq() methods. The incoming request will specify the number of seconds to be reserved. The internal UBE time balance units is 100ths seconds so a conversion will be required. The balance type name must be specified in the incoming request where the unit type of the balance type is 'Time' and the allowNonCashBalanceForAmountMethods (on page 18) configuration parameter must be 'true' for time-based reservations to be supported. If the balance type cascade name is specified in the incoming request then the amount specified will be assumed to be monies and only balance types where the unit type of the balance type is 'Cash' will be available in the balance type cascade.

- This method is invoked on the IpChargingSession to make an initial non-cash or time-based reservation.

A non-cash or time-based reservation will reserve time from the subscriber's wallet using non cash or time balance types only and store this amount in an internal table on the OSA CHAM SCS gateway to be debited and/or credited using the credit(debit)AmountReq() methods. The incoming request will specify the amount to be reserved; the internal UBE units are mapped directly to the specified amount. The balance type name must be specified in the incoming request where the unit type of the balance type is none 'Cash' or 'Time' and the allowNonCashBalanceForAmountMethods configuration parameter must be 'true' for non-cash or time-based reservations to be supported. If the balance type cascade name is specified in the incoming request then the amount specified will be assumed to be monies and only balance types where the unit type of the balance type is 'Cash' will be available in the balance type cascade.

- Prior to making a confirmation against the reserved amount it is necessary to have debited an amount from the reservation. It is not possible to credit an amount towards a reservation unless prior debits greater to or equal to the credited amount have already been made - it is not possible to credit a subscriber's wallet through the reservation-confirmation process. If a reservation already exists for this user on the OSA CHAM SCS gateway, then invoking this method will result in a subsequent reservation of the specified amount for this user on the billing engine and incrementing the amount stored in this reservation on the OSA CHAM SCS gateway. If an existing unit-based reservation already exists for this user on the OSA CHAM SCS gateway then this request will fail, i.e. reservation types cannot be mixed within an active reservation. If the amount-based reservation is first closed, then a unit-based reservation can be created within the same charging session. When a reservation is attempted by a subscriber on the OSA_CHAM_SCS this reservation may fail if the maximum concurrent access to the subscriber's wallet would be exceeded through successfully creating this reservation. This maximum concurrent access parameter is a current property of the subscriber's wallet and is used by reservation attempts of all types on the subscriber's wallet, i.e. voice calls, other amount-based or unit-based reservations through the OSA_CHAM SCS or billable event reservations using other applications. The amount of the reservation returned to the client will not be affected by the maximum concurrent access parameter; the OSA CHAM SCS will not divide the amount returned to the client by the maximum concurrent access parameter.

**Parameters**

Here is a list of the parameters for this method.

- sessionID (on page 168)
- applicationDescription (on page 166)
- chargingParameters (on page 167)
- preferredAmount (on page 167)
- minimumAmount (on page 167)
- requestNumber (on page 167)
### TpChargingParameter structure

The treatment and expected type of each element within each TpChargingParameter structure is:

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| ChargedParty       | TpString     | **Name:** P_CHS_CHARGE_PARTY  
Mandatory.  
ChargedType 1: The TpString will correspond to the MSISDN to charge.  
ChargedType 3: The TpString will correspond to the wallet ID and billing engine ID to charge. The wallet ID and billing engine ID are separated by '/'.
| ChargedType        | TpInt32      | **Name:** P_CHS_CHARGE_TYPE  
Mandatory. Value showing type of ChargedParty.  
- 1 - MSISDN  
- 2 - reserved  
- 3 - Wallet ID |
| Discount Override  | TpInt32      | **Name:** P_CHS_DISCOUNT_OVERRIDE  
Percentage discount to apply to final cost. Valid values are 0 – 10000 where these are in 1/100th of a percent.  
Therefore, 500 would equate to a 5% discount.  
The discount applied to the final cost will depend solely on the OSA discount entered here. |
| ReservationLifeTimeHint | TpInt32 | **Name:** P_CHS_RESERVATION_LIFETIME_HINT  
Optional. The lifetime of the reservation in seconds -set on both the OSA CHAM SCS gateway and the billing engine server. If the reservation times out on both the SCS and the BE, outstanding debits are not confirmed on the billing engine. The value entered must be greater than '0'. If not specified, then the default lifetime for the reservation will be '30' seconds.  
An external client should not set this value to more than 24 hours to ensure that a series of outstanding concurrent reservations would not accumulate after a period of network instability. The client should set this value to the period normally associated with use period of the charge. For example charging for a bus ticket maybe instant, but a movie download may take a period of hours.  
For a subsequent reservation request, if the reservation lifetime is set to '0', then the reservation will be extended by the 'ReservationLifeTimeHint' currently stored on the OSA CHAM SCS gateway from the initial reservation request. |
| Wallet Type Name   | TpString     | **Name:** P_CHS_WALLET_TYPE  
Optional. The wallet type to use for Charge Type 1.  
If not specified, the default wallet type for this account is used. |
<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Balance Type Name</td>
<td>TpString</td>
<td>Name: P_CHS_BALANCE_TYPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional. The balance type to use for Charge Type 1. If not specified, the default balance type for this account is used.</td>
</tr>
<tr>
<td>Balance Type Cascade Name</td>
<td>TpString</td>
<td>Name: P_CHS_BALANCE_TYPE CASCADE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional. The balance type cascade name to use for Charge Type 1. If:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• specified then the balance type name will be ignored.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• not specified, the default balance type for this account will be used.</td>
</tr>
<tr>
<td>Send SMS Flag</td>
<td>TpBoolean</td>
<td>Name: P_CHS_SEND_SMS_FLAG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional. This can be set to ‘TRUE’ to ensure an SMS is sent when this reservation is confirmed on the billing engine (SMS only invoked if the ChargedType equals ‘1’, otherwise this parameter is ignored). Defaults to ‘FALSE’.</td>
</tr>
</tbody>
</table>

**Returns**

Nothing.

**ReserveUnitReq()**

**Description**

This method is invoked on the IpChargingSession to make an initial unit-based reservation. Each unit has an associated cost on the billing engine and therefore a specified number of units translate into a fixed amount of monies against which the reservation can be made on the subscriber’s wallet. The only supported unit-based reservations are seconds and named events, where unit type cannot be changed during the lifetime of a reservation.

For unit-based reservation the OSA CHAM SCS gateway has no knowledge of monies and deals solely with the unit of the reservation. The number of units reserved is stored in an internal table on the OSA CHAM SCS gateway to be debited and/or credited using the credit/debitUnitReq() methods. Prior to making a confirmation against the reserved units it is necessary to have debited units from the reservation. It is not possible to credit units towards a reservation unless prior debits greater to or equal to the credited units have already been made - it is not possible to credit a subscriber’s wallet through the reservation-confirmation process.

If a reservation already exists for this user on the OSA CHAM SCS gateway, then invoking this method will result in a subsequent reservation of units for this user on the billing engine and incrementing the number of units stored in this reservation on the OSA CHAM SCS gateway. If an existing amount-based reservation already exists for this user on the OSA CHAM SCS gateway then this request will fail i.e. reservation types cannot be mixed within an active reservation. If the unit-based reservation is first closed, then an amount-based reservation can be created within the same charging session.
When a reservation is attempted by a subscriber on the OSA_CHAM_SCS this reservation may fail if the maximum concurrent access to the subscriber's wallet would be exceeded through successfully creating this reservation. This maximum concurrent access parameter is a current property of the subscriber's wallet and is used by reservation attempts of all types on the subscriber's wallet, i.e. voice calls, other amount-based or unit-based reservations through the OSA_CHAM SCS or billable-event reservations using other applications. The amount of the reservation returned to the client may be affected by the maximum concurrent access parameter in the same manner as currently seen for voice calls or billable-event reservations as follows: the amount returned for voice calls or reserve-unit requests for seconds-based reservations will be affected by the values configured on the UBE for the 'CCS.reservationHandler.maxReservationLength' and 'CCS.reservationHandler.greedyReservationLengthLimit' where the reservation length returned will be recalculated as the lesser of the maxReservationLength or the amount that the subscriber can afford if the greedyReservationLengthLimit is not initially satisfied.

**Parameters**

Here is a list of the parameters for this method.

- `sessionId` (on page 168)
- `applicationDescription` (on page 166)
- `chargingParameters` (on page 167)
- `volumes` - see Unit-Seconds Based Reservation (see "Parameters" on page 161) and Unit-Named Event Based Reservation (see "Parameters" on page 164).
- `requestNumber` (on page 167)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>volumes</td>
<td>a TpVolumeSet structure containing a single TpVolume which is composed of a TpUnitID value and a TpAmount structure. The TpUnitID value is P_CHS_UNIT_SECONDS. (See standard [5] for TpAmount encoding. The 'Exponent' of the TpAmount structure will always be set to '0' and the 'Number' of the TpAmount structure will always be ignored by this reservation). The 'Number' field of the TpAmount structure must be greater than zero.</td>
</tr>
</tbody>
</table>

The treatment and expected type of each element within each TpChargingParameter structure for a unit-seconds based reservation is:-

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChargedParty</td>
<td>TpString</td>
<td>Name: P_CHS_CHARGE_PARTY Mandatory. ChargedType 1: The TpString will correspond to the MSISDN to charge. ChargedType 3: The TpString will correspond to the wallet ID and billing engine ID to charge. The wallet ID and billing engine ID is separated by '/'.</td>
</tr>
<tr>
<td>Element Name</td>
<td>Element Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| ChargedType         | TpInt32      | Name: P_CHS_CHARGE_TYPE  
Mandatory. Value showing type of ChargedParty.  
- 1 - MSISDN  
- 2 - reserved  
- 3 - Wallet ID |
| Calling Number      |TpString      | Name: P_CHS_CALLING_NUMBER  
Mandatory. Network Calling Number to be used for rating-tariff calculations. |
| Called Number       |TpString      | Name: P_CHS_CALLED_NUMBER  
Mandatory. Network Called Number to be used for rating-tariff calculations. |
| Discount Override   |TpInt32       | Name: P_CHS_DISCOUNT_OVERRIDE  
Percentage discount to apply to final cost.  
Valid values are 0 – 10000 where these are in 1/100th of a percent.  
Therefore, 500 would equate to a 5% discount.  
The discount applied to the final cost will depend on the 'discountRuleType' parameter set in the CCS.ReservationHandler section of eserv.config file on the UBE. The following are valid values where the resulting discounted cost will be applied:  
- discountRuleType = 'ServiceOverride'  
  (use only OSA discount)  
  \[\text{finalCost} = \text{initialCost} \times (\text{OSA discount})\]
- discountRuleType = 'S*R*W'  
  \[\text{finalCost} = \text{initialCost} \times [\text{OSA discount} \times \text{CCS rating discount} \times \text{CCS wallet discount}]\]
- discountRuleType = 'S+R+W'  
  \[\text{finalCost} = \text{initialCost} \times [\text{OSA discount} + \text{CCS rating discount} + \text{CCS wallet discount}]\]
- discountRuleType = 'S+R+W'  
  \[\text{finalCost} = \text{initialCost} \times [\text{OSA discount} + \text{CCS rating discount} + \text{CCS wallet discount}]\]
### Element Name | Element Type | Description
--- | --- | ---
--- | --- | ---
ReservationLifeTimeHint | TpInt32 | Optional. The lifetime of the reservation in seconds - set on both the OSA CHAM SCS gateway and the billing engine server. A hint is specified as the billing server may choose to ignore this lifetime. If the reservation times out on both the SCS and the BE, outstanding debits are not confirmed on the billing engine. If not specified, then the default lifetime passed to the billing engine for the reservation is ‘30’ seconds.

For an initial reservation, the reservation lifetime is set to the number of units (seconds) that the billing engine returns in the acknowledgement (this will not be the number of units specified in the volumes parameter of the original request as this value is ignored). If the lifetime requested is less than the billing engine minimum call length then the acknowledgement will return a reservation of zero. This will close the session immediately.

- **discountRuleType = 'S*R+W'**
  \[
  \text{finalCost} = \text{initialCost} \times [\text{OSA discount} \times \text{CCS rating discount} + \text{CCS wallet discount}]
  \]

- **discountRuleType = 'S+W*R'**
  \[
  \text{finalCost} = \text{initialCost} \times [\text{OSA discount} + \text{CCS wallet discount} \times \text{CCS rating discount}]
  \]

- **discountRuleType = 'S*W+R'**
  \[
  \text{finalCost} = \text{initialCost} \times [\text{OSA discount} \times \text{CCS wallet discount} + \text{CCS rating discount}]
  \]

- **discountRuleType = 'R+W*S'**
  \[
  \text{finalCost} = \text{initialCost} \times [\text{CCS rating discount} + \text{CCS wallet discount} \times \text{OSA discount}]
  \]

- **discountRuleType = 'R*W+S'**
  \[
  \text{finalCost} = \text{initialCost} \times [\text{CCS rating discount} \times \text{CCS wallet discount} + \text{OSA discount}]
  \]
Chapter 11

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>For a subsequent reservation request, this parameter is ignored and the reservation lifetime is extended by the number of units (seconds) that the billing engine returns in the acknowledgement for the (this will not be the number of units specified in the volumes parameter of the original request as this value is ignored).</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Wallet Type Name</th>
<th>TpString</th>
<th>Name: P_CHS_WALLET_TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional. The wallet type name to use for Charge Type 1. If not specified, the default wallet type for this account is used.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Send SMS Flag</th>
<th>TpBoolean</th>
<th>Name: P_CHS_SEND_SMS_FLAG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional. This can be set to ‘TRUE’ to ensure an SMS is sent when this reservation is confirmed on the billing engine (SMS only invoked if the ChargedType equals ‘1’, otherwise this parameter is ignored). Defaults to ‘FALSE’</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>volumes</td>
<td>A TpVolumeSet structure containing a single TpVolume which is composed of a TpUnitID value and a TpAmount structure. The TpUnitID value must be P_CHS_UNIT_NUMBER. (See standard [5] for TpAmount encoding. The ‘Exponent’ of the TpAmount structure will always be set to ‘0’ and the ‘Number’ of the TpAmount structure will always be the number of named events to be used for this reservation). The ‘Number’ field of the TpAmount structure must be greater than zero.</td>
</tr>
</tbody>
</table>

The treatment and expected type of each element within each TpChargingParameter structure for a unit-named event based reservation is:-

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChargedParty</td>
<td>TpString</td>
<td>Name: P_CHS_CHARGE_PARTY</td>
</tr>
<tr>
<td>Mandatory. ChargedType 1: The TpString will correspond to the MSISDN to charge. ChargedType 3: The TpString will correspond to the wallet ID and billing engine ID to charge. The wallet ID and billing engine ID is separated by ‘/’.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ChargedType</th>
<th>TpInt32</th>
<th>Name: P_CHS_CHARGE_TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mandatory. Value showing type of ChargedParty.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>● 1 - MSISDN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>● 2 - reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>● 3 - Wallet ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Element Name</td>
<td>Element Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Discount Override</td>
<td>TpInt32</td>
<td>Percentage discount to apply to final cost. Valid values are 0 – 10000 where these are in 1/100th of a percent. Therefore, 500 would equate to a 5% discount. The discount applied to the final cost will depend solely on the OSA discount entered here. <strong>Note:</strong> CCS discounts do not apply to billable events.</td>
</tr>
<tr>
<td>Event Name</td>
<td>TpString</td>
<td>Mandatory. Event Class and Name for rating tariff calculation. The TpString will correspond to the event class and event name to charge. The event class and event name are separated by '/'.</td>
</tr>
<tr>
<td>ReservationLifeTimeHint</td>
<td>TpInt32</td>
<td>Optional. The lifetime of the reservation in seconds - set on both the OSA CHAM SCS gateway and the billing engine server. A hint is specified as the billing server may choose to ignore this lifetime. If the reservation times out on both the SCS and the BE, outstanding debits are not confirmed on the billing engine. If not specified, then the default lifetime passed to the billing engine for the reservation is '30' seconds. For an initial reservation, if the value entered is less than '0', then the default lifetime requested from the billing engine for the initial reservation will be '30' seconds. For a subsequent reservation request, this parameter is ignored and the reservation lifetime cannot be extended.</td>
</tr>
<tr>
<td>Wallet Type</td>
<td>TpString</td>
<td>Optional. The wallet type to use for Charge Type 1. If not specified, the default wallet type for this account is used.</td>
</tr>
<tr>
<td>Send SMS Flag</td>
<td>TpBoolean</td>
<td>Optional. This can be set to 'TRUE' to ensure an SMS is sent when this reservation is confirmed on the billing engine (SMS only invoked if the ChargedType equals ‘1’, otherwise this parameter is ignored). Defaults to ‘FALSE’.</td>
</tr>
</tbody>
</table>
Returns
Nothing.

Charging Session Parameters

Introduction

Here are the parameters used for this functionality.

amount

Description: A TpChargingPrice structure containing the currency and a TpAmount structure. The currency will be ignored by the OSA CHAM SCS gateway. (See standard [5] for TpAmount encoding. The 'Exponent' of the TpAmount structure will always be set to '0' and the 'Number' of the TpAmount structure will always be one of: UBE small currency monies; time in seconds; or an amount for a non cash or time balance type. For a:

- CreditAmountReq, or DebitAmountReq, the 'Number' field of the TpAmount structure must be greater than zero and less than or equal to the amount:
  - already debited against this reservation (credit), or
  - stored in this reservation (debit).
- DirectCreditAmountReq, or DirectDebitAmountReq, the 'Number' field of the TpAmount structure must be greater than or equal to zero and less than the maximum cost value defined for a single amount-based request. If the amount is set to zero then the wallet and balance dates can be changed without affecting the account balance.

Note: To summarise, the amount is one of: UBE small currency monies; time in seconds; or an amount for a non cash or time balance type that the third-party client application wishes to credit/debit:

- against the reservation on the OSA CHAM SCS gateway (CreditAmountReq, or DebitAmountReq), or
- towards the subscriber’s wallet on the billing engine (DirectCreditAmountReq, or DirectDebitAmountReq)

applicationDescription

Description: A TpApplicationDescription structure. This consists of a:

- TpDateAndTime structure (See standard [2] for TpDateAndTime encoding)
- descriptive text string.

The text string is included in the CDRs (Type 21 or 22) that is generated by the OSA CHAM SCS gateway.

Otherwise the parameter is ignored by this method.

appChargingSession

Description: The reference to an IpAppChargingSession object.
ChargingEventCriteria
Description: A TpChargingEventCriteria, which contains a set of each of TpAddress structures and TpChargingEventName values. (See standard [4] for TpChargingEventName encoding.)

chargingParameters
Description: A TpChargingParameterSet containing a single TpChargingParameter structure.

closeReservation
Description: A boolean indicating whether the reservation should be closed after this request has completed. Closing the reservation may result in debiting the user's account on the billing engine: all monies debited against the current OSA CHAM SCS reservation is confirmed; and all monies, which are still available in this reservation on the OSA CHAM SCS gateway, will now become available for alternative reservations. The current reservation on the OSA CHAM SCS gateway will then become invalid for all subsequent requests from the third-party client application. The current charging session will remain active and the third-party client application may choose to make additional reservations from this account on the billing engine. Failing to close this reservation will keep the reservation active for subsequent credit or debit amount requests.

minimumAmount
Description: A TpChargingPrice structure containing the currency and a TpAmount structure. The currency is ignored by the OSA CHAM SCS gateway. (See standard [5] for TpAmount encoding. The ‘Exponent’ of the TpAmount structure will always be set to ‘0’ and the ‘Number’ of the TpAmount structure will always be the number of small units of currency where the billing engine system currency is used). The ‘Number’ field of the TpAmount structure must be greater than zero.

Note: To summarise, the minimum amount is the minimum number in small units of currency that the third-party client application wishes to reserve on the billing engine.

preferredAmount
Description: A TpChargingPrice structure containing the currency and a TpAmount structure. The currency is ignored by the OSA CHAM SCS gateway. (See standard [5] for TpAmount encoding. The ‘Exponent’ of the TpAmount structure will always be set to ‘0’ and the ‘Number’ of the TpAmount structure will always be the number of small units of currency where the billing engine system currency is used). The ‘Number’ field of the TpAmount structure must be greater than the minimum amount and less than the maximum cost value defined for a single amount-based reservation.

Note: To summarise, the preferred amount is the preferred number in small units of currency that the third-party client application wishes to reserve on the billing engine.

requestNumber
Description: The request number (unsigned integer) of the individual request within this charging session.
sessionID

Description: The session ID (unsigned integer) of this charging session.

volumes

Description: A TpVolumeSet structure containing a single TpVolume which is composed of a TpUnitID value and a TpAmount structure. The TpUnitID value must be either:
- P_CHS_UNIT_NUMBER (named events) or
- P_CHS_UNIT_SECONDS (time-based in seconds).

(See standard [5] for TpAmount encoding. The ‘Exponent’ of the TpAmount structure will always be set to ‘0’ and the ‘Number’ of the TpAmount structure will always be the number of units (named events or seconds) to be used for this reservation).

The ‘Number’ field of the TpAmount structure must be greater than zero.
Overview

Introduction

The set of methods provided by the Charging Session object is extended to allow a group of applications to use the same charging session, for example when a stateless middleware client is used between the network element and the Charging Gateway.

To provide this functionality the standard set of OSA methods are extended to allow a call-back reference to be specified with each request, to be used for response from this charging session until such time it is modified (for example by another request message providing new call-back information).

Please note that single shot messages are included in this section to allow a single charging session from the pool to be reused by many applications. Without these methods the response to the single shot message will be returned to the application reference given when creating the charging session.

The IpChargingSession object resides on the OSA CHAM SCS gateway and provides the following methods that can be invoked by the third-party client application.

In this chapter

This chapter contains the following topics.

CreditAmountWithAppChargingSessionReq() 169
CreditUnitWithAppChargingSessionReq() 170
DebitAmountWithAppChargingSessionReq() 171
DebitUnitWithAppChargingSessionReq() 172
DirectCreditAmountWithAppChargingSessionReq() 172
DirectCreditUnitWithAppChargingSessionReq() 177
DirectDebitAmountWithAppChargingSessionReq() 184
DirectDebitUnitWithAppChargingSessionReq() 187
ExtendLifeTimeWithAppChargingSessionReq() 194
GetAmountLeftWithAppChargingSession() 194
GetLifeTimeLeftWithAppChargingSession() 195
GetUnitLeftWithAppChargingSession() 195
RateWithAppChargingSessionReq() 196
ReserveAmountWithAppChargingSessionReq() 198
ReserveUnitWithAppChargingSessionReq() 201

CreditAmountWithAppChargingSessionReq()

Description

This method is invoked on the IpChargingSession to perform one of the following:
This method is invoked on the IpAccountManager to credit monies towards an existing reservation already made on the OSA CHAM SCS gateway. The incoming request will specify the number of UBE small currency monies to be credited.

This method is invoked on the IpAccountManager to credit time towards an existing reservation already made on the OSA CHAM SCS gateway. The incoming request will specify the number of seconds to be credited, the internal UBE time balance units is 100ths seconds so a conversion will be required.

This method is invoked on the IpAccountManager to credit non cash or time towards an existing reservation already made on the OSA CHAM SCS gateway. The incoming request will specify the amount to be credited; the internal UBE units are mapped directly to the specified amount.

If a reservation does not exist for this user, or the reservation has expired then this request will fail. It is not possible to credit an amount towards a reservation unless debits greater to or equal to the credited amount have already been debited – it is not possible to credit a subscriber’s wallet through the reservation-confirmation process.

This method also set the Application Charging Session reference - all call-backs from the specified session will be sent to the new reference.

**Parameters**

Here is a list of the parameters for this method.

- `sessionID` (on page 168)
- `appChargingSession` (on page 166)
- `applicationDescription` (on page 166)
- `amount` (on page 166)
- `closeReservation` (on page 167)
- `requestNumber` (on page 167)

**Note:** The `requestNumber` parameter may be given the value of zero (0) and then the CHAM will use the currently stored value for the request number.

**Returns**

Nothing.

**CreditUnitWithAppChargingSessionReq()**

**Description**

This method is invoked on the IpChargingSession to credit units towards an existing reservation already made on the OSA CHAM SCS gateway. If a reservation does not exist for this user, or the reservation has expired, then this request will fail. It is not possible to credit units towards a reservation unless debits greater to or equal to the credited units have already been debited – it is not possible to credit a subscriber’s wallet through the reservation-confirmation process.

This method will also set the Application Charging Session reference – all call-backs from the specified session will be sent to the new reference.

**Parameters**

Here is a list of the parameters for this method.

- `sessionID` (on page 168)
- `appChargingSession` (on page 166)
- `applicationDescription` (on page 166)
volumes (on page 168)
- closeReservation (on page 167)
- requestNumber (on page 167)

Note: The requestNumber parameter may be given the value of zero (0) and then the CHAM will use the currently stored value for the request number.

Returns
Nothing.

DebitAmountWithAppChargingSessionReq()

Description
This method will be invoked on the IpChargingSession to perform one of the following:

- This method will be invoked on the IpAccountManager to debit monies towards an existing reservation already made on the OSA CHAM SCS gateway. The incoming request will specify the number of UBE small currency monies to be debited.
- This method will be invoked on the IpAccountManager to debit time towards an existing reservation already made on the OSA CHAM SCS gateway. The incoming request will specify the number of seconds to be debited, the internal UBE time balance units is 100ths seconds so a conversion will be required.
- This method will be invoked on the IpAccountManager to debit non cash or time towards an existing reservation already made on the OSA CHAM SCS gateway. The incoming request will specify the amount to be debited; the internal UBE units are mapped directly to the specified amount.
- If a reservation does not exist for this user, or the reservation has expired then this request will fail.

This method also sets the Application Charging Session reference – all call-backs from the specified session will be sent to the new reference.

Parameters
Here is a list of the parameters for this method.

- sessionID (on page 168)
- appChargingSession (on page 166)
- applicationDescription (on page 166)
- amount (on page 166)
- closeReservation (on page 167)
- requestNumber (on page 167)

Note: The requestNumber parameter may be given the value of zero (0) and then the CHAM will use the currently stored value for the request number.

Returns
Nothing.
**DebitUnitWithAppChargingSessionReq()**

**Description**

This method will be invoked on the IpChargingSession to debit units from an existing reservation already made on the OSA CHAM SCS gateway. If a reservation does not exist for this user, or the reservation has expired then this request will fail.

This method also sets the Application Charging Session reference - all call-backs from the specified session will be sent to the new reference.

**Parameters**

Here is a list of the parameters for this method.

- `sessionID` (on page 168)
- `appChargingSession` (on page 166)
- `applicationDescription` (on page 166)
- `volumes` (on page 168)
- `closeReservation` (on page 167)
- `requestNumber` (on page 167)

**Note:** The `requestNumber` parameter may be given the value of zero (0) and then the CHAM will use the currently stored value for the request number.

**Returns**

Nothing.

**DirectCreditAmountWithAppChargingSessionReq()**

**Description**

This method is invoked on the IpChargingSession to perform one of the following:

- This method is invoked on the IpChargingSession to credit monies towards a subscriber's wallet using cash balance types only. The specified (or default) balance type or the specified balance type cascade (the UBE will credit the first balance type in the balance type cascade) is used. The UBE will create the specified balance type or first balance type in the balance type cascade if the balance type does not already exist in the subscriber's wallet. If the balance type cascade name is specified then the balance type name is ignored. The incoming request will specify the number of UBE small currency monies to be credited. Typically this method is used for balance types where the unit type of the balance type is 'Cash'; if the balance type cascade name is specified in the incoming request then only balance types where the unit type of the balance type is 'Cash' is available in the balance type cascade.

- This method is invoked on the IpChargingSession to credit time towards a subscriber's wallet using time balance types. The incoming request will specify the number of seconds to be credited, the internal UBE time balance units is 100ths seconds so a conversion is required. The balance type name must be specified in the incoming request where the unit type of the balance type is 'Time' and the `allowNonCashBalanceForAmountMethods` (on page 18) configuration parameter must be 'true' for time-based reservations to be supported. If the balance type cascade name is specified in the incoming request then the amount specified is assumed to be monies and only balance types where the unit type of the balance type is 'Cash' is available in the balance type cascade.
This method is invoked on the IpChargingSession to credit an amount towards a subscriber's wallet using non-cash or time balance types. The incoming request will specify the amount to be credited; the internal UBE units are mapped directly to the specified amount. The balance type name must be specified in the incoming request where the unit type of the balance type is non 'Cash' or 'Time' and the allowNonCashBalanceForAmountMethods configuration parameter must be 'true' for non cash or time-based reservations to be supported. If the balance type cascade name is specified in the incoming request then the amount specified is assumed to be monies and only balance types where the unit type of the balance type is 'Cash' is available in the balance type cascade.

Extend the expiry date of the specified (or default) balance type by a specified number of days based on the rules detailed in *Extending balance and wallet expiry dates* (on page 131). The number of days is taken from the incoming P_CHS_BALANCE_VALIDITY_MODIFICATION parameter or if not specified and the PI_XDATE_EXTENSION table is installed then determine the appropriate balance extension from this table. If the number of days is not specified in the incoming P_CHS_BALANCE_VALIDITY_MODIFICATION parameter and the PI_XDATE_EXTENSION table is not installed then no default value is used. Specifying the balance type cascade is not supported in conjunction with specifying the balance expiry extension parameter. A configuration item (defaulting to 'false') will cause the request to fail if the balance extension parameter exceeds the wallet extension parameter or if the result of extending the balance expiry date exceeds the current or extended wallet expiry date; this expiry indicator is passed through to the UBE.

Extend the expiry date of the specified (or default) wallet type by a specified number of days based on the rules detailed in *Extending balance and wallet expiry dates*. The number of days is taken from the incoming P_CHS_WALLET_VALIDITY_MODIFICATION parameter or if not specified and the PI_YDATE_EXTENSION table is installed then determine the appropriate wallet extension from this table. If the number of days is not specified in the incoming P_CHS_WALLET_VALIDITY_MODIFICATION parameter and the PI_YDATE_EXTENSION table is not installed then no default value is used. A configuration item (defaulting to 'false') will cause the request to fail if the wallet extension parameter is less than the balance extension parameter or if the result of extending the wallet expiry date is less than the extended balance expiry date; this expiry indicator is passed through to the UBE. If the balance expiry date is not being extended then this check cannot be performed.

Set the value of the specified (or default) balance type to a specific value where the UBE will create the specified balance type if the balance type does not already exist in the subscriber's wallet. The incoming P_CHS_OVERRIDE_BALANCE_VALUE parameter will indicate the value supplied is an override and not to be used to credit an existing balance type; this override indicator is passed through to the UBE.

Set the expiry date of the specified (or default) balance type to a specific date (value taken from the incoming P_CHS_BALANCE_VALIDITY_DATE parameter). A configuration item (defaulting to 'false') will cause the request to fail if the balance specific date parameter exceeds the wallet specific date parameter; this expiry indicator is passed through to the UBE.

Set the expiry date of the specified (or default) wallet type to a specific date (value taken from the incoming P_CHS_WALLET_VALIDITY_DATE parameter). A configuration item (defaulting to 'false') will cause the request to fail if the wallet specific date parameter is less than the balance specific date parameter; this expiry indicator is passed through to the UBE. If the balance expiry date is not specified then this check cannot be performed.

Recharge a wallet using a voucher (voucher number and voucher pin taken from the incoming P_CHS_VOUCHER parameter). If the EDR details contain both the BONUS_TYPE and REFERENCE tags then an applicable bonus value may be incorporated into the amount credited to subscriber's wallet subject to the rules detailed in *Applying bonuses* (on page 133).

Recharge a wallet using a voucher type (voucher type name taken from the incoming P_CHS_VOUCHER_TYPE_NAME parameter). If the EDR details contain both the BONUS_TYPE and REFERENCE tags then an applicable bonus value may be incorporated into the amount credited to subscriber's wallet subject to the rules detailed in *Applying bonuses.*
Bonuses may be applied to the value specified in the request; these will be calculated on the UBE using the P_CHS_BONUS_TYPE and P_CHS_REFERENCE parameters when crediting a subscriber's wallet using a specified value, recharging a subscriber using a voucher or recharging a subscriber using a voucher type. The bonus type and reference parameters will be passed through to the UBE.

If the specified balance type, or default balance type if none is specified, does not exist, then the balance type will be created; the expiry date for the existing or new balance type and for the wallet will be set based on the rules detailed in Extending balance and wallet expiry dates (on page 131) or Specifying balance and wallet expiry dates (on page 132) depending on which scenario applies.

If the balance type cascade is specified, then the first balance type in the cascade will be credited and if this balance type does not exist then the balance type will be created; the expiry date for the existing or new balance type and the wallet will be set based on the rules detailed in Extending balance and wallet expiry dates; specifying a specific balance date is not supported in conjunction with specifying a balance type cascade.

If the 'override balance wallet value' flag is specified and the balance type:

- already exists, then the specified balance type will have a single bucket value updated to this value (the bucket with the latest expiry date) and all remaining buckets in the balance type will be deleted; the expiry date for the balance type and wallet will be set based on the rules detailed in Extending balance and wallet expiry dates or Specifying balance and wallet expiry dates depending on which scenario applies. Specifying the balance type cascade is not supported in conjunction with specifying the 'override balance wallet value' flag. A configuration item (defaulting to 'false') will cause the request to fail if the specific balance date parameter exceeds the specific wallet date parameter; this expiry indicator will be passed through to the UBE.

- does not exist then the balance type will be created; the expiry date for the balance type and wallet will be set based on the rules detailed in Extending balance and wallet expiry dates or Specifying balance and wallet expiry dates depending on which scenario applies.

If the EDR details contain both the BONUS_TYPE and REFERENCE tags, then an applicable bonus value may be incorporated into the amount credited to subscriber's wallet subject to the rules detailed in Applying bonuses (on page 133).

The current reservation, if one exists, will not be affected through invoking this request.

This method also sets the Application Charging Session reference - all call-backs from the specified session will be sent to the new reference.

Notes:

- It is not possible to combine balance and wallet extensions with balance and wallet specific dates in the incoming request; combining these charging parameters will cause the request to fail.

- The subscriber's wallet may not be available on the local SCS database if the user is not a prepaid subscriber. For post-paid subscribers the user can be 'cold-billed' using the information extracted from the OSA CDR. If the MSISDN is not found when the chargedParty is decoded and the 'AllowPostPaid' flag is set to true then the request will return success. Otherwise, if the chargedParty fails to decode successfully and the 'AllowPostPaid' flag is set to false, then the request will return failure.

Parameters

Here is a list of the parameters for this method.

- **sessionID** (on page 168)
- **appChargingSession** (on page 166)
- **applicationDescription** (on page 166)
- **chargingParameters** (on page 167)
- **amount** (on page 166)
• requestNumber (on page 167)

**Note:** The requestNumber parameter may be given the value of zero (0) and then the CHAM will use the currently stored value for the request number.

### TpChargingParameter structure

The treatment and expected type of each element within each TpChargingParameter structure is:

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| ChargedParty               | TpString     | Name: P_CHS_CHARGE_PARTY  
Mandatory.  
ChargedType 1: The TpString will correspond to the MSISDN to charge.  
ChargedType 3: The TpString will correspond to the wallet ID and billing engine ID to charge. The wallet ID and billing engine ID are separated by '/'. |
| ChargedType                | TpInt32      | Name: P_CHS_CHARGE_TYPE  
Mandatory. Value showing type of ChargedParty.  
• 1 - MSISDN  
• 2 - reserved  
• 3 - Wallet ID |
| Discount Override          | TpInt32      | Name: P_CHS_DISCOUNT_OVERRIDE  
Percentage discount to apply to final cost. Valid values are 0 – 10000 where these are in 1/100th of a percent.  
Therefore, 500 would equate to a 5% discount.  
The discount applied to the final cost will depend solely on the OSA discount entered here. |
| Wallet Type Name           | TpString     | Name: P_CHS_WALLET_TYPE  
Optional. The wallet type to use for Charge Type 1. If not specified, the default wallet type for this account is used. |
| Balance Type Name          | TpString     | Name: P_CHS_BALANCE_TYPE  
Optional. The balance type to use for Charge Type 1. If not specified, the default balance type for this account is used. |
| Balance Type Cascade Name  | TpString     | Name: P_CHS_BALANCE_TYPE_CASCADE  
Optional. The balance type cascade name to use for Charge Type 1. If:  
• specified then the balance type name will be ignored.  
• not specified, the default balance type for this account will be used. |
| Voucher Number and Voucher PIN | TpString | Name: P_CHS_VOUCHER  
Optional. The voucher number and voucher pin used when doing a voucher recharge. |
| Voucher Type Name          | TpString     | Name: P_CHS_VOUCHER_TYPE_NAME  
Optional. The voucher type name used when doing a voucher type recharge. |
<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bonus Type Name</td>
<td>TpString</td>
<td><strong>Name: P_CHS_BONUS_TYPE</strong>&lt;br&gt;Optional. The bonus type used in conjunction with the bonus reference to determine the applicable bonus to be applied to the value when crediting the subscriber’s wallet. A match will be done against the configured bonus in the database using the CCS_BONUS_TYPE.type column.&lt;br&gt;Valid value is ‘C(U)STOM’ where ‘U’ will be stored in the database.&lt;br&gt;Bonuses will be applied subject to the rules detailed in <em>Applying bonuses</em> (on page 133).</td>
</tr>
<tr>
<td>Reference</td>
<td>TpString</td>
<td><strong>Name: P_CHS_REFERENCE</strong>&lt;br&gt;Optional. The bonus reference used in conjunction with the bonus type to determine the applicable bonus to be applied to the value when crediting the subscriber’s wallet. A match will be done against the configured bonus in the database using the CCS_BONUS_TYPE.component column.&lt;br&gt;This is a freeform text string; maximum of 20 characters.&lt;br&gt;Bonuses will be applied subject to the rules detailed in <em>Applying bonuses</em> (on page 133).</td>
</tr>
</tbody>
</table>
| Override Balance Wallet Value and Dates Flag | TpBoolean | **Name: P_CHS_OVERRIDE_BALANCE_VALUE**<br>Optional. This can be set to 'TRUE' to override the value in the specified or default balance type with this amount; it cannot be used in conjunction with a balance type cascade name.<br>This can be used in conjunction with the balance and wallet validity date parameters to specify specific expiry dates and must be present to use either of the balance or wallet validity date parameters.<br>Defaults to ‘FALSE’.
| Balance Validity Modification      | TpInt32      | **Name:** <br>Optional. The amount of days when extending the balance expiry date. The new balance balance expiry date will be set to the current date and time plus the number of days entered here. Only those expiry dates within the determined balance type (entered balance type or default balance type if none entered), which would be increased by this modification, will be affected. Any expiry date within the determined balance type that is set to 'never expire' will not be affected by this request.<br>The new balance expiry date will be set to this value irrespective of the current balance expiry date subject to the rules detailed in *Extending balance and wallet expiry dates* (on page 131). |
### Element Name | Element Type | Description
--- | --- | ---
Wallet Validity Modification | TpInt32 | Name: P_CHS_WALLET_VALIDITY_MODIFICATION
Optional. The amount of days when extending the wallet expiry date. The new wallet expiry date will be set to the current date and time plus the number of days entered here. The wallet expiry cannot be decreased by this modification. If the expiry date within the determined wallet type is set to ‘never expire’ it will not be affected by this request.

The new wallet expiry date will be set to this value irrespective of the current balance expiry date subject to the rules detailed in *Extending balance and wallet expiry dates* (on page 131).

Balance Validity Date | TpDate | Name: P_CHS_BALANCE_VALIDITY_DATE
Optional. The specific date when setting the balance expiry date. The new balance expiry date will be set to this value irrespective of the current balance expiry date subject to the rules detailed in *Specifying balance and wallet expiry dates* (on page 132).

The override balance wallet value parameter must be present for this parameter to be taken into account.

Wallet Validity Date | TpDate | Name: P_CHS_WALLET_VALIDITY_DATE
Optional. The specific date when setting the wallet expiry date. The new wallet expiry date will be set to this value irrespective of the current wallet expiry date subject to the rules detailed in *Specifying balance and wallet expiry dates* (on page 132).

The override balance wallet value parameter must be present for this parameter to be taken into account.

Send SMS Flag | TpBoolean | Name: P_CHS_SEND_SMS_FLAG
Optional. This can be set to ‘TRUE’ to ensure an SMS is sent when this reservation is confirmed on the billing engine (SMS only invoked if the ChargedType equals ‘1’, otherwise this parameter is ignored). Defaults to ‘FALSE’.

**Returns**

Nothing.

**DirectCreditUnitWithAppChargingSessionReq()**

**Description**

This method is invoked on the IpChargingSession to perform one of the following:
Credit units towards a subscriber's wallet using the specified balance type or the specified balance type cascade (the UBE will credit the first balance type in the balance type cascade) where the UBE will create the specified balance type or first balance type in the balance type cascade if the balance type does not already exist in the subscriber's wallet. If the balance type cascade name is specified, then the balance type name will be ignored. If neither is specified then the default balance type cascade already defined for either the rating definition or the billable-event definition will be used. Each unit has an associated cost on the billing engine and therefore a specified number of units translate into a fixed amount of monies to be credited towards the subscriber's wallet. The only supported unit-based reservations are seconds and billable events.

Extend the expiry date of the specified balance type by a specified number of days based on the rules detailed in Extending balance and wallet expiry dates (on page 131), albeit the 'override balance wallet value' flag is not supported for this request. The number of days will be taken from the incoming P_CHS_BALANCE_VALIDITY_MODIFICATION parameter. Specifying the balance type cascade is not supported in conjunction with specifying the balance expiry extension parameter. A configuration item (defaulting to 'false') will cause the request to fail if the balance extension parameter exceeds the wallet extension parameter, or if the result of extending the balance expiry date exceeds the current or extended wallet expiry date; this expiry indicator will be passed through to the UBE.

Extend the expiry date of the specified (or default) wallet type by a specified number of days based on the rules detailed in Extending balance and wallet expiry dates, albeit the 'override balance wallet value' flag is not supported for this request. The number of days will be taken from the incoming P_CHS_WALLET_VALIDITY_MODIFICATION parameter. A configuration item (defaulting to 'false') will cause the request to fail if the wallet extension parameter is less than the balance extension parameter; this expiry indicator will be passed through to the UBE.

If the specified balance type does not exist then the balance type will be created; the expiry date for the existing or new balance type and for the wallet will be set based on the rules detailed in Extending balance and wallet expiry dates, albeit the 'override balance wallet value' flag is not supported for this request.

If the balance type cascade is specified then the first balance type in the cascade will be credited and if this balance type does not exist then the balance type will be created; the expiry date for the existing or new balance type and the wallet will be set based on the rules detailed in Extending balance and wallet expiry dates, albeit the 'override balance wallet value' flag is not supported for this request; specifying a specific balance date is not supported in conjunction with specifying a balance type cascade.

The current reservation, if one exists, will not be affected through invoking this request.

Note: The subscriber's wallet may not be available on the local SCS database if the user is not a prepaid subscriber. For post-paid subscribers the user can be 'cold-billed' using the information extracted from the OSA CDR. If the MSISDN is not found when the chargeParty is decoded and the 'AllowPostPaid' flag is set to true and the rating tariff for this subscriber can be determined then the request will return success. Otherwise, if the chargeParty fails to decode successfully and the 'AllowPostPaid' flag is set to false and/or the rating tariff for this subscriber can not be determined, then the request will return failure. This method will also set the Application Charging Session reference - all call-backs from the specified session will be sent to the new reference.

Parameters

Here is a list of the parameters for this method.

- **sessionID** (on page 168)
- **appChargingSession** (on page 166)
- **applicationDescription** (on page 166)
- **chargingParameters** (on page 167)
- **volumes** (on page 168)
- **requestNumber** (on page 167)
Note: The `requestNumber` parameter may be given the value of zero (0) and then the CHAM will use the currently stored value for the request number.

### Unit-Seconds based credit request

The treatment and expected type of each element within each TpChargingParameter structure for a unit-seconds based direct delta is:

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| ChargedParty    | TpString     | Name: P_CHS_CHARGE_PARTY  
Mandatory.  
ChargedType 1: The TpString will correspond to the MSISDN to charge.  
ChargedType 3: The TpString will correspond to the wallet ID and billing engine ID to charge.  
The wallet ID and billing engine ID is separated by '/'. |
| ChargedType     | TpInt32      | Name: P_CHS_CHARGE_TYPE  
Mandatory. Value showing type of ChargedParty.  
- 1 - MSISDN  
- 2 - reserved  
- 3 - Wallet ID |
| Calling Number  | TpString     | Name: P_CHS_CALLING_NUMBER  
Mandatory. Network Calling Number to be used for rating-tariff calculations. |
| Called Number   | TpString     | Name: P_CHS_CALLED_NUMBER  
Mandatory. Network Called Number to be used for rating-tariff calculations. |
| Discount Override| TpInt32    | Name: P_CHS_DISCOUNT_OVERRIDE  
Percentage discount to apply to final cost.  
Valid values are 0 – 10000 where these are in 1/100th of a percent.  
Therefore, 500 would equate to a 5% discount.  
The discount applied to the final cost will depend on the 'discountRuleType' parameter set in the CCS.ReservationHandler section of eserv.config file on the UBE. The following are valid values where the resulting discounted cost will be applied: |
<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>discountRuleType = 'ServiceOverride' (use only OSA discount)</td>
<td></td>
<td>finalCost = initialCost * (OSA discount)</td>
</tr>
<tr>
<td>discountRuleType = 'S*R+W'</td>
<td></td>
<td>finalCost = initialCost * [OSA discount * CCS rating discount * CCS wallet discount]</td>
</tr>
<tr>
<td>discountRuleType = 'S+R+W'</td>
<td></td>
<td>finalCost = initialCost * [OSA discount + CCS rating discount + CCS wallet discount]</td>
</tr>
<tr>
<td>discountRuleType = 'S+R+*W'</td>
<td></td>
<td>finalCost = initialCost * [OSA discount + CCS wallet discount * CCS rating discount]</td>
</tr>
<tr>
<td>discountRuleType = 'S+W*R'</td>
<td></td>
<td>finalCost = initialCost * [OSA discount * CCS wallet discount + CCS rating discount]</td>
</tr>
<tr>
<td>discountRuleType = 'R+W*S'</td>
<td></td>
<td>finalCost = initialCost * [CCS rating discount + CCS wallet discount + OSA discount]</td>
</tr>
<tr>
<td>discountRuleType = 'R+W+S'</td>
<td></td>
<td>finalCost = initialCost * [CCS rating discount * CCS wallet discount + OSA discount]</td>
</tr>
</tbody>
</table>

### Wallet Type Name

**Name:** P_CHS_WALLET_TYPE

Optional. The wallet type name to use for Charge Type 1. If not specified, the default wallet type for this account is used.
<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Balance Type Name       | TpString     | **Name:** P\_CHS\_BALANCE\_TYPE  
Optional. The balance type name to use for Charge Type 1. If:  
  - not specified, there will be no default balance type for this operation.  
  - specified then a balance type cascade containing only this balance type will override the balance type cascade already defined for the rating definition. |
| Balance Type Cascade Name | TpString     | **Name:** P\_CHS\_BALANCE\_TYPE\_CASCADE  
Optional. The balance type cascade name to use for Charge Type 1. If specified, this balance type cascade will override the balance type cascade already defined for the rating definition. |
| Balance Validity         | TpInt32      | **Name:** P\_CHS\_BALANCE\_VALIDITY\_MODIFICATION  
Optional. The number of days when extending the balance expiry date. The new balance expiry date will be set to the current date and time plus the number of days entered here. Only those expiry dates within the determined balance type (entered balance type), which would be increased by this modification, will be affected. Any expiry date within the determined balance type that is set to 'never expire' will not be affected by this request. The balance type parameter must be present for this parameter to be taken into account. |
| Wallet Validity          | TpInt32      | **Name:** P\_CHS\_WALLET\_VALIDITY\_MODIFICATION  
Optional. The number of days when extending the wallet expiry date. The new wallet expiry date will be set to the current date and time plus the number of days entered here. The wallet expiry cannot be decreased by this modification. If the expiry date within the determined wallet type is set to 'never expire', it will not be affected by this request. The balance type parameter must be present for this parameter to be taken into account. |
Send SMS Flag  | TpBoolean   | Name: P_CHS_SEND_SMS_FLAG
Optional. This can be set to ‘TRUE’ to ensure an SMS is sent when this reservation is confirmed on the billing engine (SMS only invoked if the ChargedType equals ‘1’, otherwise this parameter is ignored). Defaults to ‘FALSE’.

### Unit-Billable event based credit request

The treatment and expected type of each element within each TpChargingParameter structure for a unit-seconds based direct delta is:

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| ChargedParty       | TpString     | Name: P_CHS_CHARGE_PARTY
Mandatory. ChargedType 1: The TpString will correspond to the MSISDN to charge. ChargedType 3: The TpString will correspond to the wallet ID and billing engine ID to charge. The wallet ID and billing engine ID is separated by ‘/’. |
| ChargedType        | TpInt32      | Name: P_CHS_CHARGE_TYPE
Mandatory. Value showing type of ChargedParty.
- 1 - MSISDN
- 2 - reserved
- 3 - Wallet ID |
| Discount Override  | TpInt32      | Name: P_CHS_DISCOUNT_OVERRIDE
Percentage discount to apply to final cost. Valid values are 0 – 10000 where these are in 1/100th of a percent. Therefore, 500 would equate to a 5% discount. The discount applied to the final cost will depend on the OSA discount entered here.
Note: CCS discounts do not apply to billable events. |
| Event Name         | TpString     | Name: P_CHS_EVENT_NAME
Mandatory. Event Class and Name for rating tariff calculation. The TpString will correspond to the event class and event name to charge. The event class and event name are separated by ‘/’. |
| Wallet Type Name   | TpString     | Name: P_CHS_WALLET_TYPE
Optional. The wallet type name to use for Charge Type 1. If not specified, the default wallet type for this account is used. |
<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Balance Type Name</td>
<td>TpString</td>
<td>Optional. The balance type name to use for Charge Type 1. If:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• not specified, there will be no default balance type for this operation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• specified then a balance type cascade containing only this balance type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>will override the balance type cascade already defined for the rating</td>
</tr>
<tr>
<td></td>
<td></td>
<td>definition.</td>
</tr>
<tr>
<td>Balance Type Cascade Name</td>
<td>TpString</td>
<td>Optional. The balance type cascade name to use for Charge Type 1. If specified, then this balance type cascade will override the balance type cascade already defined for the billable-event definition.</td>
</tr>
<tr>
<td>Balance Validity Modification</td>
<td>TpInt32</td>
<td>Optional. The number of days when extending the balance expiry date. The new</td>
</tr>
<tr>
<td></td>
<td></td>
<td>balance expiry date will be set to the current date and time plus the number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>of days entered here. Only those expiry dates within the determined balance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>type (entered balance type), which would be increased by this modification,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>will be affected. Any expiry date within the determined balance type that is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>set to ‘never expire’ will not be affected by this request. The balance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>type parameter must be present for this parameter to be taken into account.</td>
</tr>
<tr>
<td>Wallet Validity Modification</td>
<td>TpInt32</td>
<td>Optional. The number of days when extending the wallet expiry date. The new</td>
</tr>
<tr>
<td></td>
<td></td>
<td>wallet expiry date will be set to the current date and time plus the number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>of days entered here. The wallet expiry cannot be decreased by this</td>
</tr>
<tr>
<td></td>
<td></td>
<td>modification. If the expiry date within the determined wallet type is set to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘never expire’, it will not be affected by this request. The balance type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>parameter must be present for this parameter to be taken into account.</td>
</tr>
<tr>
<td>Element Name</td>
<td>Element Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| Send SMS Flag     | TpBoolean     | Name: P_CHS_SEND_SMS_FLAG  
Optional. This can be set to ‘TRUE’ to ensure an SMS is sent when this reservation is confirmed on the billing engine (SMS only invoked if the ChargedType equals ‘1’, otherwise this parameter is ignored). Defaults to ‘FALSE’. |

**Returns**

Nothing.

**DirectDebitAmountWithAppChargingSessionReq()**

**Description**

This method is invoked on the IpChargingSession to perform one of the following:

- This method is invoked on the IpChargingSession to debit monies from a subscriber's wallet using cash balance types only. The specified (or default) balance type or the specified balance type cascade is used. If the balance type cascade name is specified, then the balance type name will be ignored. If there are insufficient funds in the specified balance type or across all balance types in the balance type cascade, then based on a parameter in the incoming request, the balance type, or in the case of the balance type cascade, the final balance type in the balance type cascade will be allowed to be taken negative; this 'allow negative balance' indicator will be passed through to the UBE. The incoming request specifies the number of UBE small currency monies to be credited. Typically this method is used for balance types where the unit type of the balance type is 'Cash'; if the balance type cascade name is specified in the incoming request, then only balance types where the unit type of the balance type is 'Cash' will be available in the balance type cascade.

- This method is invoked on the IpChargingSession to debit time towards a subscriber's wallet using time balance types. The incoming request specifies the number of seconds to be debited, the internal UBE time balance units is 100ths seconds so a conversion will be required. The balance type name must be specified in the incoming request where the unit type of the balance type is 'Time' and the allowNonCashBalanceForAmountMethods (on page 18) configuration parameter must be 'true' for time-based reservations to be supported. If the balance type cascade name is specified in the incoming request then the amount specified will be assumed to be monies and only balance types where the unit type of the balance type is 'Cash' will be available in the balance type cascade.

- This method is invoked on the IpChargingSession to debit an amount towards a subscriber's wallet using non-cash or time balance types. The incoming request specifies the amount to be debited; the internal UBE units are mapped directly to the specified amount. The balance type name must be specified in the incoming request where the unit type of the balance type is non 'Cash' or 'Time' and the allowNonCashBalanceForAmountMethods configuration parameter must be 'true' for non cash or time-based reservations to be supported. If the balance type cascade name is specified in the incoming request then the amount specified will be assumed to be monies and only balance types where the unit type of the balance type is 'Cash' will be available in the balance type cascade.
Extend the expiry date of the specified (or default) balance type by a specified number of days based on the rules detailed in *Extending balance and wallet expiry dates* (on page 131), albeit the 'override balance wallet value' flag is not supported for this request. The number of days are taken from the incoming P_CHS_BALANCE_VALIDITY_MODIFICATION parameter. Specifying the balance type cascade is not supported in conjunction with specifying the balance expiry extension parameter. A configuration item (defaulting to 'false') will cause the request to fail if the balance extension parameter exceeds the wallet extension parameter or if the result of extending the balance expiry date exceeds the current or extended wallet expiry date; this expiry indicator will be passed through to the UBE.

Extend the expiry date of the specified (or default) wallet type by a specified number of days based on the rules detailed in *Extending balance and wallet expiry dates*, albeit the 'override balance wallet value' flag is not supported for this request. The number of days are taken from the incoming P_CHS_WALLET_VALIDITY_MODIFICATION parameter. A configuration item (defaulting to 'false') will cause the request to fail if the wallet extension parameter is less than the balance extension parameter or if the result of extending the wallet expiry date is less than the extended balance expiry date; this expiry indicator will be passed through to the UBE. If the balance expiry date is not being extended, then this check cannot be performed.

The specified balance type, or default balance type, if none is specified, will be debited and the expiry date for the balance type and for the wallet will be set based on the rules detailed in *Extending balance and wallet expiry dates*, albeit the 'override balance wallet value' flag is not supported for this request.

If the balance type cascade is specified, then the balances will be debited in order of the cascade; specifying a balance expiry extension is not supported in conjunction with specifying a balance type cascade.

The current reservation, if one exists, will not be affected through invoking this request.

**Note:** The subscriber's wallet may not be available on the local SCS database if the user is not a prepaid subscriber. For post-paid subscribers the user can be 'cold-billed' using the information extracted from the OSA CDR. If the MSISDN is not found when the chargedParty is decoded and the 'AllowDirectPostPaid' flag is set to 'true' then the request will return success. Otherwise, if the chargedParty fails to decode successfully and the 'AllowDirectPostPaid' flag is set to 'false', then the request will return failure.

This method also sets the Application Charging Session reference - all call-backs from the specified session will be sent to the new reference.

**Parameters**

Here is a list of the parameters for this method.

- sessionID (on page 168)
- appChargingSession (on page 166)
- applicationDescription (on page 166)
- chargingParameters (on page 167)
- amount (on page 166)
- requestNumber (on page 167)

**Note:** The requestNumber parameter may be given the value of zero (0) and then the CHAM will use the currently stored value for the request number.
## TpChargingParameter structure

The treatment and expected type of each element within each TpChargingParameter structure is:

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChargedParty</td>
<td>TpString</td>
<td>Name: P_CHS_CHARGE_PARTY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mandatory. ChargedType 1: The TpString will correspond to the MSISDN to charge.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ChargedType 3: The TpString will correspond to the wallet ID and billing engine ID to charge. The wallet ID and billing engine ID are separated by ‘/’.</td>
</tr>
<tr>
<td>ChargedType</td>
<td>TpInt32</td>
<td>Name: P_CHS_CHARGE_TYPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mandatory. Value showing type of ChargedParty.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 - MSISDN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 2 - reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 3 - Wallet ID</td>
</tr>
<tr>
<td>Discount Override</td>
<td>TpInt32</td>
<td>Name: P_CHS_DISCOUNT_OVERRIDE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Percentage discount to apply to final cost. Valid values are 0 – 10000 where these are in 1/100th of a percent. Therefore, 500 would equate to a 5% discount. The discount applied to the final cost will depend solely on the OSA discount entered here.</td>
</tr>
<tr>
<td>Wallet Type Name</td>
<td>TpString</td>
<td>Name: P_CHS_WALLET_TYPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional. The wallet type name to use for Charge Type 1. If not specified, the default wallet type for this account is used.</td>
</tr>
<tr>
<td>Balance Type Name</td>
<td>TpString</td>
<td>Name: P_CHS_BALANCE_TYPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional. The balance type name or balance type cascade name to use for Charge Type 1. If not specified, the default balance type for this account will be used. To specify a balance cascade, a ‘;’ is used to identify the parameter as a cascade. e.g. &quot;;promo_cascade&quot;. When a balance type cascade name is specified only balances containing 'Cash' Balance Units will be included in the result. The additional ‘;’ is maintained when specifying the balance type cascade name for backwards compatibility only, if the balance type cascade parameter is specified separately then this parameter will take priority.</td>
</tr>
<tr>
<td>Balance Type Cascade Name</td>
<td>TpString</td>
<td>Name: P_CHS_BALANCE_TYPE_CASCADE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional. The balance type cascade name to use for Charge Type 1. If:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• specified then the balance type name will be ignored.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• not specified, the default balance type for this account will be used.</td>
</tr>
<tr>
<td>Element Name</td>
<td>Element Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Balance Validity Modification| TpInt32      | Name: P_CHS_BALANCE_VALIDITY_MODIFICATION  
Optional. The number of days when extending the balance expiry date. The new balance balance expiry date will be set to the current date and time plus the number of days entered here. Only those expiry dates within the determined balance type (entered balance type or default balance type if none entered), which would be increased by this modification, will be affected. Any expiry date within the determined balance type that is set to 'never expire' will not be affected by this request. |
| Wallet Validity Modification | TpInt32      | Name: P_CHS_WALLET_VALIDITY_MODIFICATION  
Optional. The number of days when extending the wallet expiry date. The new wallet expiry date will be set to the current date and time plus the number of days entered here. The wallet expiry cannot be decreased by this modification. If the expiry date within the determined wallet type is set to 'never expire', it will not be affected by this request. |
| Send SMS Flag                | TpBoolean    | Name: P_CHS_SEND_SMS_FLAG  
Optional. This can be set to 'TRUE' to ensure an SMS is sent when this reservation is confirmed on the billing engine (SMS only invoked if the ChargedType equals '1', otherwise this parameter is ignored). Defaults to 'FALSE'. |
| Allow Negative Balance Flag  | TpBoolean    | Name: P_CHS_ALLOW_NEGATIVE_BALANCE  
Optional. This can be set to 'TRUE' to set the 'IgnoreWalletBalance' flag in the UBE request message. Defaults to 'FALSE'. |

**Returns**

Nothing.

**DirectDebitUnitWithAppChargingSessionReq()**

**Description**

This method is invoked on the IpChargingSession to perform one of the following:
• Debit units from a subscriber's wallet using the specified balance type or the specified balance type cascade. If the balance type cascade name is specified, then the balance type name will be ignored. If there are insufficient funds in the specified balance type or across all balance types in the balance type cascade, then based on a parameter in the incoming request, the balance type, or in the case of the balance type cascade, then the final balance type in the balance type cascade will be allowed to be taken negative; this 'allow negative balance' indicator will be passed through to the UBE. If the 'allow negative balance' indicator is not set, for a seconds based charge if there is insufficient funds then the maximum number of seconds will be returned. Each unit has an associated cost on the billing engine and therefore a specified number of units translate into a fixed amount of monies to be debited from the subscriber's wallet. The only supported unit-based reservations are seconds and billable events.

• Extend the expiry date of the specified balance type by a specified number of days based on the rules detailed in Extending balance and wallet expiry dates (on page 131), albeit the ‘override balance wallet value’ flag is not supported for this request. The number of days will be taken from the incoming P_CHS_BALANCE_VALIDITY_MODIFICATION parameter. Specifying the balance type cascade is not supported in conjunction with specifying the balance expiry extension parameter. A configuration item (defaulting to 'false') will cause the request to fail if the balance extension parameter exceeds the wallet extension parameter, or if the result of extending the balance expiry date exceeds the current or extended wallet expiry date; this expiry indicator will be passed through to the UBE. If the balance expiry date is not being extended, then this check cannot be performed.

The specified balance type will be debited and the expiry date for the balance type and for the wallet will be set based on the rules detailed in Extending balance and wallet expiry dates, albeit the ‘override balance wallet value’ flag is not supported for this request.

If the balance type cascade is specified, then the balances will be debited in order of the cascade; specifying a balance expiry extension is not supported in conjunction with specifying a balance type cascade.

The current reservation, if one exists, will not be affected through invoking this request.

Note: The subscriber's wallet may not be available on the local SCS database if the user is not a prepaid subscriber. For post-paid subscribers the user can be 'cold-billed' using the information extracted from the OSA EDR. If the MSISDN is not found when the chargeParty is decoded and the 'AllowPostPaid' flag is set to true and the rating tariff for this subscriber can be determined, then the request will return success. Otherwise, if the chargeParty fails to decode successfully and the 'AllowPostPaid' flag is set to false and/or the rating tariff for this subscriber can not be determined, then the request will return failure.

This method also sets the Application Charging Session reference - all call-backs from the specified session will be sent to the new reference.

Parameters

Here is a list of the parameters for this method.

- **sessionID** (on page 168)
- **appChargingSession** (on page 166)
- **applicationDescription** (on page 166)
- **chargingParameters** (on page 167)
- **volumes** (on page 168)
- *requestNumber* (on page 167)

**Note:** The *requestNumber* parameter may be given the value of zero (0) and then the CHAM will use the currently stored value for the request number.

### Unit-Seconds based debit request

The treatment and expected type of each element within each TpChargingParameter structure for a unit-seconds based direct delta is:

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| ChargedParty      | TpString     | **Name:** P_CHS_CHARGE_PARTY  
Mandatory.  
ChargedType 1: The TpString will correspond to the MSISDN to charge.  
ChargedType 3: The TpString will correspond to the wallet ID and billing engine ID to charge.  
The wallet ID and billing engine ID is separated by '/'. |
| ChargedType       | TpInt32      | **Name:** P_CHS_CHARGE_TYPE  
Mandatory.  
Value showing type of ChargedParty.  
- 1 - MSISDN  
- 2 - reserved  
- 3 - Wallet ID |
| Calling Number    | TpString     | **Name:** P_CHS_CALLING_NUMBER  
Mandatory.  
Network Calling Number to be used for rating-tariff calculations. |
| Called Number     | TpString     | **Name:** P_CHS_CALLED_NUMBER  
Mandatory.  
Network Called Number to be used for rating-tariff calculations. |
| Discount Override | TpInt32      | **Name:** P_CHS_DISCOUNT_OVERRIDE  
Percentage discount to apply to final cost.  
Valid values are 0 – 10000 where these are in 1/100th of a percent.  
Therefore, 500 would equate to a 5% discount.  
The discount applied to the final cost will depend on the 'discountRuleType' parameter set in the CCS.ReservationHandler section of eserv.config file on the UBE.  
The following are valid values where the resulting discounted cost will be applied:  
- discountRuleType = 'ServiceOverride'  
  (use only OSA discount)  
  \[ finalCost = initialCost \times (OSA\ discount) \] |
<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>discountRuleType = 'S<em>R</em>W'</td>
<td></td>
<td>finalCost = initialCost * [OSA discount * CCS rating discount * CCS wallet discount]</td>
</tr>
<tr>
<td>discountRuleType = 'S+R+W'</td>
<td></td>
<td>finalCost = initialCost * [OSA discount + CCS rating discount + CCS wallet discount]</td>
</tr>
<tr>
<td>discountRuleType = 'S+R*W'</td>
<td></td>
<td>finalCost = initialCost * [OSA discount + CCS rating discount * CCS wallet discount]</td>
</tr>
<tr>
<td>discountRuleType = 'S*R+W'</td>
<td></td>
<td>finalCost = initialCost * [OSA discount * CCS rating discount + CCS wallet discount]</td>
</tr>
<tr>
<td>discountRuleType = 'S+W*R'</td>
<td></td>
<td>finalCost = initialCost * [OSA discount + CCS wallet discount * CCS rating discount]</td>
</tr>
<tr>
<td>discountRuleType = 'S*W+R'</td>
<td></td>
<td>finalCost = initialCost * [OSA discount * CCS wallet discount + CCS rating discount]</td>
</tr>
<tr>
<td>discountRuleType = 'R+W*S'</td>
<td></td>
<td>finalCost = initialCost * [CCS rating discount + CCS wallet discount * OSA discount]</td>
</tr>
<tr>
<td>discountRuleType = 'R*W+S'</td>
<td></td>
<td>finalCost = initialCost * [CCS rating discount * CCS wallet discount + OSA discount]</td>
</tr>
</tbody>
</table>

**Wallet Type Name**

**TpString**

**Name:** P_CHS_WALLET_TYPE

Optional. The wallet type name to use for Charge Type 1. If not specified, the default wallet type for this account is used.
<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Balance Type Name               | TpString     | Name: P_CHS_BALANCE_TYPE_NAME  
Optional. The balance type name to use for Charge Type 1. If: 
- not specified, there will be no default balance type for this operation. 
- specified then a balance type cascade containing only this balance type will override the balance type cascade already defined for the rating definition. |
| Balance Type Cascade Name       | TpString     | Name: P_CHS_BALANCE_TYPE_CASCADE_NAME  
Optional. The balance type cascade name to use for Charge Type 1. If specified, this balance type cascade will override the balance type cascade already defined for the rating definition. |
| Balance Validity Modification   | TpInt32      | Name: P_CHS_BALANCE_VALIDITY_MODIFICATION_NAME  
Optional. The number of days when extending the balance expiry date. The new balance expiry date will be set to the current date and time plus the number of days entered here. Only those expiry dates within the determined balance type (entered balance type), which would be increased by this modification, will be affected. Any expiry date within the determined balance type that is set to 'never expire' will not be affected by this request.  
The balance type parameter must be present for this parameter to be taken into account. |
| Wallet Validity Modification    | TpInt32      | Name: P_CHS_WALLET_VALIDITY_MODIFICATION_NAME  
Optional. The number of days when extending the wallet expiry date. The new wallet expiry date will be set to the current date and time plus the number of days entered here. The wallet expiry cannot be decreased by this modification. If the expiry date within the determined wallet type is set to 'never expire', it will not be affected by this request.  
The balance type parameter must be present for this parameter to be taken into account. |
### Element Name | Element Type | Description
--- | --- | ---
Send SMS Flag | TpBoolean | **Name:** P_CHS_SEND_SMS_FLAG
Optional. This can be set to 'TRUE' to ensure an SMS is sent when this reservation is confirmed on the billing engine (SMS only invoked if the ChargedType equals '1', otherwise this parameter is ignored). Defaults to 'FALSE'.

Allow Negative Balance Flag | TpBoolean | **Name:** P_CHS_ALLOW_NEGATIVE_BALANCE
Optional. This can be set to 'TRUE' to set the 'IgnoreWalletBalance' flag in the UBE request message. Defaults to 'FALSE'.

#### Unit-Billable event based debit request

The treatment and expected type of each element within each TpChargingParameter structure for a unit-billable based direct delta is:

### Element Name | Element Type | Description
--- | --- | ---
ChargedParty | TpString | **Name:** P_CHS_CHARGE_PARTY
Mandatory.
ChargedType 1: The TpString will correspond to the MSISDN to charge.
ChargedType 3: The TpString will correspond to the wallet ID and billing engine ID to charge. The wallet ID and billing engine ID is separated by '/'.

ChargedType | TpInt32 | **Name:** P_CHS_CHARGE_TYPE
Mandatory. Value showing type of ChargedParty.
- 1 - MSISDN
- 2 - reserved
- 3 - Wallet ID

Discount Override | TpInt32 | **Name:** P_CHS_DISCOUNT_OVERRIDE
Percentage discount to apply to final cost. Valid values are 0 – 10000 where these are in 1/100th of a percent. Therefore, 500 would equate to a 5% discount.
The discount applied to the final cost will depend on the OSA discount entered here.

**Note:** CCS discounts do not apply to billable events.

Event Name | TpString | **Name:** P_CHS_EVENT_NAME
Mandatory. Event Class and Name for rating tariff calculation.
The TpString will correspond to the event class and event name to charge. The event class and event name are separated by '/'.

---

192  |  NCC OSA CHAM Technical Guide
<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Wallet Type Name             | TpString     | Name: P_CHS_WALLET_TYPE  
Optional. The wallet type name to use for Charge Type 1. If not specified, the default wallet type for this account is used.                                                                        |
| Balance Type Name            | TpString     | Name: P_CHS_BALANCE_TYPE  
Optional. The balance type name to use for Charge Type 1. If:  
- not specified, there will be no default balance type for this operation.  
- specified then a balance type cascade containing only this balance type will override the balance type cascade already defined for the rating definition. |
| Balance Type Cascade Name    | TpString     | Name: P_CHS_BALANCE_TYPECASCADE  
Optional. The balance type cascade name to use for Charge Type 1. If specified, then this balance type cascade will override the balance type cascade already defined for the billable-event definition. |
| Balance Validity Modification| TpInt32      | Name: P_CHS_BALANCE_VALIDITY_MODIFICATION  
Optional. The number of days when extending the balance expiry date. The new balance expiry date will be set to the current date and time plus the number of days entered here. Only those expiry dates within the determined balance type (entered balance type), which would be increased by this modification, will be affected. Any expiry date within the determined balance type that is set to 'never expire' will not be affected by this request. The balance type parameter must be present for this parameter to be taken into account. |
| Wallet Validity Modification | TpInt32      | Name: P_CHS_WALLET_VALIDITY_MODIFICATION  
Optional. The number of days when extending the wallet expiry date. The new wallet expiry date will be set to the current date and time plus the number of days entered here. The wallet expiry cannot be decreased by this modification. If the expiry date within the determined wallet type is set to 'never expire', it will not be affected by this request. The balance type parameter must be present for this parameter to be taken into account. |
<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send SMS Flag</td>
<td>TpBoolean</td>
<td>Name: P_CHS_SEND_SMS_FLAG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional. This can be set to 'TRUE' to ensure an SMS is sent when this</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reservation is confirmed on the billing engine (SMS only invoked if the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ChargedType equals '1', otherwise this parameter is ignored). Defaults</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to 'FALSE'.</td>
</tr>
<tr>
<td>Allow Negative Balance Flag</td>
<td>TpBoolean</td>
<td>Name: P_CHS_ALLOW_NEGATIVE_BALANCE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional. This can be set to 'TRUE' to set the 'IgnoreWalletBalance' flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>in the UBE request message. Defaults to 'FALSE'.</td>
</tr>
</tbody>
</table>

### Returns

Nothing.

### ExtendLifeTimeWithAppChargingSessionReq()

#### Description

This method is invoked on the IpChargingSession to extend the lifetime of the reservation on both the OSA CHAM SCS gateway and on the billing engine for amount-based reservations only. The reservation is extended by the value currently stored on the OSA CHAM SCS gateway entered as the 'ReservationLifeTimeHint' parameter of the TpChargingParameter structure for this reservation. It is not possible to extend the lifetime for a unit-based reservation using the extendLifeTimeReq() method.

**Note:** extendLifetimeReq is only used with amount based reservations. It is not possible to use extendLifetimeReq with unit based reservations.

This method also sets the Application Charging Session reference - all call-backs from the specified session will be sent to the new reference.

#### Parameters

Here is a list of the parameters for this method.

- `sessionId` (on page 168)
- `appChargingSession` (on page 166)

#### Returns

Nothing.

### GetAmountLeftWithAppChargingSession()

#### Description

This method is invoked on the IpAccountManager to retrieve the amount stored in the internal table on the OSA CHAM SCS gateway remaining for this reservation. This method can only be invoked for amount-based reservations.
This method also sets that Application Charging Session reference - all call-backs from the specified session will be sent to the new reference.

**Parameters**

Here is a list of the parameters for this method.

- `sessionID` (on page 168)
- `appChargingSession` (on page 166)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TpChargingPrice</code></td>
<td>this is a structure containing the currency and a <code>TpAmount</code> structure. The currency is returned by the OSA CHAM SCS gateway as an empty string. (See standard [5] for <code>TpAmount</code> encoding. The 'Exponent' of the <code>TpAmount</code> structure will always be set to '0' and the 'Number' of the <code>TpAmount</code> structure will always be the number of small units of currency where the billing engine system currency is used).</td>
</tr>
</tbody>
</table>

**GetLifeTimeLeftWithAppChargingSession()**

**Description**

This method is invoked on the `IpAccountManager` to retrieve the lifetime remaining on the OSA CHAM SCS gateway for this reservation. This method can be invoked for both amount-based and unit-based reservations.

This method also sets the Application Charging Session reference - all call-backs from the specified session will be sent to the new reference.

**Parameters**

Here is a list of the parameters for this method.

- `sessionID` (on page 168)
- `appChargingSession` (on page 166)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>TpInt32</code></td>
<td>an unsigned integer corresponding to the number of seconds that this reservation will remain active.</td>
</tr>
</tbody>
</table>

**GetUnitLeftWithAppChargingSession()**

**Description**

This method will be invoked on the `IpChargingSession` to retrieve the units stored in the internal table on the OSA CHAM SCS gateway remaining for this reservation. This method can only be invoked for unit-based reservations.
This method also sets the Application Charging Session reference - all call-backs from the specified session will be sent to the new reference.

**Parameters**

Here is a list of the parameters for this method.

- `sessionID` (on page 168)
- `appChargingSession` (on page 166)

**Returns**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TpVolumeSet</td>
<td>A TpVolumeSet structure containing a single TpVolume which is composed of a TpUnitID value and a TpAmount structure. The TpUnitID value must be either P_CHS_UNIT_NUMBER (named events) or P_CHS_UNIT_SECONDS (time-based in seconds). (See standard [5] for TpAmount encoding. The 'Exponent' of the TpAmount structure will always be set to '0' and the 'Number' of the TpAmount structure will always be the number of units (named events or seconds) ).</td>
</tr>
</tbody>
</table>

**RateWithAppChargingSessionReq()**

**Description**

This method will be invoked on the IpChargingSession to determine the individual rates that are associated with this user's account. The rates will have the associated discount applied prior to returning them to the third-party client application. This method can only be invoked for unit-based reservations.

This method also sets the Application Charging Session reference - all call-backs from the specified session will be sent to the new reference.

**Parameters**

Here is a list of the parameters for this method.

- `sessionID` (on page 168)
- `appChargingSession` (on page 166)
- `chargingParameters` (on page 167)

**TpChargingParameter structure**

The treatment and expected type of each element within each TpChargingParameter structure for rate request is:-

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| ChargedParty    | TpString     | Name: P_CHS_CHARGE_PARTY  
Mandatory.  
ChargedType 1: The TpString will correspond to the MSISDN to charge.  
ChargedType 3: The TpString will correspond to the wallet ID and billing engine ID to charge.  
The wallet ID and billing engine ID is separated by '/'. |
<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChargedType</td>
<td>TpInt32</td>
<td>Name: P_CHS_CHARGE_TYPE Mandatory. Value showing type of ChargedParty.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 - MSISDN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 2 - reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 3 - Wallet ID</td>
</tr>
<tr>
<td>Calling Number</td>
<td>TpString</td>
<td>Name: P_CHS_CALLING_NUMBER Optional. Network Calling Number to be used for rating-tariff calculations</td>
</tr>
<tr>
<td>Called Number</td>
<td>TpString</td>
<td>Name: P_CHS_CALLED_NUMBER Optional. Network Called Number to be used for rating-tariff calculations.</td>
</tr>
<tr>
<td>Discount Override</td>
<td>TpInt32</td>
<td>Name: P_CHS_DISCOUNT_OVERRIDE Percentage discount to apply to final cost. Valid values are 0 – 10000 where these are in 1/100th of a percent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Therefore, 500 would equate to a 5% discount.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The discount applied to the final cost will depend on the ‘overrideDiscountType’ parameter set in the CCS.ReservationHandler section of eserv.config file found on the BE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The following are valid values where the resulting discounted cost will be applied:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> CCS discounts do not apply to named events</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• overrideDiscountType = ‘Override’ (ignore OSA discount)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• finalCost = initialCost * (CCS discount for seconds only)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• overrideDiscountType = ‘Compound’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>finalCost = initialCost * [CCS discount for seconds only * OSA discount]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• overrideDiscountType = ‘Cumulative’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>finalCost = initialCost * [CCS discount for seconds only + OSA discount]</td>
</tr>
<tr>
<td>Event Name</td>
<td>TpString</td>
<td>Name: P_CHS_EVENT_NAME Optional. Event Class and Name for rating tariff calculation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The TpString will correspond to the event class and event name to charge. The event class and event name will be separated by ‘/’.</td>
</tr>
<tr>
<td>Element Name</td>
<td>Element Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Wallet Type Name     | TpString     | Name: P_CHS_WALLET_TYPE  
Optional. The wallet type to use for Charge Type 1. If not specified, the default wallet type for this account is used.                                                                                     |

**Returns**

Nothing.

**ReserveAmountWithAppChargingSessionReq()**

**Description**

This method is invoked on the IpChargingSession to perform one of the following:

- This method is invoked on the IpChargingSession to make an initial monies-based reservation.
  
  A monies-based reservation will reserve monies from the subscriber's wallet using cash balance types only and store this amount in an internal table on the OSA CHAM SCS gateway to be debited and/or credited using the credit(debit)AmountReq() methods. The incoming request will specify the number of UBE small currency monies to be reserved. Typically this method is used for balance types where the unit type of the balance type is 'Cash'; if the balance type cascade name is specified in the incoming request then only balance types where the unit type of the balance type is 'Cash' will be available in the balance type cascade.

- This method is invoked on the IpChargingSession to make an initial time-based reservation.
  
  A time-based reservation will reserve time from the subscriber's wallet using time balance types only and store this amount in an internal table on the OSA CHAM SCS gateway to be debited and/or credited using the credit(debit)AmountReq() methods. The incoming request will specify the number of seconds to be reserved. The internal UBE time balance units is 100ths seconds so a conversion will be required. The balance type name must be specified in the incoming request where the unit type of the balance type is 'Time' and the allowNonCashBalanceForAmountMethods (on page 18) configuration parameter must be 'true' for time-based reservations to be supported. If the balance type cascade name is specified in the incoming request then the amount specified will be assumed to be monies and only balance types where the unit type of the balance type is 'Cash' will be available in the balance type cascade.

- This method is invoked on the IpChargingSession to make an initial non-cash or time-based reservation.
  
  A non-cash or time-based reservation will reserve time from the subscriber's wallet using non cash or time balance types only and store this amount in an internal table on the OSA CHAM SCS gateway to be debited and/or credited using the credit(debit)AmountReq() methods. The incoming request will specify the amount to be reserved; the internal UBE units are mapped directly to the specified amount. The balance type name must be specified in the incoming request where the unit type of the balance type is none 'Cash' or 'Time' and the allowNonCashBalanceForAmountMethods configuration parameter must be 'true' for non-cash or time-based reservations to be supported. If the balance type cascade name is specified in the incoming request then the amount specified will be assumed to be monies and only balance types where the unit type of the balance type is 'Cash' will be available in the balance type cascade.
Prior to making a confirmation against the reserved amount it is necessary to have debited an amount from the reservation. It is not possible to credit an amount towards a reservation unless prior debits greater to or equal to the credited amount have already been made - it is not possible to credit a subscriber's wallet through the reservation-confirmation process. If a reservation already exists for this user on the OSA CHAM SCS gateway, then invoking this method will result in a subsequent reservation of the specified amount for this user on the billing engine and incrementing the amount stored in this reservation on the OSA CHAM SCS gateway. If an existing unit-based reservation already exists for this user on the OSA CHAM SCS gateway then this request will fail, i.e. reservation types cannot be mixed within an active reservation. If the amount-based reservation is first closed, then a unit-based reservation can be created within the same charging session. When a reservation is attempted by a subscriber on the OSA_CHAM_SCS this reservation may fail if the maximum concurrent access to the subscriber's wallet would be exceeded through successfully creating this reservation. This maximum concurrent access parameter is a current property of the subscriber's wallet and is used by reservation attempts of all types on the subscriber's wallet, i.e. voice calls, other amount-based or unit-based reservations through the OSA_CHAM_SCS or billable event reservations using other applications. The amount of the reservation returned to the client will not be affected by the maximum concurrent access parameter i.e the OSA CHAM SCS will not divide the amount returned to the client by the maximum concurrent access parameter.

This method will also set the Application Charging Session reference - all call backs from the specified session will be sent to the new reference.

To ensure continuous performance of the UBE and UAS CHAM nodes, the collected clients should not reserve more that 10 concurrent amount or unit reservations on a single UBE wallet/subscriber.

**Parameters**

Here is a list of the parameters for this method.

- `sessionID` (on page 168)
- `appChargingSession` (on page 166)
- `applicationDescription` (on page 166)
- `chargingParameters` (on page 167)
- `preferredAmount` (on page 167)
- `minimumAmount` (on page 167)
- `requestNumber` (on page 167)

**Note:** The `requestNumber` parameter may be given the value of zero (0) and then the CHAM will use the currently stored value for the request number.

**TpChargingParameter structure**

The treatment and expected type of each element within each TpChargingParameter structure is:-

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChargedParty</td>
<td>TpString</td>
<td>Name: P_CHS_CHARGE_PARTY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mandatory.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ChargedType 1: The TpString will correspond to the MSISDN to charge.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ChargedType 3: The TpString will correspond to the wallet ID and billing engine ID to charge. The wallet ID and billing engine ID are separated by ‘/’.</td>
</tr>
<tr>
<td>Element Name</td>
<td>Element Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ChargedType</td>
<td>TpInt32</td>
<td>Name: P_CHS_CHARGE_TYPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mandatory. Value showing type of ChargedParty.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 - MSISDN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 - reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 - Wallet ID</td>
</tr>
<tr>
<td>Discount Override</td>
<td>TpInt32</td>
<td>Name: P_CHS_DISCOUNT_OVERRIDE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Percentage discount to apply to final cost. Valid values are 0 – 10000 where</td>
</tr>
<tr>
<td></td>
<td></td>
<td>these are in 1/100th of a percent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Therefore, 500 would equate to a 5% discount.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The discount applied to the final cost will depend solely on the OSA discount</td>
</tr>
<tr>
<td>ReservationLifeTimeHint</td>
<td>TpInt32</td>
<td>Name: P_CHS_RESERVATION_LIFETIME_HINT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional. The lifetime of the reservation in seconds - set on both the OSA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHAM SCS gateway and the billing engine server. If the reservation times out</td>
</tr>
<tr>
<td></td>
<td></td>
<td>on both the SCS and the BE, outstanding debits are not confirmed on the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>billing engine. The value entered must be greater than '0'. If not specified,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>then the default lifetime for the reservation will be '30' seconds.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>An external client should not set this value to more than 24 hours to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ensure that a series of outstanding concurrent reservations would not</td>
</tr>
<tr>
<td></td>
<td></td>
<td>accumulate after a period of network instability. The client should set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>this value to the period normally associated with use period of the charge.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For example charging for a bus ticket maybe instant, but a movie download</td>
</tr>
<tr>
<td></td>
<td></td>
<td>may take a period of hours.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For a subsequent reservation request, if the reservation lifetime is set to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'0', then the reservation will be extended by the 'ReservationLifeTimeHint'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>currently stored on the OSA CHAM SCS gateway from the initial reservation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>request.</td>
</tr>
<tr>
<td>Wallet Type Name</td>
<td>TpString</td>
<td>Name: P_CHS_WALLET_TYPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional. The wallet type to use for Charge Type 1. If not specified, the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>default wallet type for this account is used.</td>
</tr>
<tr>
<td>Balance Type Name</td>
<td>TpString</td>
<td>Name: P_CHS_BALANCE_TYPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional. The balance type to use for Charge Type 1. If not specified, the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>default balance type for this account is used.</td>
</tr>
<tr>
<td>Balance Type Cascade</td>
<td>TpString</td>
<td>Name: P_CHS_BALANCE_TYPE_CASCADE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional. The balance type cascade name to use for Charge Type 1. If:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>specifed then the balance type name will be ignored.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>not specified, the default balance type for this account will be used.</td>
</tr>
<tr>
<td>Element Name</td>
<td>Element Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Send SMS Flag</td>
<td>TpBoolean</td>
<td>Name: P_CHS_SEND_SMS_FLAG Optional. This can be set to 'TRUE' to ensure an SMS is sent when this reservation is confirmed on the billing engine (SMS only invoked if the ChargedType equals '1', otherwise this parameter is ignored). Defaults to 'FALSE'.</td>
</tr>
</tbody>
</table>

Returns
Nothing.

ReserveUnitWithAppChargingSessionReq()

Description
This method is invoked on the IpChargingSession to make an initial unit-based reservation. Each unit has an associated cost on the billing engine and therefore a specified number of units translate into a fixed amount of monies against which the reservation can be made on the subscriber's wallet. The only supported unit-based reservations are seconds and named events, where unit type cannot be changed during the lifetime of a reservation.

For unit-based reservation the OSA CHAM SCS gateway has no knowledge of monies and deals solely with the unit of the reservation. The number of units reserved is stored in an internal table on the OSA CHAM SCS gateway to be debited and/or credited using the credit(debit)UnitReq() methods. Prior to making a confirmation against the reserved units it is necessary to have debited units from the reservation. It is not possible to credit units towards a reservation unless prior debits greater to or equal to the credited units have already been made - it is not possible to credit a subscriber's wallet through the reservation-confirmation process.

If a reservation already exists for this user on the OSA CHAM SCS gateway, then invoking this method will result in a subsequent reservation of units for this user on the billing engine and incrementing the number of units stored in this reservation on the OSA CHAM SCS gateway. If an existing amount-based reservation already exists for this user on the OSA CHAM SCS gateway then this request will fail i.e. reservation types cannot be mixed within an active reservation. If the unit-based reservation is first closed, then an amount-based reservation can be created within the same charging session.

When a reservation is attempted by a subscriber on the OSA_CHAM_SCS this reservation may fail if the maximum concurrent access to the subscriber's wallet would be exceeded through successfully creating this reservation. This maximum concurrent access parameter is a current property of the subscriber's wallet and is used by reservation attempts of all types on the subscriber's wallet, i.e. voice calls, other amount-based or unit-based reservations through the OSA_CHAM SCS or named event reservations using other applications. The amount of the reservation returned to the client may be affected by the maximum concurrent access parameter in the same manner as currently seen for voice calls or named event reservations as follows: the amount returned for voice calls or reserved-unit requests for seconds-based reservations will be affected by the values configured on the UBE for the 'CCS.reservationHandler.maxReservationLength' and 'CCS.reservationHandler.greedyReservationLengthLimit' where the reservation length returned will be recalculated as the lesser of the maxReservationLength or the amount that the subscriber can afford if the greedyReservationLengthLimit is not initially satisfied.

This method also sets the Application Charging Session reference - all call-backs from the specified session will be sent to the new reference.
To ensure continuous performance of the UBE and OSA CHAM nodes, the collected clients should not reserve more than 10 concurrent amount or unit reservations.

**Parameters**

Here is a list of the parameters for this method.

- `sessionID` (on page 168)
- `appChargingSession` (on page 166)
- `applicationDescription` (on page 166)
- `chargingParameters` (on page 167)
- `volumes` - see Unit-Seconds Based Reservation and Unit-Named Event Based Reservation
- `requestNumber` (on page 167)

**Note:** The `requestNumber` parameter may be given the value of zero (0) and then the CHAM will use the currently stored value for the request number.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>volumes</td>
<td>a TpVolumeSet structure containing a single TpVolume which is composed of a TpUnitID value and a TpAmount structure. The TpUnitID value is P_CHS_UNIT_SECONDS. (See standard [5] for TpAmount encoding. The ‘Exponent’ of the TpAmount structure will always be set to ‘0’ and the ‘Number’ of the TpAmount structure will always be ignored by this reservation). The ‘Number’ field of the TpAmount structure must be greater than zero.</td>
</tr>
</tbody>
</table>

The treatment and expected type of each element within each TpChargingParameter structure for a unit-seconds based reservation is:-

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| ChargedParty      | TpString     | Name: P_CHS_CHARGE_PARTY  
Mandatory.  
ChargedType 1: The TpString will correspond to the MSISDN to charge.  
ChargedType 3: The TpString will correspond to the wallet ID and billing engine ID to charge. The wallet ID and billing engine ID is separated by ‘/’. |
| ChargedType       | TpInt32      | Name: P_CHS_CHARGE_TYPE  
Mandatory. Value showing type of ChargedParty.  
- 1 - MSISDN  
- 2 - reserved  
- 3 - Wallet ID |
| Calling Number    | TpString     | Name: P_CHS_CALLING_NUMBER  
Mandatory. Network Calling Number to be used for rating-tariff calculations. |
| Called Number     | TpString     | Name: P_CHS_CALLED_NUMBER  
Mandatory. Network Called Number to be used for rating-tariff calculations. |
<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Discount Override | TpInt32      | Name: P_CHS_DISCOUNT_OVERRIDE  
Percentage discount to apply to final cost. Valid values are 0 – 10000 where these are in 1/100th of a percent. Therefore, 500 would equate to a 5% discount. The discount applied to the final cost will depend on the 'discountRuleType' parameter set in the CCS.ReservationHandler section of eserv.config file on the UBE. The following are valid values where the resulting discounted cost will be applied:
- discountRuleType = 'ServiceOverride' (use only OSA discount)  
  \[ \text{finalCost} = \text{initialCost} \times (\text{OSA discount}) \]
- discountRuleType = 'S*R*W'  
  \[ \text{finalCost} = \text{initialCost} \times (\text{OSA discount} \times \text{CCS rating discount} \times \text{CCS wallet discount}) \]
- discountRuleType = 'S+R+W'  
  \[ \text{finalCost} = \text{initialCost} \times (\text{OSA discount} + \text{CCS rating discount} + \text{CCS wallet discount}) \]
- discountRuleType = 'S+R*W'  
  \[ \text{finalCost} = \text{initialCost} \times (\text{OSA discount} + \text{CCS rating discount} \times \text{CCS wallet discount}) \]

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>discountRuleType = 'S*R+W'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>finalCost = initialCost * [OSA discount * CCS rating discount + CCS wallet discount]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>discountRuleType = 'S+W*R'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>finalCost = initialCost * [OSA discount + CCS wallet discount * CCS rating discount]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>discountRuleType = 'S+W+R'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>finalCost = initialCost * [OSA discount * CCS wallet discount + CCS rating discount]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>discountRuleType = 'R+W*S'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>finalCost = initialCost * [CCS rating discount + CCS wallet discount * OSA discount]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>discountRuleType = 'R+W+S'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>finalCost = initialCost * [CCS rating discount * CCS wallet discount + OSA discount]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ReservationLifeTimeHint</th>
<th>TpInt32</th>
<th>Name: P_CHS RESERVATION_LIFETIME_HINT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional. The lifetime of the reservation in seconds - set on both the OSA CHAM SCS gateway and the billing engine server. A hint is specified as the billing server may choose to ignore this lifetime. If the reservation times out on both the SCS and the BE, outstanding debits are not confirmed on the billing engine. If not specified, then the default lifetime passed to the billing engine for the reservation is '30' seconds. For an initial reservation, the reservation lifetime is set to the number of units (seconds) that the billing engine returns in the acknowledgement (this will not be the number of units specified in the volumes parameter of the original request as this value is ignored). If the lifetime requested is less than the billing engine minimum call length then the acknowledgement will return a reservation of zero. This will close the session immediately. For a subsequent reservation request, this parameter is ignored and the reservation lifetime is extended by the number of units (seconds) that the billing engine returns in the acknowledgement for the (this will not be the number of units specified in the volumes parameter of the original request as this value is ignored).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Element Name</td>
<td>Element Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Wallet Type Name</td>
<td>TpString</td>
<td>Name: P_CHS_WALLET_TYPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional. The wallet type name to use for Charge Type 1. If not specified,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the default wallet type for this account is used.</td>
</tr>
<tr>
<td>Send SMS Flag</td>
<td>TpBoolean</td>
<td>Name: P_CHS_SEND_SMS_FLAG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional. This can be set to 'TRUE' to ensure an SMS is sent when this</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reservation is confirmed on the billing engine (SMS only invoked if the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ChargedType equals ‘1’, otherwise this parameter is ignored). Defaults</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to ‘FALSE’.</td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>volumes</td>
<td>A TpVolumeSet structure containing a single TpVolume which is composed of a TpUnitID value and a TpAmount structure. The TpUnitID value must be P_CHS_UNIT_NUMBER. (See standard [5] for TpAmount encoding. The 'Exponent' of the TpAmount structure will always be set to '0' and the 'Number' of the TpAmount structure will always be the number of named events to be used for this reservation). The 'Number' field of the TpAmount structure must be greater than zero.</td>
</tr>
</tbody>
</table>

The treatment and expected type of each element within each TpChargingParameter structure for a unit-named event based reservation is:-

<table>
<thead>
<tr>
<th>Element Name</th>
<th>Element Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChargedParty</td>
<td>TpString</td>
<td>Name: P_CHS_CHARGE_PARTY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mandatory. ChargedType 1: The TpString will correspond to the MSISDN to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>charge.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ChargedType 3: The TpString will correspond to the wallet ID and billing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>engine ID to charge. The wallet ID and billing engine ID is separated by</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'='.</td>
</tr>
<tr>
<td>ChargedType</td>
<td>TpInt32</td>
<td>Name: P_CHS_CHARGE_TYPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mandatory. Value showing type of ChargedParty.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 - MSISDN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 2 - reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 3 - Wallet ID</td>
</tr>
<tr>
<td>Element Name</td>
<td>Element Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| Discount Override       | TpInt32      | Name: P_CHS_DISCOUNT_OVERRIDE  
Percentage discount to apply to final cost.  
Valid values are 0 – 10000 where these are in  
1/100th of a percent.  
Therefore, 500 would equate to a 5%  
discount.  
The discount applied to the final cost will  
depend solely on the OSA discount entered  
here.  

**Note:** CCS discounts do not apply to billable  
events. |
| Event Name              | TpString     | Name: P_CHS_EVENT_NAME  
Mandatory. Event Class and Name for rating  
tariff calculation.  
The TpString will correspond to  
the event class and event name to charge. The event  
class and event name are separated by '/'. |
| ReservationLifeTimeHint | TpInt32      | Name: P_CHS_RESERVATION_LIFETIME_HINT  
Optional. The lifetime of the reservation in  
seconds - set on both the OSA CHAM SCS  
gateway and the billing engine server. A hint  
is specified as the billing server may choose to  
ignore this lifetime. If the reservation times out  
on both the SCS and the BE, outstanding  
debits are not confirmed on the billing engine.  
If not specified, then the default lifetime  
passed to the billing engine for the reservation  
is '30' seconds.  
For an initial reservation, if the value entered  
is less than '0', then the default lifetime  
requested from the billing engine for the initial  
reservation will be '30' seconds.  
For a subsequent reservation request, this  
parameter is ignored and the reservation  
lifetime cannot be extended. |
| Wallet Type             | TpString     | Name: P_CHS_WALLET_TYPE  
Optional. The wallet type to use for Charge  
Type 1. If not specified, the default wallet type  
for this account is used. |
| Send SMS Flag           | TpBoolean    | Name: P_CHS_SEND_SMS_FLAG  
Optional. This can be set to 'TRUE' to ensure  
an SMS is sent when this reservation is  
confirmed on the billing engine (SMS only  
invoked if the ChargedType equals '1';  
otherwise this parameter is ignored). Defaults  
to 'FALSE'. |

**Returns**

Nothing.
Chapter 13

Application Charging Session Functionality

Overview

Introduction

This chapter explains the methods that are available using the IpAppChargingSession. The IpAppChargingSession object resides on the third-party client application and provides the following methods that can be invoked by the OSA CHAM SCS gateway.

In this chapter

This chapter contains the following topics.

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreditAmountErr ()</td>
<td>209</td>
</tr>
<tr>
<td>CreditAmountRes()</td>
<td>210</td>
</tr>
<tr>
<td>CreditUnitErr ()</td>
<td>210</td>
</tr>
<tr>
<td>CreditUnitRes()</td>
<td>210</td>
</tr>
<tr>
<td>DebitAmountErr ()</td>
<td>211</td>
</tr>
<tr>
<td>DebitAmountRes()</td>
<td>211</td>
</tr>
<tr>
<td>DebitUnitErr ()</td>
<td>211</td>
</tr>
<tr>
<td>DebitUnitRes()</td>
<td>212</td>
</tr>
<tr>
<td>DirectCreditAmountErr ()</td>
<td>212</td>
</tr>
<tr>
<td>DirectCreditAmountRes()</td>
<td>212</td>
</tr>
<tr>
<td>DirectCreditUnitErr ()</td>
<td>213</td>
</tr>
<tr>
<td>DirectCreditUnitRes ()</td>
<td>213</td>
</tr>
<tr>
<td>DirectDebitAmountErr ()</td>
<td>213</td>
</tr>
<tr>
<td>DirectDebitAmountRes ()</td>
<td>214</td>
</tr>
<tr>
<td>DirectDebitUnitErr ()</td>
<td>214</td>
</tr>
<tr>
<td>DirectDebitUnitRes ()</td>
<td>214</td>
</tr>
<tr>
<td>ExtendLifeTimeErr ()</td>
<td>215</td>
</tr>
<tr>
<td>ExtendLifeTimeRes ()</td>
<td>215</td>
</tr>
<tr>
<td>RateErr ()</td>
<td>215</td>
</tr>
<tr>
<td>RateRes()</td>
<td>216</td>
</tr>
<tr>
<td>ReserveAmountErr ()</td>
<td>216</td>
</tr>
<tr>
<td>ReserveAmountRes()</td>
<td>216</td>
</tr>
<tr>
<td>ReserveUnitErr ()</td>
<td>217</td>
</tr>
<tr>
<td>ReserveUnitRes()</td>
<td>217</td>
</tr>
<tr>
<td>SessionEnded ()</td>
<td>217</td>
</tr>
<tr>
<td>Application Charging Session Parameters</td>
<td>218</td>
</tr>
</tbody>
</table>

CreditAmountErr ()

Description

This method will be invoked on the IpAppChargingmanager as a result of an unsuccessful creditAmountReq().
Parameters

Here is a list of the parameters for this method.

- sessionID (on page 220)
- requestNumber (on page 219)
- error (on page 219)
- requestNumberNextRequest (on page 219)

CreditAmountRes()

Description

This method is invoked on the IpAppChargingmanager as a result of a successful creditAmountReq(). The information returned will be the amount of monies credited towards the reservation stored on the OSA CHAM SCS gateway and the amount of monies remaining in this reservation stored on the OSA CHAM SCS gateway after the request has been completed.

Parameters

Here is a list of the parameters for this method.

- sessionID (on page 220)
- requestNumber (on page 219)
- creditedAmount (on page 218)
- reservedAmountLeft
- requestNumberNextRequest (on page 219)

CreditUnitErr ()

Description

This method is invoked on the IpAppChargingmanager as a result of an unsuccessful creditUnitReq().

Parameters

Here is a list of the parameters for this method.

- sessionID (on page 220)
- requestNumber (on page 219)
- error (on page 219)
- requestNumberNextRequest (on page 219)

CreditUnitRes()  

Description

This method is invoked on the IpAppChargingmanager as a result of a successful creditUnitReq(). The information returned will be the number of units credited towards the reservation stored on the OSA CHAM SCS gateway and the number of units remaining in this reservation stored on the OSA CHAM SCS gateway after the request has been completed.
Parameters
Here is a list of the parameters for this method.

- sessionID (on page 220)
- requestNumber (on page 219)
- creditedVolumes (on page 218)
- reservedUnitsLeft (on page 220)
- requestNumberNextRequest (on page 219)

DebitAmountErr ()

Description
This method is invoked on the IpAppChargingmanager as a result of an unsuccessful debitAmountReq().

Parameters
Here is a list of the parameters for this method.

- sessionID (on page 220)
- requestNumber (on page 219)
- error (on page 219)
- requestNumberNextRequest (on page 219)

DebitAmountRes()

Description
This method is invoked on the IpAppChargingmanager as a result of a successful debitAmountReq(). The information returned will be the amount of monies debited from the reservation stored on the OSA CHAM SCS gateway and the amount of monies remaining in this reservation stored on the OSA CHAM SCS gateway after the request has been completed.

Parameters
Here is a list of the parameters for this method.

- sessionID (on page 220)
- requestNumber (on page 219)
- debitedAmount (on page 218)
- reservedAmountLeft
- requestNumberNextRequest (on page 219)

DebitUnitErr ()

Description
This method is invoked on the IpAppChargingmanager as a result of an unsuccessful debitUnitReq().
Parameters

Here is a list of the parameters for this method.

- `sessionID` (on page 220)
- `requestNumber` (on page 219)
- `error` (on page 219)
- `requestNumberNextRequest` (on page 219)

DebitUnitRes()

Description

This method is invoked on the IpAppChargingmanager as a result of a successful `debitUnitReq()`. The information returned will be the number of units debited from the reservation stored on the OSA CHAM SCS gateway and the number of units remaining in this reservation stored on the OSA CHAM SCS gateway after the request has been completed.

Parameters

Here is a list of the parameters for this method.

- `sessionID` (on page 220)
- `requestNumber` (on page 219)
- `debitedVolumes` (on page 219)
- `reservedUnitsLeft` (on page 220)
- `requestNumberNextRequest` (on page 219)

DirectCreditAmountErr()

Description

This method is invoked on the IpAppChargingmanager as a result of an unsuccessful `directCreditAmountReq()`.

Parameters

Here is a list of the parameters for this method.

- `sessionID` (on page 220)
- `requestNumber` (on page 219)
- `error` (on page 219)
- `requestNumberNextRequest` (on page 219)

DirectCreditAmountRes()

Description

This method is invoked on the IpAppChargingmanager as a result of a successful `directCreditAmountReq()`. The information returned will be the amount of monies credited towards the user’s account on the billing engine.
Parameters

Here is a list of the parameters for this method.

- `sessionID` (on page 220)
- `requestNumber` (on page 219)
- `creditedAmount` (on page 218)
- `requestNumberNextRequest` (on page 219)

DirectCreditUnitErr ()

Description

This method is invoked on the IpAppChargingmanager as a result of an unsuccessful directCreditUnitReq().

Parameters

Here is a list of the parameters for this method.

- `sessionID` (on page 220)
- `requestNumber` (on page 219)
- `error` (on page 219)
- `requestNumberNextRequest` (on page 219)

DirectCreditUnitRes ()

Description

This method is invoked on the IpAppChargingmanager as a result of a successful directCreditUnitReq(). The information returned will be the number of units that will translate into an amount of monies to credit towards the user’s account on the billing engine.

Parameters

Here is a list of the parameters for this method.

- `sessionID` (on page 220)
- `requestNumber` (on page 219)
- `creditedVolumes` (on page 218)
- `requestNumberNextRequest` (on page 219)

DirectDebitAmountErr ()

Description

This method is invoked on the IpAppChargingmanager as a result of an unsuccessful directDebitAmountReq().
Parameters

Here is a list of the parameters for this method.

- `sessionID` (on page 220)
- `requestNumber` (on page 219)
- `error` (on page 219)
- `requestNumberNextRequest` (on page 219)

DirectDebitAmountRes()

Description

This method is invoked on the IpAppChargingmanager as a result of a successful directDebitAmountReq(). The information returned will be the amount of monies debited from the user’s account on the billing engine.

Parameters

Here is a list of the parameters for this method.

- `sessionID` (on page 220)
- `requestNumber` (on page 219)
- `debitedAmount` (on page 218)
- `requestNumberNextRequest` (on page 219)

DirectDebitUnitErr()

Description

This method is invoked on the IpAppChargingmanager as a result of an unsuccessful directDebitUnitReq().

Parameters

Here is a list of the parameters for this method.

- `sessionID` (on page 220)
- `requestNumber` (on page 219)
- `error` (on page 219)
- `requestNumberNextRequest` (on page 219)

DirectDebitUnitRes()

Description

This method is invoked on the IpAppChargingmanager as a result of a successful directDebitUnitReq(). The information returned will be the number of units that will translate into an amount of monies to debit from the user’s account on the billing engine.
Parameters
Here is a list of the parameters for this method.

- `sessionID` (on page 220)
- `requestNumber` (see "requestNumber" on page 219)
- `debitedVolumes` (on page 219)
- `requestNumberNextRequest` (on page 219)

**ExtendLifeTimeErr ()**

**Description**
This method is invoked on the IpAppChargingmanager as a result of an unsuccessful `extendLifeTimeReq()`.

**Parameters**
Here is a list of the parameters for this method.

- `sessionID` (on page 220)
- `error` (on page 219)

**ExtendLifeTimeRes()**

**Description**
This method is invoked on the IpAppChargingmanager as a result of a successful `extendLifeTimeReq()`.
The information returned will be the session lifetime (in seconds) held open on both the OSA CHAM SCS gateway and the billing engine for this reservation after the request has been completed.

**Parameters**
Here is a list of the parameters for this method.

- `sessionID` (on page 220)
- `sessionTimeLeft` (on page 220)

**RateErr ()**

**Description**
This method is invoked on the IpAppChargingmanager as a result of an unsuccessful `rateReq()`.

**Parameters**
Here is a list of the parameters for this method.

- `sessionID` (on page 220)
- `error` (on page 219)
RateRes()

Description

This method is invoked on the IpAppChargingmanager as a result of a successful rateReq(). The information returned will depend on the unit type of the response message. If the value of the TpUnitID parameter in the TpVolume structure is:

- **P_CHS_UNIT_SECONDS**, then the information returned will be the individual rating tariffs (in small units of billing engine currency per minute) and the duration for each rating tariff (in seconds)
- **P_CHS_UNIT_NUMBER**, then the information returned will be a single rating tariff (in small units of billing engine currency per named event) and a duration for this rating tariff of ‘1’.

Parameters

Here is a list of the parameters for this method.

- `sessionId` (on page 220)
- `requestNumber` (on page 219)
- `rates` (on page 219)
- `validityTimeLeft` (on page 220)

ReserveAmountErr()

Description

This method will be invoked on the IpAppChargingmanager as a result of an unsuccessful reserveAmountReq().

Parameters

Here is a list of the parameters for this method.

- `sessionId` (on page 220)
- `requestNumber` (on page 219)
- `error` (on page 219)
- `requestNumberNextRequest` (on page 219)

ReserveAmountRes()

Description

This method will be invoked on the IpAppChargingmanager as a result of a successful reserveAmountReq(). The information returned will be the amount of monies reserved from the user’s account for either an initial or subsequent reservation. The session lifetime (in seconds) held open on both the OSA CHAM SCS gateway and the billing engine for this reservation will also be returned by this message.

Parameters

Here is a list of the parameters for this method.

- `sessionId` (on page 220)
- `requestNumber` (on page 219)
- `reservedAmount` (on page 220)
- `sessionTimeLeft` (on page 220)
- `requestNumberNextRequest` (on page 219)

### ReserveUnitErr()

**Description**

This method is invoked on the IpAppChargingManager as a result of an unsuccessful `reserveUnitReq()`.

**Parameters**

Here is a list of the parameters for this method.

- `sessionID` (on page 220)
- `requestNumber` (on page 219)
- `error` (on page 219)
- `requestNumberNextRequest` (on page 219)

### ReserveUnitRes()

**Description**

This method is invoked on the IpAppChargingManager as a result of a successful `reserveUnitReq()`.

Each unit has an associated cost on the billing engine and therefore a specified number of units translate into a fixed amount of monies against which the reservation can be made on the user’s account. The information returned will be the amount of units reserved from the user’s account for either an initial or subsequent reservation. The session lifetime (in seconds) held open on both the OSA CHAM SCS gateway and the billing engine for this reservation will also be returned by this message.

**Parameters**

Here is a list of the parameters for this method.

- `sessionID` (on page 220)
- `requestNumber` (on page 219)
- `reservedUnits` (on page 220)
- `sessionTimeLeft` (on page 220)
- `requestNumberNextRequest` (on page 219)

### SessionEnded()

**Description**

This method is invoked on the IpAppChargingManager as a result of the charging session expiring on the OSA CHAM SCS gateway. This is typically caused when the lifetime of the reservation expires. This will result in the following events on the OSA CHAM SCS gateway:

1. Any amount remaining in the current reservation will be returned to the user’s account and be available for alternative reservations from the third-party client application.
2 An OSA EDR (Type 22) will be created. If any funds have already been confirmed on the billing engine for this reservation prior to this transaction, and if the current reservation is:
- unit-seconds based, then a CCS EDR (Type 24) would have been created.
- unit-named event based, then a CCS EDR (Type 25) would have been created.

3 The charging session on the OSA CHAM SCS gateway will be removed from the internal table of active sessions. This charging session will no longer be valid for any requests from the third-party client application.

Parameters

Here is a list of the parameters for this method.

- `sessionID` (on page 220)
- `error` (on page 219)

Application Charging Session Parameters

Introduction

Here are the parameters used for this functionality.

**creditedAmount**

**Description:** A `TpChargingPrice` structure containing the currency and a `TpAmount` structure. The currency will be returned by the OSA CHAM SCS gateway as an empty string. (See standard [5] for `TpAmount` encoding. The ‘Exponent’ of the `TpAmount` structure will always be set to ‘0’ and the ‘Number’ of the `TpAmount` structure will always be the number of small units of currency where the billing engine system currency will be used). This is the amount of monies that was credited towards the reservation.

**creditedVolumes**

**Description:** A `TpVolumeSet` structure containing a single `TpVolume` which is composed of a `TpUnitID` value and a `TpAmount` structure. The `TpUnitID` value must be either `P_CHS_UNIT_NUMBER` (named events) or `P_CHS_UNIT_SECONDS` (time-based in seconds). (See standard [5] for `TpAmount` encoding. The ‘Exponent’ of the `TpAmount` structure will always be set to ‘0’ and the ‘Number’ of the `TpAmount` structure will always be the number of units (named events or seconds)). This is the number of units that was credited towards the reservation.

**debitedAmount**

**Description:** A `TpChargingPrice` structure containing the currency and a `TpAmount` structure. The currency will be returned by the OSA CHAM SCS gateway as an empty string. (See standard [5] for `TpAmount` encoding. The ‘Exponent’ of the `TpAmount` structure will always be set to ‘0’ and the ‘Number’ of the `TpAmount` structure will always be the number of small units of currency where the billing engine system currency will be used). This is the amount of monies that was debited from the reservation.
debitedVolumes
Description: A TpVolumeSet structure containing a single TpVolume which is composed of a TpUnitID value and a TpAmount structure. The TpUnitID value must be either P_CHS_UNIT_NUMBER (named events) or P_CHS_UNIT_SECONDS (time-based in seconds). (See standard [5] for TpAmount encoding. The ‘Exponent’ of the TpAmount structure will always be set to ‘0’ and the ‘Number’ of the TpAmount structure will always be the number of units (named events or seconds)). This is the number of units that was debited from the reservation.

error
Description: A TpChargingError value which is either a mapping of the error returned from the billing engine or the error resolved at the OSA CHAM SCS gateway for the invoked request. (See standard [5] for the TpChargingError encoding.)

Note: The error return codes are an explicit encoding predefined by the standard in [5], therefore the error returned to the OSA CHAM SCS gateway may not have a suitable mapped error definition. When this occurs P_CHS_ERR_UNDEFINED will be returned to the third-party client application.

rates
Description: A TpPriceVolumeSet structure which is a numbered set of elements containing a TpVolume structure and a TpChargingPrice structure. The TpVolume structure is composed of a TpUnitID value and a TpAmount structure. The TpUnitID value must be either P_CHS_UNIT_NUMBER (named events) or P_CHS_UNIT_SECONDS (time-based in seconds). (See standard [5] for TpAmount encoding. The ‘Exponent’ of the TpAmount structure will always be set to ‘0’ and the ‘Number’ of the TpAmount structure will be the duration (in seconds) of the corresponding rating tariff extracted in the TpChargingPrice structure if the TpUnitId value equals P_CHS_UNIT_SECONDS, or the ‘Number’ will always be ‘1’ if the TpUnitId value equals P_CHS_UNIT_NUMBER).

The TpChargingPrice structure contains the currency and a TpAmount structure. The currency will be returned by the OSA CHAM SCS gateway as an empty string. (See standard [5] for TpAmount encoding. The ‘Exponent’ of the TpAmount structure will always be set to ‘0’ and the ‘Number’ of the TpAmount structure will always be the number of small units of currency where the billing engine system currency will be used). This is the rating tariff that will be returned on a per minute basis or on a per named event basis depending on the value of the TpUnitID parameter in the TpVolume structure.

requestNumber
Description: The request number (unsigned integer) of the individual request within this charging session.

requestNumberNextRequest
Description: An unsigned integer corresponding to the request number that the third-party client application should use for the next request on this charging session.
reservedAmount

Description: A TpChargingPrice structure containing the currency and a TpAmount structure. The currency will be returned by the OSA CHAM SCS gateway as an empty string. (See standard [5] for TpAmount encoding. The ‘Exponent’ of the TpAmount structure will always be set to ‘0’ and the ‘Number’ of the TpAmount structure will always be the number of small units of currency where the billing engine system currency will be used). If cascade has been specified, the number will be the summation of the balance types used from the cascade, in small units.

reservedAmountLeft

Description: A TpChargingPrice structure containing the currency and a TpAmount structure. The currency will be returned by the OSA CHAM SCS gateway as an empty string. (See standard [5] for TpAmount encoding. The ‘Exponent’ of the TpAmount structure will always be set to ‘0’ and the ‘Number’ of the TpAmount structure will always be the number of small units of currency where the billing engine system currency will be used). This is the amount of monies remaining in this reservation after this request has been processed.

reservedUnits

Description: A TpVolumeSet structure containing a single TpVolume which is composed of a TpUnitID value and a TpAmount structure. The TpUnitID value must be either P_CHS_UNIT_NUMBER (named events) or P_CHS_UNIT_SECONDS (time-based in seconds). (See standard [5] for TpAmount encoding. The ‘Exponent’ of the TpAmount structure will always be set to ‘0’ and the ‘Number’ of the TpAmount structure will always be the number of units (named events or seconds)).

Note: For Unit Seconds requests, the reservedUnits returned will be set to the value of the sessionLifeTime left.

reservedUnitsLeft

Description: A TpVolumeSet structure containing a single TpVolume which is composed of a TpUnitID value and a TpAmount structure. The TpUnitID value must be either P_CHS_UNIT_NUMBER (named events) or P_CHS_UNIT_SECONDS (time-based in seconds). (See standard [5] for TpAmount encoding. The ‘Exponent’ of the TpAmount structure will always be set to ‘0’ and the ‘Number’ of the TpAmount structure will always be the number of units (named events or seconds)). This is the number of units remaining in this reservation after this request has been processed.

sessionID

Description: The session ID (unsigned integer) of this charging session.

sessionTimeLeft

Description: An unsigned integer corresponding to the number of seconds that this reservation will remain active.

validityTimeLeft

Description: A TpDuration structure corresponding to the number of milliseconds that this reservation will remain active. This parameter will not be used and a ‘0’ will be returned by the OSA CHAM SCS gateway. (See standard [2] for TpDuration encoding)
About Installation and Removal

Overview

Introduction

This chapter provides details of the installation and removal process for the application.

In this chapter

This chapter contains the following topics.

Installation and Removal Overview 223
Execution - osaChamScs 224
Auxiliary Scripts (SignalOsaChamScs.sh) 224

Installation and Removal Overview

Introduction

For information about the following requirements and tasks, see NCC Installation Guide:

- NCC system requirements
- Pre-installation tasks
- Installing and removing NCC packages

DAP packages

An installation of OSA CHAM includes the following packages, on the:

- SMS:
  - osaChamSms
- SLC:
  - osaChamScs
- VWS:
  - osaChamBe

Additional configuration of osaChamBe

Once the osaChamBe package has been successfully installed, the following process is required to complete the configuration of the service.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 1    | Check to see if startup scripts exist for the beVWARS and beServer in /IN/service_packages/E2BE/bin.  
  - If so, go to step 6,  
  - If not, complete steps 2-5, but omit step 6. |
| 2    | Create the startup scripts in the /IN/service_packages/E2BE/bin/ directory. |
Chapter 14

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 2    | An example of a startup script for the beServer (called IN/service_packages/E2BE/bin/beServerStartup.sh) may be as follows:  
```bash
#!/bin/sh
. /IN/service_packages/E2BE/.profile-be
exec /IN/service_packages/E2BE/bin/beServer >> /IN/service_packages/E2BE/tmp/beServer.log 2>&1
```
| Note: You must ensure that .profile-be exists. If not a warning will have been created when the osaChamBe package was installed. |
| 3    | Modify the SLEE.cfg, located in /IN/service_packages/SLEE/etc/SLEE.cfg, to call the startup script instead of the beServer binary directly, for example; by modifying the following line as shown:
- existing line:
  ```bash
  INTERFACE=beServer beServer /IN/service_packages/E2BE/bin EVENT
  ```
- modified line:
  ```bash
  INTERFACE=beServer beServerStartup.sh /IN/service_packages/E2BE/bin EVENT
  ```
| 4    | A startup script for each instance of the beVWARS will also need to be created in a similar fashion. |
| 5    | The lines which start each instance of the beVWARS will also need to be modified in a similar fashion. |
| 6    | Add the following line (if not already present) before the line which calls the binary, for example:
  ```bash
  'exec /IN/service_packages/E2BE/bin/beServer >> /IN/service_packages/E2BE/tmp/beServer.log 2>&1':
  . /IN/service_packages/E2BE/.profile-be
  ```
| Note: You must ensure that .profile-be exists. If not a warning will have been created when the osaChamBe package was installed. |

Execution - osaChamScs

Execution

The preferred method to execute the gateway is to install an inittab entry through the use of the osaChamScs.conf.sh script.

Alternatively, the following command may be executed as the osa_oper user:

```
/IN/service_packages/OSA/bin/runOsaChamScs.sh
```

To specify alternate configuration sections (in order to run multiple gateways) append the following command line arguments:

```
-s section name
```

Auxilliary Scripts (SignalOsaChamScs.sh)

Introduction

When the client interface package is installed, within the /IN/service_packages/OSA/bin directory, a script can be run by the user at runtime to send signals to the OSA Gateway process.
**Important:** All currently executing instances of the osaChamScs binary will be sent the same signal upon execution of this script.

### Procedure - aux scripts

Follow these steps to use this auxiliary scripts.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Type <code>cd /IN/service_packages/OSA/bin</code></td>
</tr>
<tr>
<td>2</td>
<td>Type <code>sh ./signalOsaChamScs.sh &lt;signal&gt;</code></td>
</tr>
</tbody>
</table>

**Where:** `<signal>` can be one of the following:

- **SIGHUP** – reread the ‘Oracle.conf’ configuration file.
- **SIGUSR1** – switch debug output on.
- **SIGUSR2** – switch debug output off (This is the default on start-up).
- **SIGINT** – shutdown the process cleanly. The client interface will complete the following:
  - Refuse any further requests from the client application.
  - Complete all the current open transactions.
  - Notify the billing engine and close any open connections to the billing engine.
  - Close the connection to ORACLE.
- **SIGKILL** – end process immediately (cannot be trapped).
- **SIGABRT** – abort process and produce core file.

### Debugging

Debug output is turned off by default.

All debug output will be sent to the standard output only, and not to the standard error, which is reserved for alarms.
NCC Glossary of Terms

AAA

AC
Application Context. A parameter in a TCAP message which indicates what protocol is conveyed. May indicate, for example, MAP, CAMEL, or INAP. Also usually specifies the particular version of the conveyed protocol, for example, which CAMEL Phase.

ACS
Advanced Control Services configuration platform.

ANI
Automatic Number Identification - Term used in the USA by long-distance carriers for CLI.

API
Application Programming Interface

BE
Billing Engine

CAMEL
Customized Applications for Mobile network Enhanced Logic
This is a 3GPP (Third Generation Partnership Project) initiative to extend traditional IN services found in fixed networks into mobile networks. The architecture is similar to that of traditional IN, in that the control functions and switching functions are remote. Unlike the fixed IN environment, in mobile networks the subscriber may roam into another PLMN (Public Land Mobile Network), consequently the controlling function must interact with a switching function in a foreign network. CAMEL specifies the agreed information flows that may be passed between these networks.

CC
Country Code. Prefix identifying the country for a numeric international address.

CCS
1) Charging Control Services (or Prepaid Charging) component.
2) Common Channel Signalling. A signalling system used in telephone networks that separates signalling information from user data.

CDR
Call Data Record

Note: The industry standard for CDR is EDR (Event Detail Record). Over time EDR will replace CDR in the Oracle documentation.
CLI
Calling Line Identification - the telephone number of the caller. Also referred to as ANI.

CORBA
Common Object Request Broker Architecture. It is a framework that provides interoperability between objects built in different programming languages, running on different physical machines perhaps on different networks. It specifies an Interface Definition Language, and API that allows client / server interaction with the ORB.

Diameter
A feature rich AAA protocol. Utilises SCTP and TCP transports.

DTMF
Dual Tone Multi-Frequency - system used by touch tone telephones where one high and one low frequency, or tone, is assigned to each touch tone button on the phone.

E2BE
Code used to designate some components and path locations used by the UBE.

EDR
Event Detail Record

Note: Previously CDR. The industry standard for CDR is EDR (Event Detail Record). Over time EDR will replace CDR in the NCC documentation.

GUI
Graphical User Interface

HLR
The Home Location Register is a database within the HPLMN (Home Public Land Mobile Network). It provides routing information for MT calls and SMS. It is also responsible for the maintenance of user subscription information. This is distributed to the relevant VLR, or SGSN (Serving GPRS Support Node) through the attach process and mobility management procedures such as Location Area and Routing Area updates.

HTML
HyperText Markup Language, a small application of SGML used on the World Wide Web. It defines a very simple class of report-style documents, with section headings, paragraphs, lists, tables, and illustrations, with a few informational and presentational items, and some hypertext and multimedia.

IN
Intelligent Network

INAP
Intelligent Network Application Part - a protocol offering real time communication between IN elements.
IOR
Inter-operable Object Reference. A reference that is used in the CORBA world that clients can use to send their requests to a particular process executing on a particular machine. Every CORBA based server has an IOR that uniquely identifies it within a distributed computing platform. IOR consists of information such as the IP address of the machine on which the process is executing, or the port number to which it is listening. This IOR is usually exported/sent to some form of central registry when the process is started up. Clients can then retrieve this information, that is, IORs, from the central registry if they want to send a request to a server.

IP
1) Internet Protocol
2) Intelligent Peripheral - This is a node in an Intelligent Network containing a Specialized Resource Function (SRF).

IP address
Internet Protocol Address - network address of a card on a computer

ISDN
Integrated Services Digital Network - set of protocols for connecting ISDN stations.

MAP
Mobile Application Part - a protocol which enables real time communication between nodes in a mobile cellular network. A typical usage of the protocol would be for the transfer of location information from the VLR to the HLR.

MSISDN
Mobile Station ISDN number. Uniquely defines the mobile station as an ISDN terminal. It consists of three parts; the country code (CC), the national destination code (NDC) and the subscriber number (SN).

Oracle
Oracle Corporation

ORB
Object Request Broker. Within an Object based communication system, an ORB keeps track of the actual addresses of all defined objects and thus is used to route traffic to the correct destination. The CORBA defines the ORB in a series of standards enabling different platforms to share common information.

OSA
Open Service Access provides a standard interface through which developers can design services that may interact with functions within the network.

PI
Provisioning Interface - used for bulk database updates/configuration instead of GUI based configuration.
PIN
Personal Identification Number

PLMN
Public Land Mobile Network

SCF
Service Control Function - this is the application of service logic to control functional entities in providing Intelligent Network services.

SCP
Service Control Point. Also known as SLC.

SCS
Service Capability Server (OSA)

SCTP
Stream Control Transmission Protocol. A transport-layer protocol analogous to the TCP or User Datagram Protocol (UDP). SCTP provides some similar services as TCP (reliable, in-sequence transport of messages with congestion control) but adds high availability.

Service Provider
See Telco.

Session
Diameter exchange relating to a particular user or subscriber access to a provided service (for example, a telephone call).

SGML

SLC
Service Logic Controller (formerly UAS).

SLEE
Service Logic Execution Environment

SMS
Depending on context, can be:
- Short Message Service
- Service Management System platform
- NCC Service Management System application
SMSC
Short Message Service Centre - stores and forwards a short message to the indicated destination subscriber number.

SN
Service Number

SQL
Structured Query Language - a database query language.

SRF
Specialized Resource Function - This is a node on an IN which can connect to both the SSP and the SLC and delivers additional special resources into the call, mostly related to voice data, for example play voice announcements or collect DTMF tones from the user. Can be present on an SSP or an Intelligent Peripheral (IP).

SSP
Service Switching Point

TCAP
Transaction Capabilities Application Part – layer in protocol stack, message protocol.

TCP
Transmission Control Protocol. This is a reliable octet streaming protocol used by the majority of applications on the Internet. It provides a connection-oriented, full-duplex, point to point service between hosts.

Telco
Telecommunications Provider. This is the company that provides the telephone service to customers.

Telecommunications Provider
See Telco.

VLR
Visitor Location Register - contains all subscriber data required for call handling and mobility management for mobile subscribers currently located in the area controlled by the VLR.

VWS
Oracle Voucher and Wallet Server (formerly UBE).
Index

A
AAA • 243
About Installation and Removal • 239
About This Document • vii
AC • 243
Account Manager Functionality • 103
Account Manager Parameters • 109
Account transaction history • 5
ACS • 243
Additional configuration of osaChamBe • 239
Additional information • 119, 120
Alarm text and variables • 49
Alarm Topic Description • 49
Alarms • 81
allowBalanceDateAfterWalletDate • 17
AllowDirectPostPaid • 17
AllowInvalidReleaseSequenceNumber • 18
allowNonCashBalanceForAmountMethods • 18, 136, 151, 167, 184, 198, 214
AlwaysSendNotification • 18
AlwaysUseAbsoluteCostValueForNotifications • 18
amount • 134, 135, 140, 152, 177, 182, 183, 186, 199
Amount Based Reservations • 88
Amount-Based Transactions • 7
ANI • 243
API • 243
AppAccountManager • 107, 109
appAccountManagerIORFile • 41
appChargingSession • 126, 127, 128, 129, 177, 182, 183, 184, 186, 191, 199, 202, 210, 211, 212, 214, 217
appInterface • 103, 110, 125, 129
Application Account Manager Functionality • 111
Application Account Manager Parameters • 122
Application Charging Manager Functionality • 131
Application Charging Manager Parameters • 131
Application Charging Session Functionality • 225
Application Charging Session Parameters • 234
applicationDescription • 134, 135, 140, 144, 152, 155, 167, 170, 177, 182, 183, 184, 186, 191, 199, 202, 214, 217
Applying bonuses • 137, 139, 141, 185, 186, 188
ArraySize • 23, 35
assignmentID • 108, 109, 110
Audience • vii
Auxiliary Scripts (SignalOsaChamScs.sh) • 240

B
Background Processes • 73
Balance/Account Info Queries • 85
balanceCascade • 41
balances • 112, 122
BalanceThreshold • 37
balanceType • 46
balanceTypes • 38
BE • 243
BE parameters • 37
beLocationPluginLib • 19
beVWARS parameters • 38
Billing Engine EDRs • 8, 10, 76
billingEngineId • 43
bucketId • 44
Business • 32
calledNumber • 44
callingNumber • 44
CAMEL • 243
cause • 119, 122
CC • 243
CCS • 243
CCS parameters • 33, 38
ccsSSMDispatcher • 75
ccsSSMMaster • 33
CDR • 243
CDRDirectory • 19
CDRMaxAge • 19
CDRMaxSize • 20
ChangeNotification() • 108
Charging management • 7, 8
Charging Manager Functionality • 125
Charging Manager Parameters • 129
Charging party fields • 7, 8
Charging Session Functionality • 133
Charging session management • 6
Charging Session OSA Plus Functionality • 181
Charging Session Parameters • 176
Charging sessions • 6
ChargingEventCriteria • 107, 108, 110, 177
chargingEventInfo • 120, 122
chargingParameters • 140, 144, 152, 155, 164, 167, 170, 178, 186, 191, 199, 202, 212, 214, 217
CLI • 244
Client Configuration • 40
clientName • 20
closeReservation • 134, 135, 178, 182, 183, 184
Communications Framework • 10
Components • 2
cfg • 27
Configuration • 13, 73, 74, 84
Configuration components • 14
Configuration Overview • 13
Configuration process overview • 13
Configuring Balance Type Currencies • 34
Configuring osaChamBe • 35
Configuring osaChamScs • 15
connectionRetryTime • 20
Copyright • ii
CORBA • 244
correlationID • 126, 127, 129
CreateChargingSession() • 126
CreateChargingSessionWithSessionID() • 127
CreateNotification() • 107
Creating new balances • 138, 139
Creation • 76
Creation of Charging Session • 86
CreditAmountErr() • 225
CreditAmountReq() • 133
CreditAmountRes() • 226
CreditAmountWithAppChargingSessionReq() • 181
creditedAmount • 226, 229, 234
creditedVolumes • 227, 229, 234
CreditUnitErr() • 226
CreditUnitReq() • 134
CreditUnitRes() • 226
CreditUnitWithAppChargingSessionReq() • 182
Critical errors • 51

D
DAP packages • 239
DebitAmountErr() • 227
DebitAmountReq() • 134
DebitAmountRes() • 227
DebitAmountWithAppChargingSessionReq() • 183
debitedAmount • 227, 230, 234
debitedVolumes • 228, 231, 235
DebitUnitErr() • 227
DebitUnitReq() • 135
DebitUnitRes() • 228
DebitUnitWithAppChargingSessionReq() • 184
Debugging • 241
defaultBaseDirectory • 41
defaultTmp • 41
Description • 103, 106, 107, 108, 109, 111, 119, 120, 125, 126, 127, 128, 131, 133, 134, 135, 143, 151, 154, 162, 163, 164, 166, 170, 181, 182, 183, 184, 190, 198, 201, 209, 210, 211, 213, 217, 225, 226, 227, 228, 229, 230, 231, 232, 233
DestroyNotification() • 109
Diagram • 11, 79, 82
Diameter • 244
Direct Credit • 92, 98
Direct Debit • 91
Direct Debit (Named Events) • 97
DirectCreditAmountErr() • 228
DirectCreditAmountReq() • 135
DirectCreditAmountRes() • 228
DirectCreditAmountWithAppChargingSessionReq() • 184
DirectDebitAmountErr() • 229
DirectDebitAmountReq() • 151
DirectDebitAmountRes() • 230
DirectDebitAmountWithAppChargingSessionReq() • 190
DirectDebitUnitErr() • 229
DirectDebitUnitReq() • 154
DirectDebitUnitRes() • 230
DirectDebitUnitWithAppChargingSessionReq() • 201
discount • 44
dispatcherOraclePassword • 33
dispatcherOracleUsername • 33
Document Conventions • viii
DTMF • 244

E
E2BE • 244
Editing the file • 15
EDR • 244
error • 226, 227, 228, 229, 230, 231, 232, 233, 234, 235
Errors • 59
eserv.config Configuration • 13, 14
eserv.config files delivered • 15
event • 45
Events • 76
Example BE section • 37
Example beVWARs section • 38
Example CCS section • 38
Example configuration file • 15, 38
Example EDRs • 8, 9, 76, 77
Example osaChamClient.conf file • 40
Execution • 75, 240
Execution - osaChamScs • 240
ExpiryDays • 23, 35
Extending balance and wallet expiry dates • 136, 137, 142, 144, 151, 155, 185, 186, 189, 190, 198, 202
ExtendLifeTimeErr() • 231
ExtendLifeTimeReq() • 162
ExtendLifeTimeRes() • 231
ExtendLifeTimeWithAppChargingSessionReq() • 209

F
Failure • 86
fifoFileName • 20
filename • 34
function • 28
Functionality • 81
Further information • 50

G
GetAmountLeft() • 162
GetAmountLeftWithAppChargingSession() • 210
getchargingSession( ) • 128
GetLifeTimeLeft() • 163
GetLifeTimeLeftWithAppChargingSession() • 210
GetNotification() • 109
GetUnitLeft() • 163
GetUnitLeftWithAppChargingSession() • 211
GUI • 244

H
heartbeatPeriod • 21
HLR • 244
hostname • 34
HTML • 244

I
IN • 244
INAP • 244
Installation and Removal Overview • 239
IntermediateCommit • 24, 36
Introduction • 1, 4, 5, 6, 10, 11, 13, 14, 15, 35, 40, 49, 78, 81, 109, 122, 129, 131, 176, 234, 239, 240
IOR • 245
IP • 245
IP address • 245
ipAccountManagerIORFile • 21, 42
ipChargingManagerIORFile • 21, 42
ISDN • 245

J
jacob.properties • 46

L
libnotificationQM • 74
library • 28
Lifetime • 9
lifeTimeHint • 45
List of statistics • 79
Loading eserv.config changes • 15
localTZ • 21
Location • 73, 74, 75

M
MAP • 245
maxChargingSession • 42
maxFileCycles • 42
MaximumAmountValue • 22
MaximumEventsValue • 22
MaximumSecondsValue • 22
MaxReports • 37
maxUsers • 43
merchantAccount • 126, 127, 129
Message Details • 85, 86, 87, 99
Message Flow • 86, 87
Message flow diagram • 5, 6
Message Flows • 85
Messages • 88, 89, 91, 92, 93, 95, 96, 97, 98
MessageTimeout • 37
minimumAmount • 167, 178, 215
msisdn • 43
MSISDN • 245
Multiple Reservations With Confirmation • 89, 94
Multiple Reservations with Confirmation - OSA Plus • 99

N
Notices • 71
Notification triggers • 5
NotificationDatabase • 23, 35
NotificationEvents • 36
notificationPollInterval • 24
NotificationQueueManager • 36
notificationRequestsEnabled • 24
Notifications • 87
NotificationServerHandler • 37
NumberOfSegments • 36
NumberOfSemaphores • 37

O
Oracle • 245
OracleUsernamePassword • 24
ORB • 245
ORBDebugLevel • 25
ORBIIOPAddr • 25
ORBIIOPPort • 25
ORBIIOPSetKeepalive • 25
OSA • 245
OSA Account Management • 5
OSA CHAM SCS Gateway • 8, 76
OSA Charging and Account Management • 4
OSA Charging Management • 6
OSA Client Interface Alarms • 51
OSA discount • 7, 9
OSA Service Handlers library • 37
osaCham parameters • 35
osaChamBe • 3
osaChamClient.conf parameters • 14, 40
osaChamScs • 3, 75
osaChamSms • 3
osaServerNotificationHandlers • 73
osaVWARSNotificationPlugins • 74
osaVWARSQueryHandlers • 74
overrideDiscountType • 26
OverrideInitialEventCdrType • 2.6.0.7 - changed to uppercase • 26
OverrideInitialTariffCdrType • 26
Overview • 1, 13, 49, 73, 83, 85, 103, 111, 125, 131, 133, 181, 225, 239

P
Parameter • 109
Parameters of osaCham • 17
Personal • 33
PI • 245
PIN • 246
PLMN • 246
plugins • 27, 31, 32, 38
port • 34
Post-paid charging • 10
Post-Paid Charging • 11
postPaidMsisdn • 29
PostPaidMsisdnInvalidRange • 28
preferredAmount • 167, 178, 215
Prerequisites • vii
Procedure - aux scripts • 241
Process • 2, 11
Purpose • 1, 73, 74, 75, 84

Q
Queries • 43
QueryBalanceErr() • 119
QueryBalanceReq() • 103
QueryBalanceRes() • 111
queryld • 112, 119, 120, 122
Querying an account • 5
queryWalletTolerance • 35
QueueSize • 36
QueueTimeout • 36

R
rangeStart • 29
rangeStop • 29
RateErr() • 231
RateReq() • 164
RateRes() • 232
rates • 232, 235
RateWithAppChargingSessionReq() • 211
Related documents • vii
Relationship diagram • 4
Release() • 166
remoteSMF1Name • 30
ReportNotification() • 120

requestNumber • 134, 135, 140, 144, 152, 155, 166, 167, 170, 179, 182, 183, 184, 187, 191, 199, 203, 215, 217, 226, 227, 228, 229, 230, 231, 232, 233, 236
requestNumberNextRequest • 226, 227, 228, 229, 230, 231, 232, 233, 236
requestTimeout • 30
Reservation Released • 91, 96
ReserveAmountErr() • 232
ReserveAmountReq() • 166
ReserveAmountRes() • 232
ReserveAmountWithAppChargingSessionReq() • 213
reservedAmount • 233, 236
reservedAmountLeft • 236
reservedUnits • 233, 236
reservedUnitsLeft • 227, 228, 237
ReserveUnitErr() • 233
ReserveUnitReq() • 170
ReserveUnitRes() • 233
ReserveUnitWithAppChargingSessionReq() • 217
retrievalId • 119, 123
RetrieveTransactionHistoryErr() • 120
RetrieveTransactionHistoryReq() • 106
RetrieveTransactionHistoryRes() • 119
Returns • 103, 105, 106, 107, 108, 109, 125, 126, 127, 128, 134, 135, 143, 150, 154, 162, 163, 164, 166, 169, 176, 182, 183, 184, 190, 197, 201, 209, 210, 211, 213, 216, 223
Running the OSA Gateway • 83
Running the Test Client Application • 84
runOsaChamScs • 75

S
SCF • 246
Scope • vii
SCP • 246
SCS • 246
SCTP • 246
SegmentSize • 36
SemaphoreKey • 36, 37
Sending Notification Short Messages • 11
sendSMSFlag • 45
Service Provider • 246
Session • 246
SessionAborted() • 131
sessionDescription • 126, 127, 129
SessionEnded() • 233
sessionId • 127, 129, 131, 134, 135, 140, 144, 152, 155, 162, 163, 164, 166, 167, 170, 179, 182, 183, 184, 191, 199, 202, 210, 211, 212, 214, 217, 226, 227, 228, 229, 230, 231, 232, 233, 234, 237
sessionTimeLeft • 231, 233, 237
SetCallBack() • 103
SetCallBack() • 125
Severity levels • 49
SGML • 246
ShmKeyFile • 36
signalOsaChamScs • 75
signalOsaChamScs.sh • 84
Single Reservation with Confirmation • 93
Single Reservations with Confirmation • 88
SLC • 246
SLEE • 246
SMS • 246
SMSC • 247
SMSLanguage • 30
SN • 247
Specifying balance and wallet expiry dates • 137, 138, 143, 186, 189, 190
SQL • 247
SRF • 247
SSP • 247
Starting up a client instance • 84
Starting up a gateway instance • 83
Statistics • 78
Stopping a gateway instance • 83
Stopping the client instance • 84
Success • 85
System Alarms • 49
System architecture • 2
System Overview • 1

T
Target location • 81
TCAP • 247
TCP • 247
Telco • 247
Telecommunications Provider • 247
throttleRate • 30
TimeoutCheckPeriod • 37
TimeUnitExponent • 31
Tools and Utilities • 83
TpAddress structure • 104, 106, 107, 108, 112
TpBalanceSet Additional Information • 114
TpChargingParameter structure • 140, 152, 164, 168, 187, 199, 212, 215
Transaction History • 86
Transaction History Retrieval • 86
transactionHistory • 119, 123
transactionHistoryError • 120, 123
transactionInterval • 106, 110
types • 46
Typographical Conventions • viii

U
Unit Based Reservations (Named Events) • 93
Unit-Based Transactions • 8
Unit-Billable event based credit request • 148, 195
Unit-Billable event based debit request • 159, 207
Unit-Seconds based credit request • 145, 192
Unit-Seconds based debit request • 156, 203
Updating existing balances • 137, 138
user • 106, 110, 126, 127, 129
users • 104, 110

V
validityTimeLeft • 232, 237
VLR • 247
volumes • 134, 135, 144, 155, 179, 183, 184, 191, 203
voucherRechargeOptions • 31
voucherRechargeTriggers • 31
voucherTypeRechargeOptions • 31
voucherTypeRechargeTriggers • 32
VWS • 247

W
walletId • 45
WalletNameTranslations • 32
walletType • 46
Warnings • 71