Oracle® Communications Billing and Revenue Management

Elastic Charging Engine 11.3 Implementation Guide

Release 7.5

E70768-05

July 2018

ORACLE

Oracle Communications Billing and Revenue Management Elastic Charging Engine 11.3 Implementation Guide, Release 7.5

E70768-05

Contents

2 Integrating ECE with PDC

3 Integrating ECE with BRM

9 Configuring Business Rules for Charging

10 Configuring Charging Runtime Options

Part IV Integrating ECE with Client Programs

11 Overview of Integrating Client Applications with ECE

12 About the ECE Sample Programs

13 Integrating Charging Clients with ECE

14 Integrating Policy Clients with ECE

15 Integrating Top-Up Clients with ECE

16 Integrating Query Clients with ECE

Part V Testing an ECE Implementation

17 Overview of ECE Testing Tools

18 Using the Simulator to Test ECE

19 Using the Query Tool to Test ECE

20 Testing Scenarios in an Integrated System

Part VI Implementation Utilities

21 ECE Implementation Utilities

Part VII Appendices for Implementation Guide

A ECE API Reference

Preface

This guide describes the procedures for implementing Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE) in your charging system, including configuring usage-charging business rules and configuring connections to other applications in the charging system.

Audience

This guide is for system administrators, database administrators, and developers who plan a charging system that includes ECE, who integrate ECE with applications in the charging system, and who implement ECE charging functionality in the manner required for their business needs.

This guide assumes that users have a working knowledge of Linux, Solaris, Oracle Coherence, and Oracle NoSQL Database, and are familiar with the following topics:

- Operating system commands
- Database configuration
- Network management
- Data store configuration in an Oracle NoSQL system
- Product offering creation in Oracle Communications Pricing Design Center
- Customer account creation in Oracle Communications Billing and Revenue Management (BRM)

This guide assumes that users have installed ECE, either as an ECE standalone installation (to become familiar with how ECE works), or as an ECE integrated installation. See *BRM Elastic Charging Engine Installation Guide*.

This guide assumes that users have become familiar with ECE concepts. See *BRM Elastic Charging Engine Concepts*.

Directory Placeholders Used in This Guide

The following placeholders are used in this guide to refer to the directories that contain ECE system components:

Placeholder	Definition
ECE_home	The directory in which ECE server software is installed. This directory contains the ECE Server directory (**/oceceserver**) and the ECE SDK directory (**/ocecesdk**) and various installation-related files.
PDC_home	The directory in which PDC software is installed.

Accessing Oracle Communications Documentation

ECE documentation and additional Oracle documentation; such as Oracle Database documentation, is available from Oracle Help Center:

- http://docs.oracle.com

Additional Oracle Communications documentation is available from the Oracle software delivery Web site:

- https://edelivery.oracle.com

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Document Revision History

The following table lists the revision history for this book.

Version	Date	Description
E70768-01	April 2016	Initial release.
E70768-02	September 2016	Documentation updates for ECE 11.3 Patch Set 1. ■ Added the following sections: Configuring Headers for External Notifications Creating the BRM Gateway Suspense Queue Custom Plug-in API ■ Updated the following information: Installing and Configuring Rated Event Formatter Sample Notification Payloads All procedures involving MBeans Integrating ECE with BRM

Version	Date	Description
E70768-04	December 2016	Documentation updates for ECE 11.3 Patch Set 2. ■ Added the following sections: Configuring ECE to Override a Product Price Loading Customer Data Incrementally with customerLoader ■ Updated the following sections: About Rounding Charging Results Configuring Item Assignment in ECE for BRM About customerLoader Using customerLoader
E70768-05	July 2018	Documentation updates for ECE 11.3 Patch Set 8. ■ Updated the following sections: Configuring BRM Gateway Configuring External Manager Gateway for High Availability Configuring Threshold Breach Notifications

Part I

Implementation Concepts

Part I describes concepts for implementing Oracle Communications Billing and Revenue Management (BRM) Elastic Charging Engine (ECE). It contains the following chapter:

- Implementation Concepts

1

Implementation Concepts

This chapter explains concepts critical for implementing Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE).

Before reading this chapter, you should be familiar with ECE concepts and architecture. See *BRM Elastic Charging Engine Concepts* for information about how ECE works as a rating and balance management application.

About Implementing ECE in a Charging System

To implement ECE in a charging system, you configure integration points to other applications in the charging system so that ECE can send data to and receive data from those applications. For information about configuring integration points for an Oracle Communications Billing and Revenue Management (BRM) charging system, see "About Integration Points".

For ECE to rate events, the definition of those events must be stored in ECE. You define each of your events by creating or editing an event specification in BRM. You then export the event specification to Oracle Communications Pricing Design Center (PDC), where information required for network mediation processing, pricing, charging, and billing is added. The event specification in PDC is then published through Pricing Updater to ECE, where it is stored in the ECE event cache. See "Integrating ECE with PDC" for information about how event specifications are enriched in PDC for ECE.

You can configure various business rules for charging and runtime behavior of the ECE system. To implement business rules and charging-related configurations in ECE, see "Implementing Charging Configurations".

ECE APIs align with standards in the charging industry to facilitate integrating ECE with client programs and network charging functions. For information about integrating ECE with client programs, see "Integrating ECE with Client Programs".

For information about testing an ECE implementation, see "Testing an ECE Implementation".

About Implementation Tasks

After installing ECE, you need to integrate it with your charging system. A first-time implementation of ECE in a BRM charging system requires that you perform tasks in a particular order.

ECE implementation tasks include the following:

- Defining your events in BRM and PDC and storing those definitions in ECE so ECE can rate the events

  You define each of your events by creating or editing an event storable class definition in BRM. You then export the definition as an event specification to PDC, where information required for processing usage requests is added. The event specification in PDC is then published through Pricing Updater to ECE, where it is stored in the ECE event cache.

- Defining and loading your pricing so that events can be rated

  A pricing analyst typically defines pricing for events when pricing and services are set up in BRM. Pricing often determines which events you define in BRM. You define pricing by creating pricing components in PDC. You use those pricing components to create product offerings. You then load this pricing data into ECE. See "About Loading Data from PDC".

- Configuring integration points so that ECE can connect to other applications in the charging system

  For example, you configure integration points for connecting to and communicating with BRM so that ECE can send rated events to BRM. You also configure integration points for connecting to PDC so that ECE can receive pricing data from PDC. See "About Integration Points".

- Loading required data into ECE caches so that ECE has the data required for charge processing

  You must load your mediation specification (if you use Diameter Gateway for network integration for online charging), your pricing data, and your customer data. See "About Loading Data from BRM" and "About Loading Data from PDC".

- Configuring notifications so that ECE charging servers can send data to client applications such as network mediation software programs

  See "Configuring Notifications for Charging".

- Configuring usage-charging preferences so that ECE charging servers apply the charging business logic required for your business

  See "Configuring Business Rules for Charging".

About Defining Chargeable Usage Events

When implementing ECE in a charging system, you have already defined the attributes from your network requests that are associated with each event you will charge for. You define event attributes by creating or editing an event storable class definition in BRM. You then export the event storable class definition as an event specification to PDC, where information ECE requires for processing usage requests is added. The event specification in PDC is then published through Pricing Updater to ECE, where it is stored in the ECE cache.

About Usage Requests

ECE must contain definitions of the events for which you charge your customers so that it can rate the events when processing usage requests. ECE stores request specification data objects, which contain the definition of the events defined in BRM and PDC.

Client applications, such as network mediation software programs, create usage requests and submit them to ECE for charging. The network mediation software

program maps the usage attributes relevant to charging from the network request to the usage attributes in the ECE usage request.

ECE usage requests contain fixed attributes that apply for all of the events in ECE, such as the attribute that represents the public user identity of the person or entity using the product. Fixed attributes are mandatory in all usage requests. ECE usage requests also contain configurable attributes that represent the usage of the customer.

When you introduce products or extend ways of charging customers for your products, you extend usage requests by adding the products, events, or attributes required for the new type of charging. See the discussion about enriching event definitions in *PDC User's Guide*.

For information about building usage requests (useful if you use third-party network mediation software), see "Integrating Charging Clients with ECE".

About Request Specification Data Objects

Request specification data objects must be deployed into the ECE runtime environment before charging applications can send usage requests to ECE. See "Integrating Charging Clients with ECE".

For an ECE standalone system, you can use sample request specification files in the *ECE_home*/**oceceserver/sample_data/config_data/specifications** directory to load request specification data objects into the ECE cluster.

About Integration Points

This section describes integration points ECE uses for communicating with software products in a BRM charging system.

About Integration Points for PDC

ECE uses the following integration points for PDC, which are set up in the PDC system.

- Java Message Service (JMS) queue

  Pricing Updater listens on the JMS queue to which PDC publishes pricing data; it dequeues pricing data for loading into ECE.

See "Integrating ECE with PDC" for instructions on integrating ECE with PDC.

About Integration Points for BRM

ECE uses the following integration points for BRM, which are set up in the BRM system:

- Connection Manager (CM)

  BRM Gateway connects to the CM to make BRM opcode calls.

  The CM calls opcodes implemented in an external manager (EM). External Manager (EM) Gateway receives requests from the CM at a predefined port. EM Gateway is used, for example, for ECE to receive charging requests from BRM during rerating operations.

- Rated Event (RE) Loader

  The BrmCdrPlugDirect Plug-in creates RE Loader files (rated event files). RE Loader loads rated events from the files into the BRM database.

- Account Synchronization Data Manager (DM) Oracle Advanced Queuing (Oracle AQ) database queue

  When events occur that update rerating, account migration, discount, and product data in the BRM database, the updates are published asynchronously (not in real time) to ECE through Customer Updater. To do so, the updates are sent in business events to the Account Synchronization DM, which publishes the business events to an Oracle AQ database queue. Customer Updater retrieves the business events from the queue and updates the ECE cache accordingly.

  For information about synchronizing *customer* data updates, see the discussion about synchronizing BRM and ECE customer data in *BRM Elastic Charging Engine Concepts*.

- Suspense queue (Oracle AQ database queue)

  The suspense queue stores failed update requests that BRM tried to send to ECE until they can be reprocessed.

  For information about processing failed update requests from BRM, see the discussion on troubleshooting ECE in *BRM Elastic Charging Engine System Administrator's Guide*.

  For information about configuring the suspense queue, see "Configuring the Suspense Queue".

- ECE Acknowledgment Queue (Oracle AQ database queue)

  BRM uses the Acknowledgment Queue for sending messages that the rerating process is set to start or has completed. ECE uses the Acknowledgment Queue for sending data to BRM about the state of processing rerating jobs.

See "Integrating ECE with BRM" for instructions on integrating ECE with BRM.

About Integration Points for Network Mediation Software for Online Charging

ECE uses the following integration points for network mediation software for online charging. Network mediation software would leverage these open interfaces to integrate with ECE.

> **Note:** This information is relevant to those integrating a third party network mediation system with ECE for online charging.
>
> If you use Diameter Gateway for receiving Diameter messages and translating them into ECE requests, Diameter Gateway is prebuilt to use these interfaces for constructing all of the requests that ECE supports.

- Elastic Charging Client

  Elastic Charging Server accepts usage requests and policy requests submitted from the Elastic Charging Client.

- ECE request specification data

  Request builders use the request specification data (generated automatically and deployed to ECE from PDC) for building usage requests in the correct format.

- ECE charging API

The network mediation software uses the ECE charging API to populate the payload blocks of usage requests.

- ECE policy API

  The network mediation software uses the ECE policy API to build and submit policy requests.

See "Integrating Charging Clients with ECE" and "Integrating Policy Clients with ECE" for related information.

About Integration Points for Offline Mediation Controller

ECE uses the following integration points for Oracle Communications Offline Mediation Controller, which are set up on the Offline Mediation Controller system:

- Elastic Charging Client

  Elastic Charging Server accepts charging requests submitted from the Elastic Charging Client.

- ECE request specification data

  Request builders use the request specification data deployed in ECE for building requests in the correct format.

- ECE charging API

  Offline Mediation Controller uses the ECE charging API to populate the payload blocks of usage requests.

See "Integrating ECE with Offline Mediation Controller" for instructions on integrating ECE with Offline Mediation Controller.

See "Integrating Charging Clients with ECE" for related information.

About Implementation Tools

Implementation tools that help you configure and test an ECE implementation include the following:

- Java Management Extensions (JMX) editor

  A JMX editor enables you to edit ECE MBeans made accessible by the ECE configuration service. You use a JMX editor to configure most ECE nodes. After ECE is running, your edits to configuration parameters impact the running system.

  See "About Accessing and Editing ECE MBean Parameters".

- Simulator

  The simulator emulates the role of a client application, such as a network mediation software program, sending requests to ECE.

  See "Using the Simulator to Test ECE".

- **loader** utility

  The **loader** utility, used with the simulator, loads sample data into ECE caches required for processing requests and loads sample request specification files used to validate requests sent by the simulator.

  See "Using the Simulator to Test ECE".

- Customer File Generator

  Customer File Generator generates customer files according to provided specifications; it allows you to create customer data without having to create it in BRM.

  See "About the Customer File Generator".

- Sample programs

  Sample programs demonstrate how to use the ECE API for sending requests to ECE.

  See "About the ECE Sample Programs".

- Query tool

  The query tool enables you to execute queries on ECE Coherence caches for development or debugging purposes.

  See "Using the Query Tool to Test ECE".

- Data-loading utilities enable you to prime ECE caches with required data and reload data as needed in testing environments.
  - **configLoader** utility
  - **pricingLoader** utility (used only when PDC is not used)
  - **customerLoader** utility (used only when BRM is not used)

  See "About Data-Loading Utilities".

About Accessing and Editing ECE MBean Parameters

Each time you start an ECE charging server node with JMX management enabled, the ECE configuration service makes ECE configuration parameters accessible by exposing them as MBeans.

When you are logged on to the JMX-management-enabled charging server node using the valid host name and JMX port (the default port is **9999**), the JMX MBean editor displays the ECE MBeans. You can expand the MBeans navigation tree to view and modify the ECE configuration parameters associated with them.

The ECE system components obtain configuration settings from an application-configuration replicated cache in the ECE charging nodes. When you modify an ECE configuration parameter in the JMX MBean editor, the ECE configuration service validates the modified value in its source XML file, updates that file, and updates the parameter in the application-configuration replicated cache, thus making the new values available to all nodes in the ECE charging system.

About Loading Data from BRM

This section describes the data that ECE needs from BRM for processing usage requests.

About Customer Data

Customer data is information about customers, such as customer account details, account status, and product details.

When you install ECE, its caches are empty, and you must load customer data from BRM into ECE. To do so, you use Customer Updater, which extracts the customer data

and product cross-reference data that ECE requires from the BRM database and loads it into the ECE caches. See "Loading Data from BRM into ECE" for more information.

You load only the customer data relevant for processing usage requests. For example, you load the customer data that ECE uses to charge for network events, such as what product the customer has purchased.

After the initial load of data, Customer Updater automatically extracts all new or modified customer data from BRM and loads it into ECE. See "About Customer Updater" for more information.

About Configuration Data

The initial load of customer data into ECE includes customer offer profiles and systemwide credit profiles, which are created and managed in BRM.

Credit profiles define the credit limits, credit floors, and credit threshold combinations that can be associated with balance elements. When you create systemwide credit profiles in BRM, you must include the default systemwide credit profiles for currency and noncurrency that ECE will use. See "Defining Systemwide Credit Profiles" for more information.

Configuration data also includes the following:

- Request specification data used to define events. The **pricingUpdater** utility loads this data into ECE (see "About Pricing Data").
- Mediation specification data required by Diameter Gateway. The **configLoader** utility loads mediation specification data into ECE (see "configLoader").

About Product Cross-Reference Data

The initial load of BRM customer data into ECE includes product cross-reference data.

Product cross-reference data is stored in ECE and enables ECE to map the BRM product offering identifier to the product offering migrated from PDC. The product offering is loaded into ECE from PDC. ECE maps the product offering to a BRM charge offer, discount offer, or chargeshare offer with the BRM Portal object ID (POID) in the offering. The product cross-reference data is initially loaded into ECE by Customer Updater. Subsequent new or modified offerings are updated in ECE through EM Gateway (see the discussion about synchronizing BRM and ECE customer data in *BRM Elastic Charging Engine Concepts*).

About Customer Updater

Customer Updater does the following:

- Performs the initial extraction of customer data, credit profiles, offer profiles, configuration objects, and pricing cross-reference data from BRM and loads the data into ECE.
- Handles asynchronous updates from BRM to ECE of the following data: rerating, account migration, discount, and product.

The rerating, account migration, discount, and product data stored in ECE must stay current with the corresponding data in the BRM database.

When events occur that update that data in the BRM database, the updates can be published asynchronously (not in real time) to ECE through Customer Updater. To do so, the updates are sent in business events to Account Synchronization DM, which

publishes the business events to an Oracle AQ database queue. Customer Updater retrieves the business events from the queue and updates the ECE cache accordingly.

When you install ECE, configure Customer Updater to dequeue update requests sent from BRM. For example, you provide the name of the Oracle Advanced Queuing (AQ) database queue to which BRM publishes business events. To change the information you provide during installation, see "Configuring Customer Updater".

If ECE cannot process update requests from BRM, ECE puts the update requests in the suspense queue to process later. See "Configuring the Suspense Queue".

About Preselecting Customer Data for Loading

Customer Updater extracts and loads all customer data into ECE by default. However, you can configure Customer Updater to load only preselected customer data. To do so, prepopulate the Customer Updater distribution tables with customers whose data you want to load. You specify filter criteria that determines which customers to include; for example, you can filter based on whether customers are in a closed or preprovisioned state. Customer Updater extracts and loads only data for the selected customers. See "Creating Prepopulated Distribution Tables".

About Loading Data from PDC

This section describes the data that ECE requires from PDC for processing usage requests.

About Pricing Data

After you install ECE, the pricing cache contains no data. You must load the pricing data relevant to processing usage requests into ECE.

You define your pricing data (when you create your product offerings) in PDC. See "Creating Pricing Data for the ECE Runtime Environment".

Pricing Updater loads the pricing data into ECE. See "About Pricing Updater".

Important: The **pricingLoader** utility is an ECE utility that loads sample pricing data into the ECE pricing cache from XML files.

Do *not* run **pricingLoader** and Pricing Updater on the same ECE deployment.

Use **pricingLoader** only on a standalone system when PDC is not used.

For information about loading pricing data, see "Loading Pricing Data into ECE".

About Pricing Updater

Pricing Updater reads pricing updates from the JMS queue, where PDC publishes pricing data, and loads it into ECE.

Whenever you create or modify pricing data in PDC and publish it, Pricing Updater automatically loads the pricing data updates into ECE. After you install ECE, you start Pricing Updater. As long as it is running, it loads pricing data updates from PDC.

For information about configuring Pricing Updater, see "Configuring the Pricing Updater".

About Loading Sample Data

After installing an ECE standalone system, you can load sample data. Sample data is in the *ECE_home*/**oceceserver/sample_data** directory, which includes the following:

- Sample customer data
- Sample pricing data
- Sample data for integrating with clients that send policy requests (used for policy testing)

Sample data includes sample ECE request specification files, sample configuration data, sample product offering cross-reference data, and sample customer data. Subsets of sample data geared for ECE implementations for policy-related charging is also available.

To use sample data, you configure your data-loading utilities to load data from sample data directories. See "Using Data-Loading Utilities" for more information.

Part II

Implementing ECE in a BRM Charging Solution

Part II describes how to implement Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE) in the BRM system, including integrating with the Pricing Design Center (PDC) and network mediation client applications. It contains the following chapters:

2

Integrating ECE with PDC

This chapter provides instructions for integrating Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE) with Pricing Design Center (PDC).

Before reading this chapter, you should be familiar with ECE concepts and architecture. See the following chapters in *BRM Elastic Charging Engine Concepts*:

- ECE Overview
- ECE System Architecture

Overview of Integrating ECE with PDC

ECE must have access to pricing data to charge the usage requests it receives. ECE uses pricing data that you define in PDC. Pricing data in PDC is referred to as *pricing components*.

The following steps summarize what you need to do to integrate ECE with PDC:

Important: Before integrating ECE with PDC, the following is assumed:

- PDC is installed and it is configured for publishing pricing data for the ECE target charging engine.
- The JMS queue has been set up on the Oracle WebLogic Server domain on which PDC is deployed and PDC is configured to publish pricing data to it.
- ECE is installed and the Pricing Updater is configured and running.

---

1. Define your events by creating or extending your event storable class definitions.

   You define each of your events by creating or editing an event storable class definition in BRM. You then export the event storable class definition as an event specification to PDC where additional information is added that is required for processing usage requests. The event specification in PDC is then published to ECE through the Pricing Updater where it is stored in the ECE event cache.

   See the discussion about creating custom fields and storable classes in *BRM Developer's Guide*.

2. Export the event storable class definition as an event specification to PDC and add additional information that is required for processing usage requests.

See the discussion about enriching event definitions in *PDC User's Guide*.

3. Define your pricing in PDC and create your product offerings.

   Defining your pricing involves defining the setup components and pricing components that make up your product offerings. ECE requires this pricing data in its runtime environment.

   See "Creating Pricing Data for the ECE Runtime Environment" for information about pricing data used in the ECE runtime environment.

   See *PDC Help* for instructions on creating product offerings.

4. After you create product offerings in PDC using your event and product definitions, synchronize product offerings created in PDC with BRM.

   This task is performed in PDC and must be done before you create customers so that the customer data, which you will load into ECE from BRM later during implementation, has the information ECE requires for charge processing.

   See *Oracle Communications Pricing Design Center* documentation for information about synchronizing pricing-component data in PDC with BRM.

5. Load your pricing data.

   When you installed ECE, you specified the PDC JMS queue to which PDC publishes pricing data. Provided this configuration is set up correctly, each time PDC publishes pricing data to the JMS queue, it will be loaded into ECE. See "Configuring ECE for Receiving Pricing Data Updates from PDC".

Creating Pricing Data for the ECE Runtime Environment

You create pricing data for loading into the ECE runtime environment in PDC. ECE uses the account-level and product-level charge offer, alteration offer (discount offer) and distribution offer (charge sharing offer) you design in PDC to determine the cost of an event (which includes the effect of a discount) when processing usage requests. See "Creating Pricing Data for the ECE Runtime Environment".

You design *pricing components* in PDC such as charge offers and discount offers and design *setup components* in PDC that support the creation of pricing components such as impact categories, time models, ratable usage metrics (RUMs), service-event maps, and zone models. ECE uses the criteria in your pricing components to calculate the price of your services. See *PDC User's Guide* for a description of pricing components.

Creating pricing data in PDC is about creating product offerings. Product offerings are made up of various pricing components. Some pricing components are created in the PDC UI while other pricing components are created in an XML file and imported into the PDC database by using the **ImportExportPricing** utility. All pricing components are saved or imported into the PDC database.

Some pricing components, which you can create in the PDC UI, are not for the ECE runtime environment, but are rather subscription related and used by BRM. Components such as bundles, packages, and package lists are sent to BRM (the subscription system).

Table 2–1 summarizes the pricing data that ECE can process in its runtime environment and how you create the data.

Table 2–1 Pricing Data Used in the ECE Runtime Environment

Pricing Data	Description
Setup Components:	-
Products	Define the product (service) in BRM and synchronize it with PDC. In PDC, declare the name of the product type in the event specification to which the product applies. You define the **ProductType** name.
Events	For events, you do the following in PDC: ■ Declare the name of ECE event types in the event definition. You define the **EventType** name. ■ Define the payload blocks for the event. Configure the payload blocks of the event definition. The payload and other attributes in the event definition makes up the event specification. The event definitions and request specification data associated with the event definitions are published to ECE by Pricing Updater.
Ratable Usage Metrics (RUMs)	Use the PDC UI to define RUMs.
Service-event maps	Use the PDC UI to define service-event maps.
Balance elements	Use **ImportExportPricing** utility to import balance elements.
Impact categories	Use the PDC UI to define impact categories.
Zone models	Use the PDC UI to define zone models.
Special-day calendar	Use the PDC UI to define a special-day calendar.
Custom analyzer rules	Create an XML file in which you define your custom analyzer rules. Sample XML files are available in the *PDC_Home*/**apps/samples/examples** directory. Use the sample XML data file as a template for creating your custom analyzer rule XML data file. Use the PDC **ImportExportPricing** utility to retrieve the data from your XML file and load it into the PDC database.
Profile attribute specifications	Create an XML file in which you define your profile attribute specifications. You can find the XSD file for profile attribute specifications in its respective subdirectory in *PDC_home*/**apps/xsd**, where *PDC_home* is the directory in which the PDC software is installed. Use the PDC **ImportExportPricing** utility to retrieve the data from your XML file and load it into the PDC database.
Tax codes and general ledger IDs (G/L IDs)	Define tax codes and G/L IDs in BRM. Automatically sync the tax codes and G/L IDs from BRM to PDC by using the **SyncPDC** utility. Define ECE tax codes in ECE. ECE tax codes must match the tax codes defined in BRM. See the discussion on configuring taxation in *ECE Implementation Guide* for information about how tax codes are configured in ECE.
Pricing Components:	-
Charges	Use the PDC UI to create charges.

Table 2–1 (Cont.) Pricing Data Used in the ECE Runtime Environment

Pricing Data	Description
Discounts (Alterations)	Use the PDC UI to create discounts (alterations).
Chargeshares	Use the PDC UI to create chargeshares
Charge offers	Use the PDC UI to create charge offers.
Discount offers	Use the PDC UI to create discount offers.
Chargeshare offers	Use the PDC UI to create chargeshare offers.
Time models	Use the PDC UI to define time models.
Generic selectors	Use the PDC UI to define generic attribute selectors.

Applying Custom Attribute Analyzer Rules to Usage Request Attributes

ECE can use custom analyzer rules when processing usage requests. A custom analyzer rule is a pricing setup component that you define in PDC. Fixed usage-request attributes, which are attributes applicable for all the events in ECE (such as USER_IDENTITY, REQUEST_START, and REQUEST_END), are typically used in custom analyzer rules with the prefix **"ANYEVENT.***Fixed_Attribute***"**.

In addition to these fixed attributes, custom analyzer rules can be applied to any dynamic attribute on the payload.

If you want to use the CALLED_ID dynamic attribute in custom analyzer rules with the **"ANYEVENT"** prefix (for example, **ANYEVENT.CALLED_ID**), you must define the CALLED_ID attribute in all payloads.

Configuring ECE to Override a Product Price

ECE uses the default value of the pricing attributes to determine the product price when processing a usage request. However, you can override the price specified in the product offering at run time. To override the price, create a pricing XML file with dynamic tags and import the file into the PDC database by using the **ImportExportPricing** utility. PDC provides a sample XML file located in the *PDC_home*/**apps/Samples/Examples** directory.

> **Note:** Dynamic tags are the XML elements that are used for overriding the value of the pricing attributes, such as price, incrementStep, lowerBound, and UpperBound. Dynamic tags can be hierarchical.

To enable ECE to override the default value of the pricing attributes, you can implement a custom logic using the pre-rating extension. When you implement custom logic, the overridden values are populated in the payload of the request specification file. ECE uses these the overridden values to determine the price when processing process usage requests. ECE processes the tag hierarchy from left to right to determine the overridden value of the pricing attributes. ECE uses the default value of the attributes to determine the price when no custom logic is implemented.

For more information on the pre-rating extension, see *BRM Elastic Charging Engine Extensions*.

Loading Pricing Data into ECE

Caution: In a production environment, PDC is the master of pricing data. Do not load pricing data into ECE using the **pricingLoader** utility or the simulator **loader** utility when you use PDC. Ready-to-use sample pricing data that is loaded by these utilities can conflict with pricing data created using PDC.

To load pricing data into ECE you start the ECE Pricing Updater node and publish your pricing data from PDC. In an ECE integrated installation, Pricing Updater loads pricing updates from PDC.

To load pricing data from PDC:

1. Verify the Pricing Updater is configured.

   See "Configuring the Pricing Updater".

2. On the machine on which you have Elastic Charging Controller (ECC) installed, go to *ECE_Home***/oceceserver/bin**.

3. Start the Elastic Charging Controller:

   ```
   ./ecc
   ```

4. Run the following commands in this order:

   ```
   > start
   > start configLoader
   > start pricingUpdater
   ```

   After the ECE Pricing Updater node is running, each time you alter pricing data in PDC (such as create or modify a charge offer), PDC writes the data to the PDC JMS queue. The Pricing Updater listens on the queue and automatically loads the pricing data into ECE.

Configuring ECE for Receiving Pricing Data Updates from PDC

When you installed ECE, you provided information for configuring the Pricing Updater. The Pricing Updater reads pricing updates from the JMS queue where PDC publishes pricing data and loads it into ECE. You can change this configuration if needed. See "Configuring the Pricing Updater".

Configuring the Pricing Updater

To configure the Pricing Updater:

1. Open the *ECE_Home***/config/management/migration-configuration.xml** file.

2. In the **PdcEceQueue** section, specify the JMS queue details of the queue where the Pricing Updater dequeues the pricing-data work items from PDC.

3. In the **PDCResultQueue** section, specify the JMS result queue details of the queue where the Pricing Updater publishes the results back to PDC.

4. (Optional) If you need to change the password value for connecting to PDC computers, run the encrypt password utility script to obtain the password, and then enter the password value for the **Password** parameter.

You may need to change the password, for example, if ECE needs to interact with new PDC computers, or if the password used to connect to existing PDC computers must change.

For information about running the encrypt password utility script, see the discussion on encrypting new passwords in *BRM Elastic Charging Engine System Administrator's Guide*.

5. Save the file.

3

Integrating ECE with BRM

This chapter explains how to integrate Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE) with Oracle Communications Billing and Revenue Management (BRM).

Before reading this chapter, you should be familiar with ECE concepts and architecture. See the following chapters in *BRM Elastic Charging Engine Concepts*:

- ECE Overview
- ECE System Architecture

Overview of Integrating ECE with BRM

The following steps summarize what you must do to integrate ECE with BRM:

> **Important:** Before integrating ECE with BRM, you must load ECE event and product definitions into Pricing Design Center (PDC), define product offerings in PDC, and synchronize those product offerings with BRM. See "Integrating ECE with PDC".

1. (Optional) Reconfigure External Manager (EM) Gateway for your system.

   See "Reconfiguring External Manager Gateway".

2. Load BRM configuration files into BRM.

   See "Loading BRM Configuration Files into BRM".

3. Configure ECE to receive customer data from BRM.

   See "Configuring ECE to Receive Customer Data from BRM".

4. Create customer accounts in BRM and associate them with your product offerings.

   See the BRM documentation for information about creating customer accounts.

5. Load customer data from BRM into ECE.

   See "Loading Data from BRM into ECE".

6. Configure ECE for synchronizing customer data with BRM.

   See "Configuring ECE for Synchronizing Customer Data with BRM".

7. Configure ECE for sending rated events to BRM.

   See "Configuring ECE to Send Rated Events to BRM".

8. (Optional) Configure ECE for rerating.

See "Configuring ECE for Rerating".

9. Enable ECE to rate events during account migration.

   See "Enabling ECE to Rate Events during Account Migration".

10. (Optional) Configure EM Gateway for high availability.

    See "Configuring External Manager Gateway for High Availability".

11. Start BRM Gateway by running the following command:

    ```
    start brmGateway
    ```

12. (Optional) To integrate ECE with BRM multischema environments, see "Configuring ECE for a Multischema BRM Environment".

Reconfiguring External Manager Gateway

External Manager (EM) Gateway is a mandatory ECE component used for the following tasks:

- Synchronizing customer data between BRM and ECE in real time
- Processing rerating requests
- Fetching real-time balances from ECE

When you installed ECE, you provided information for configuring EM Gateway. To change the information you provided during installation, follow these instructions.

To reconfigure EM Gateway:

1. Access the ECE MBeans:
   a. Log on to the driver machine, which is the machine on which you installed ECE.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file, where *ECE_home* is the directory in which ECE is installed.

      The **eceTopology.conf** file also contains the host name and port number for the node.

   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.emGatewayConfigurations.***Instance_Name*, where *Instance_Name* is the name of the instance you want to configure.
3. Expand **Attributes**.
4. Specify values for the following attributes:
   - **threadPoolSize**: Enter the number of EM Gateway threads to start each time EM Gateway starts.
   - **port**: Specify the port number of the machine on which EM Gateway resides.
   - **name**: Enter the host name of the machine on which EM Gateway resides.
5. Open the *ECE_home*/**oceceserver/config/eceTopology.conf** file.
6. Verify that an entry exists for EM Gateway with the correct settings.

For information about configuring settings in the ECE topology file, see the discussion about configuring the ECE system in *BRM Elastic Charging Engine System Administrator's Guide*.

After EM Gateway is configured, you can start it. Alternatively, you can start it immediately before you perform rerating.

For information about starting EM Gateway, see the discussion about starting and stopping ECE in *BRM Elastic Charging Engine System Administrator's Guide*.

Loading BRM Configuration Files into BRM

The ECE installation includes the following BRM configuration files, which contain specific configurations for ECE. Copy these files to your BRM installation, and configure BRM to use them.

Important: Before loading files that may overwrite existing files, make backups of the files on your BRM system.

- *ECE_home***/oceceserver/brm_config/payloadconfig_ece_sync.xml**. See the discussion about loading the ECE configuration files into your BRM environment in *BRM Setting Up Pricing and Rating*.
- *ECE_home***/oceceserver/brm_config/pin_notify**: See the discussion about using event notification in *BRM Developer's Guide* and about using the **load_pin_notify** utility in *BRM Managing Customers*.

Configuring ECE to Receive Customer Data from BRM

To configure ECE to receive the initial load of customer data from BRM and then to receive customer data updates, configure Customer Updater. To configure ECE to move failed update requests from Customer Updater, configure the suspense queue. See the following for more information:

- Configuring Customer Updater
- Configuring the Suspense Queue

Configuring Customer Updater

To configure Customer Updater:

1. Point the BRM Payload Generator External Module (EM) configuration file to the ECE payload configuration file (**payloadconfig_ecc_sync.xml**).

   See "Loading BRM Configuration Files into BRM".

2. (Optional) To load data for only specified customers, enable Customer Updater to use prepopulated distribution tables as follows:

   a. Ensure that you have created prepopulated distribution tables. To create prepopulated distribution tables, see "Creating Prepopulated Distribution Tables".

   b. Open the *ECE_home***/oceceserver/config/defaultTuningProfile.properties** file.

      This is the default Java virtual machine (JVM) tuning file. For information about custom JVM tuning files, see the discussion about configuring JVM parameters in *BRM Elastic Charging Engine System Administrator's Guide*.

c. Add the following Java property to the file:

```
-DpreDistributedWorkItems=true
```

d. Save and close the file.

3. (Optional, Testing Only) Enable Customer Updater to continue extracting data even if validation fails as follows:

a. Open the *ECE_home*/**oceceserver/config/defaultTuningProfile.properties** file.

This is the default Java virtual machine (JVM) tuning file. For information about custom JVM tuning files, see the discussion about configuring JVM parameters in *BRM Elastic Charging Engine System Administrator's Guide*.

b. Add the following Java property to the file:

```
-DcontinueCustomerLoaderOnError=true
```

Note: Set this property to **true** only in a testing environment. In a production environment, all customer data is loaded into ECE, so set this property to **false**.

c. Save and close the file.

4. Open the *ECE_home*/**config/management/migration-configuration.xml** file.

5. Set the **remoteWmThreads** parameter to the number of parallel work manager threads to be used for Customer Updater.

By default, this parameter is set to **1**.

6. Set the **batchSize** parameter to the number of customers to put into the repository in one put operation.

Use **batchSize** to optimize performance of put operations into the repository. The more customers you insert into the repository in one put operation (rather than inserting each one individually), the better the performance.

By default, this parameter is set to **5000**.

7. Set the **dbConnections** parameter to the number of parallel database connections to use for initial extraction and load of customer data.

By default, this parameter is set to **1**.

8. Set the **dbFetchSize** parameter to the number of records that Customer Updater should extract from the BRM database at one time.

By default, this parameter is set to **5000**.

9. Save and close the file.

10. Access the ECE MBeans:

a. Log on to the driver machine.

b. Start the ECE charging servers (if they are not started).

c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.

d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

The **eceTopology.conf** file also contains the host name and port number for the node.

e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.

11. Expand **charging.connectionConfigurations.oracleQueueConnection1**.
12. Expand **Attributes**.
13. Specify values for the following attributes:
    - **password**: Enter the encrypted password for logging on to the computer on which the database queue resides.

      When you install ECE, the password you enter is encrypted and stored in the keystore.
    - **queueName**: Enter the name of the database queue that holds the published business events from BRM.
    - **jdbcUrl**: Enter the Oracle JDBC URL to use to connect to the BRM database:

      ```
      jdbcUrl="jdbc:oracle:thin@//hostname:port:sid"
      ```

      where *hostname* and *port* are the host name and port number of the computer on which the database queue resides, and *sid* is the name of the BRM database service.

Configuring Customer Updater for a BRM Multischema Environment

In a BRM multischema environment, each schema publishes business events to its own Oracle Advanced Queuing (Oracle AQ) database queue for ECE to consume. Configure one Customer Updater process for each Oracle AQ database queue. Each Customer Updater process loads the initial BRM customer data before listening for customer updates.

To configure multiple Customer Updater processes:

1. Open the *ECE_home***/oceceserver/config/eceTopology.conf** file.
2. Add a row to the file for each Customer Updater process.

   Use a unique name for each process so that Elastic Charging Controller (ECC) can distinguish between them. (All Customer Updater processes use the same role.)

   For example, to configure three Customer Updater processes, add three rows in which the **name** value is **customerUpdater**, **customerUpdater2**, and **customerUpdater3** respectively, and the **role** value for all three rows is **customerUpdater**.
3. Save and close the file.
4. Open the *ECE_home***/oceceserver/config/management/charging-settings.xml** file.
5. Add the details of each BRM schema to the file.

   Ensure that you map a unique ID to each schema number for each Customer Updater process.

Note: To support account migration, the definition for each Customer Updater process in the **charge-settings** file must specify the fully qualified name of the Account Migration Manager (AMM) acknowledgment queue in the **amtAckQueueName** entry. (All BRM database schemas use the same AMM acknowledgment queue.)

See "Configuring ECE to Use the AMM Acknowledgment Queue".

For example, to add the details for BRM schema 1:

```
<ConnectionConfiguration
config-class="oracle.communication.brm.charging.appconfiguration.beans.connecti
on.OracleQueueConnectionConfiguration"
         name="customerUpdater1"
         gatewayName="CustomerUpdater"
         hostName="@HOST_NAME@"
         port="@PORT@"
         sid="@SERVICE_NAME@"
         userName="@USER_NAME@"
         Password="@PASSWORD@"
         queueName="@QUEUE_NAME@"
         suspenseQueueName="@SUSPENSE_QUEUE_SUSPENSE@"
         ackQueueName="@RERATING_ACK_QUEUE@"
         amtAckQueueName="@PIN01.AMM_ACK_QUEUE@"
         batchSize="1"
         schemaNumber="1"
</connectionConfiguration>
```

And to add the details for BRM schema 2:

```
<ConnectionConfiguration
         config-
class="oracle.communication.brm.chraging.appconfiguration.beans.connection.Orac
leQueueConnectionConfiguration"
         name="customerUpdater2"
         gatewayName="CustomerUpdater"
         hostName="@HOST_NAME@"
         port="@PORT@"
         sid="@SERVICE_NAME@"
         userName="@USER_NAME@"
         Password="@PASSWORD@"
         queueName="@QUEUE_NAME@"
         suspenseQueueName="@SUSPENSE_QUEUE_SUSPENSE@"
         ackQueueName="@RERATING_ACK_QUEUE"
         amtAckQueueName="@PIN01.AMM_ACK_QUEUE@"
         batchSize="1"
         schemaNumber="2"
</connectionConfiguration>
```

6. Save and close the file.

Configuring the Suspense Queue

ECE moves failed update requests from Customer Updater to the suspense queue.

For information about handling failed events in the suspense queue, see the discussion about handling failed update requests from BRM in *ECE System Administrator's Guide*.

To configure the suspense queue:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.

   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.connectionConfigurations.oracleQueueConnection1**.
3. Expand **Attributes**.
4. Set the **suspenseQueueName** attribute to the name of the queue on the BRM system that holds failed update requests from BRM.

Creating Prepopulated Distribution Tables

By default, Customer Updater loads all BRM customer data into ECE.

To load data for only specified customers (that is, to load data incrementally), you must use prepopulated distribution tables. Only data for the customers listed in the prepopulated distribution tables is loaded.

For example, if customers are migrated from an external billing system into BRM by a customer management tool, the customer management tool can directly load the customers into the BRM database. In such cases, the migrated customers can then be loaded into ECE by using the ECE incremental loading mechanism.

For more information about incremental loading, see "About customerLoader".

To create prepopulated distribution tables:

1. Open the *ECE_home*/**oceceserver/bin/preselect_customer.groovy** file.
2. Specify the filter criteria to use for selecting customers to load by updating the **filterCondition** string.

   Note: The filter criteria you specify are added to the WHERE clause of the SQL query that selects the customer account Portal object IDs (POIDs) from the BRM **account_t** table.

3. Enter the following command:

   ```
   ./preselect_customers.sh -u dbUser -p dbPassword -s dbHost -o oracleSID -c
   dbPort -n noOfdbConnections -f dbFetchSize
   ```

   where:

   - *dbUser* is the name of the BRM database user.
   - *dbPassword* is the password of the BRM database user.
   - *dbHost* is the host name or IP address of the BRM database user.
   - *oracleSID* is the Oracle database alias.
   - *dbPort* is the number for the Oracle database port.

- *noOfdbConnections* is the number of connections to use when extracting BRM customer data from the database. You must set *noOfdbConnections* and the Customer Updater **dbConnections** parameter to the same value. By default, **1**.
- *dbFetchSize* is the number of records to fetch from the database at one time. You must set *dbFetchSize* and the Customer Updater **dbFetchSize** parameter to the same value. By default, **5000**.

  The distribution tables that Customer Updater uses to load BRM customers into ECE are created and populated with details of the selected customers.

4. (Multischema environment only) Run the **preselect_customer.sh** script in each schema.

   When you run the script, specify the database credentials for that schema.

Loading Data from BRM into ECE

To load data from BRM into ECE:

1. Verify that PDC is installed.
2. Verify that the ECE charging server nodes are not running.
3. On the machine on which ECC is installed, go to *ECE_home***/oceceserver/bin**.
4. Start ECC:

   ```
   ./ecc
   ```

5. Run the following commands in this order:

   ```
   start
   start configLoader
   start pricingUpdater
   start customerUpdater
   ```

   These commands start **configLoader**, Pricing Updater, and Customer Updater. In an integrated system:

   - **configLoader** loads mediation specification updates.
   - Pricing Updater loads pricing, configuration, and event definition (request specification data) updates from PDC.

     Note: Only PDC data created *after* Pricing Updater starts is automatically published to ECE.

     PDC data that existed *before* Pricing Updater started is not automatically published to ECE. Instead, you must first extract such data from PDC and then reload it into PDC to trigger publication to ECE. For more information on extracting and reloading the PDC pricing data, see the discussion about importing and exporting pricing and setup components in *PDC User's Guide*.

   - Customer Updater loads data keys, customer data updates, and product cross-reference data from BRM.

   When RADIUS Gateway is started, the data keys are decrypted using the BRM root key in the Oracle wallet file and are stored in the memory with a data key ID

for each data key. These data keys are used for decrypting the passwords in authentication responses from ECE.

After loading the initial customer data from BRM, you can load customer data incrementally, in batches or in bulk, from BRM into ECE. See "Loading Customer Data Incrementally with customerLoader".

Loading Data Keys from BRM into ECE

As part of the initial load of customer data into ECE, Customer Updater loads data keys from BRM into ECE. When you add a new data key or modify an existing data key in BRM, use a JMX editor to load those data keys from BRM into ECE.

To load data keys from BRM into ECE:

1. Access the Customer Updater MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started) and Customer Updater.

      See "Starting and Stopping ECE" in *ECE System Administrator's Guide*.
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the Customer Updater node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.
   e. In the editor's MBean hierarchy, expand the **UpdateEventhandler** node.
2. Expand **Update**.
3. Expand **Operations**.
4. Select **updateDataKeys**.
5. Click the **updateDataKeys** button.

   The newly added or modified data keys are loaded from BRM into ECE.

Configuring ECE for Synchronizing Customer Data with BRM

To configure ECE for synchronizing customer data with BRM, see the following topics:

- Configuring BRM Gateway
- Configuring Notifications for BRM
- Creating the BRM Gateway Suspense Queue
- Enabling Real-Time Synchronization of BRM and ECE Customer Data Updates
- Configuring the CM to Get Real-Time Balances for a Service from ECE
- Configuring Item Assignment in ECE for BRM
- Configuring Life Cycle States in ECE for BRM

Configuring BRM Gateway

BRM Gateway listens on a JMS topic for notifications that ECE publishes. The notifications ECE publishes to the JMS topic trigger update requests to BRM.

To configure BRM Gateway:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.
   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.connectionConfigurations.brmConnection**.
3. Expand **Attributes**.
4. Specify connection values to the BRM Connection Manager (CM):
   - **hostName**. Enter the IP address or the host name of the computer on which BRM is installed.
   - **password**. Enter the password for logging in to BRM.

     If you change the password, you must run a utility to encrypt it. See the discussion about encrypting new passwords in *BRM Elastic Charging Engine System Administrator's Guide* for instructions.
   - **loginName**. Enter the user name for logging in to BRM. The default is **root.0.0.0.1**.
   - **numberofConnections**. Enter the number of connections configured for pooling.
   - **failOverConnectionUrls**. Enter connection values to the additional BRM CMs available for failover in the following format:

     ```
     hostName1:cmPort,hostName2:cmPort,hostName3:cmPort...
     ```

     When this attribute is set, ECE switches to the next available BRM CM when the active connection fails.
   - **cmPort**. Enter the port number for the CM.

Configuring Notifications for BRM

To configure notifications for BRM, see the following topics:

- Configuring JMS for Sending Notifications to BRM Gateway
- Configuring Notifications for BRM

Configuring JMS for Sending Notifications to BRM Gateway

For ECE to publish external notifications, you must configure the JMS credentials for the JMS server on which the ECE notification queue (JMS topic) resides. When you install ECE, the installer prompts you to enter the JMS credentials, which are specified in the *ECE_home*/**oceceserver/config/JMSConfiguration.xml** file. BRM Gateway refers to the **JMSConfiguration.xml** file to obtain the JMS credentials for the ECE notification queue. To change these settings, see "Configuring JMS Credentials for Publishing External Notifications".

Configuring Notifications for BRM

To configure notifications to send information (updates) to BRM from ECE, see the following topics:

- Configuring Billing Notifications
- Configuring Notifications Replenishing POID IDs
- Configuring Life Cycle Transition Notifications
- Configuring First-Usage Validity Notifications

Configuring Billing Notifications

To configure billing notifications:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.
   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.notification**.
3. Expand **Attributes**.
4. Set the **billingNotificationMode** attribute to one of the following values:
   - To disable the generation of billing notifications, enter **NONE**.
   - To generate an external notification and send it to a notification queue (JMS topic) when the **BILLING_NOTIFICATION_EVENT** service event is created, enter **ASYNCHRONOUS**.

Configuring Notifications Replenishing POID IDs

You configure notifications that trigger BRM to update POID IDs from ECE to BRM for each BRM schema. For information about setting the initial caching of POID IDs, see "Configuring Item Assignment in ECE for BRM".

To configure notifications that trigger BRM to update POID IDs:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.
   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.notification**.

3. Expand **Attributes**.
4. Set the **replenishPoidIdNotificationMode** attribute to one of the following values:
   - To disable the generation of life cycle transition notifications, enter **NONE**.
   - To generate an external notification and send it to a notification queue (JMS topic) when the **REPLENISH_POIDID_NOTIFICATION_EVENT** service event is created, enter **ASYNCHRONOUS**.

Configuring Life Cycle Transition Notifications

Configure life cycle transition notifications so that when changes occur to a customer's subscriber life cycle state in ECE, ECE sends the subscriber life cycle state information to BRM so that BRM can update the state in the BRM database.

To configure life cycle transition notifications:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.
   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.notification**.
3. Expand **Attributes**.
4. Set the **lifeCycleTransitionNotificationMode** attribute to one of the following values:
   - To disable the generation of life cycle transition notifications, enter **NONE**.
   - To generate an external notification and send it to a notification queue (JMS topic) when the **LIFE_CYCLE_TRANSITION_NOTIFICATION_EVENT** service event is created, enter **ASYNCHRONOUS**.

Configuring First-Usage Validity Notifications

Configure first-usage validity notifications for synchronizing validity of first-usage resources from ECE to BRM and from ECE to NCC Notification Gateway.

To configure first-usage validity notifications:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.
   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.

2. Expand **charging.notification**.
3. Expand **Attributes**.
4. Set the **firstUsageValidityInitNotificationMode** attribute to one of the following values:
   - To disable the generation of first-usage validity notifications, enter **NONE**.
   - To generate an external notification and send it to a notification queue (JMS topic) when the **FIRST_USAGE_VALIDITY_INIT_NOTIFICATION_EVENT** service event is created, enter **ASYNCHRONOUS**.

Creating the BRM Gateway Suspense Queue

If you installed an ECE integrated installation for the first time, you can skip this task. The ECE post-installation script automatically creates the BRM Gateway suspense queue.

If you installed a patch set (for example, ECE 11.3 Patch Set 2) on top of an existing ECE integrated installation, use the WebLogic server to manually create the BRM Gateway suspense queue.

To create the BRM Gateway suspense queue manually:

1. Log on to a WebLogic server on which a notification queue (JMS topic) resides.
2. Log in to WebLogic Server Administration Console.
3. In the **Services** section, select **JMS Modules**.
4. On the Summary of JMS Modules page, click **ECE Module**.
5. In the **Change Center** pane, click **Lock and Edit**.
6. In the **Summary of Resources** table, click **New**.

   The **Summary of Resources** table contains the resources of the JMS modules.
7. On the Create a New JMS System Module Resource page, select **Queue**.
8. Click **Next**.
9. In the **Name** field, enter **SuspenseQueue**.
10. In the **JNDI Name** field, enter **ECE/SuspenseQueue**.
11. Click **Next**.
12. From the **Subdeployments** list, select **ECEQUEUE**.
13. From the **Targets** list, select **NotificationServer**.
14. Click **Finish**.
15. Click **Activate**.

    The changes are applied immediately.

Enabling Real-Time Synchronization of BRM and ECE Customer Data Updates

When customer data is updated in the BRM database, the updates must be applied synchronously (in real time). For more information, see the discussion about synchronizing BRM and ECE customer data in *BRM Elastic Charging Engine Concepts*.

To enable real-time synchronization of BRM and ECE customer data updates:

1. Open the *BRM_home*/**sys/cm/pin.conf** file in a text editor.

2. Add the following entries to the end of the file:

```
-cm ece_real_time_sync_db_no DBNumber
-cm em_group ece PCM_OP_ECE_PUBLISH_EVENT
-cm em_pointer ece ip emGateway_host emGateway_port
```

   where:

   - *DBNumber* is the publisher database number for EM Gateway.

     Note: By default, the publisher database number for EM Gateway is 0.0.9.8.

   - *emGateway_host* is the name or IP address of the server on which EM Gateway is running.
   - *emGateway_port* is the number of the port through which EM Gateway connects to the host.

3. Save and close the file.
4. If your publisher database number for EM Gateway is not 0.0.9.8:
   a. Open the *BRM_home***/sys/eai_js/payloadconfig_ifw_sync.xml** file (or the merged file if you merged payload configuration files) in a text editor.
   b. Locate the **PublisherDefs** section.
   c. In the **Publisher DB="***DBNumber***"** entry, replace *DBNumber* with the publisher database number of your EM Gateway.
   d. Save and close the file.
5. If you changed the publisher database number for EM Gateway in the **payloadconfig_ifw_sync.xml** file, restart the Payload Generator External Module (also called the Enterprise Application Integration (EAI) Java Server or **eai_js**).
6. Restart the CM.

   See the discussion about starting and stopping the BRM system in *BRM System Administrator's Guide*.

Configuring the CM to Get Real-Time Balances for a Service from ECE

The CM connects to ECE through EM Gateway.

To configure the CM to get real-time balances for a service from ECE:

1. Open the *BRM_home***/sys/cm/pin.conf** file in a text editor.
2. Add the following entry:

```
- cm em_group ece_bal PCM_OP_BAL_GET_ECE_BALANCES
```

3. Set the following entry to match your environment:

```
- cm em_pointer ece_bal ip emGateway_host emGateway_port
```

   - *emGateway_host* is the name or IP address of the server on which EM Gateway is running.
   - *emGateway_port* is the number of the port through which EM Gateway connects to the host.

4. Save and close the file.
5. Stop and restart the CM.

Configuring Item Assignment in ECE for BRM

You configure item assignment in ECE so that customer balance impacts can be applied to bill items. The default item type configuration is sufficient for default item assignments. For custom item assignments, configure item assignments and the appropriate item type selectors in PDC. See the PDC Help.

If you configured delayed billing in BRM, you must configure item assignment in ECE to process delayed usage requests in the appropriate accounting cycle.

To configure item assignment in ECE:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home***/oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.
   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.itemAssignmentConfig**.
3. Expand **Attributes**.
4. Review the following attributes:
   - **itemAssignmentEnabled**: Enter **true** to turn on item assignment or **false** to turn it off.
   - **poidQuantityPerSchema**: Double-click the **Value** field. A list of schemas and the quantity of POID IDs that is reserved at ECE startup for each schema appears. To add schemas to the list or to change the quantity of POID IDs for a schema in the list, see step 5.
   - **delayToleranceIntervalInDays**: Enter the number of days during which delayed usage requests are processed for the current accounting cycle. This interval must be less than the delayed billing interval (the value of the **config_billing_delay** entry in the *BRM_home***/sys/cm/pin.conf** file). See the discussion about processing delayed usage requests for more information.
5. To add a schema to the **poidQuantityPerSchema** list or to change the quantity of POID IDs for a schema in the list:
   a. Expand **Operations**.
   b. Select **setPoidQuantity**.
   c. Specify values for the following parameters:

      schema: Enter the BRM schema number for which the POID IDs must be reserved. For example, in a multischema environment, enter **1** for the primary schema, **2** for the secondary schema, and so on.

quantity: Enter the number of POID IDs that must be reserved at ECE startup for the specified schema.

d. Click the **setPoidQuantity** button.

Configuring Life Cycle States in ECE for BRM

ECE supports the BRM subscriber life cycle state feature. If the subscriber life cycle state feature is disabled in BRM, ECE supports only the default subscriber life cycle, which has the following states: Active, Inactive, and Closed. If the subscriber life cycle state feature is enabled in BRM, ECE supports custom subscriber life cycles, which has the following states: Preactive, Active, Recharge Only, Credit Expired, Fraud Investigated, Dormant, Suspended, and Closed. See the discussions about service life cycles and enabling BRM to use custom service life cycles in *BRM Managing Customers* for more information.

You can customize the preconfigured subscriber life cycle state. You must configure life cycle states in ECE, so they stay synchronized with life cycle states you add in BRM.

To configure life cycle states in ECE for BRM:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.

   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.lifecycleConfiguration**.
3. Expand **Operations**.
4. Select **addLifecycleDetails**.
5. Specify values for the operation's parameters to associate the appropriate life cycle state business rules and transitions with each product type in your system.

   For more information about life cycle state business rules and transitions, see the discussion about managing service life cycles in *BRM Managing Customers*.
6. Click the **addLifecycleDetails** button.

   Your configurations are saved to the *ECE_home*/**oceceserver/config/management/charging-settings.xml** file.

Configuring ECE to Send Rated Events to BRM

The ECE charging servers generate rated events as a result of usage-request processing. ECE sends the rated events to BRM so that customer balances in BRM can be updated.

For ECE to send rated events to BRM, you must configure the following components:

- Rated Event Publisher

See "Enabling and Configuring Rated Event Publisher".

- Rated Event Formatter

  See "Installing and Configuring Rated Event Formatter".

- BrmCdrPluginDirect Plug-in

  See "Configuring the Rated Event Formatter Output".

- Rated Event Loader

  See "Setting Up Rated Event Loader for ECE".

Assuming these components are configured and Rated Event Formatter is running, the ECE charging servers generate event data record (EDR) files as a result of usage-request processing. The BrmCdrPluginDirect Plug-in formats the EDR files in BRM call details record (CDR) format. The EDRs can then be loaded by Rated Event (RE) Loader into the BRM database. See "Setting Up Rated Event Loader for ECE".

Enabling and Configuring Rated Event Publisher

For ECE to send rated events to BRM, you must enable and configure Rated Event Publisher. Rated Event Publisher publishes ECE-generated rated events to the Oracle NoSQL database data store.

To verify that Rated Event Publisher is enabled and to configure Rated Event Publisher:

1. Obtain the BRM database schema number (PoidDB) of each target BRM database schema.
2. Open the *ECE_home*/**oceceserver/config/charging-cache-config.xml** file.
3. For the RatedEventPublisher module, verify that the following entry is used for the cache-store configuration:

```
<init-param>
  <param-name>cache-store</param-name>

<param-value>oracle.communication.brm.charging.ratedevent.publisher.internal.coherence.RatedEventPublisher</param-value>
```

4. Save the file.
5. Open the *ECE_home*/**oceceserver/config/management/charging-settings.xml** file.
6. In the **ratedEventPublisher** section, set the required configuration parameters.

   Table 3–1 provides configuration parameter descriptions and default values.

Table 3–1 Rated Event Publisher Configuration Parameters

Name	Default	Description and Guideline
dataStoreConnection	"localhost:5000"	This parameter configures Rated Event Publisher to connect to the Oracle NoSQL database; it configures the data store connection to the Oracle NoSQL database system. The Oracle NoSQL database connection string uses the format *hostname*:*port* for connecting to a pre-configured Oracle NoSQL database system. The default is "localhost:5000" for connecting to a standalone Oracle NoSQL database system (KV-Lite).
dataStoreName	"kvstore"	This parameter configures the data store name to an Oracle NoSQL database system. The data store name is for using a pre-configured data store in an Oracle NoSQL database system. The default is "kvstore" for using a standalone Oracle NoSQL database system (KV-Lite).
threadPoolSize	"4"	This parameter configures the number of threads in the thread pool. Multiple threads can be used in a RatedEventPublisher module where each thread can publish rated events to an Oracle NoSQL database system independently. The valid number is greater than zero. For best performance, Oracle recommends that you set this parameter to the number of Oracle NoSQL database partitions. Setting the number of threads higher than the number of partitions does not increase performance. Threads that you configure higher than the number of partitions are not used.

7. Save the file.

For information about how to verify that Rated Event Publisher published rated events to NoSQL, see "Verifying That Rated Events Are Published to Oracle NoSQL Database".

Installing and Configuring Rated Event Formatter

To send rated events to BRM, you must install and configure Rated Event Formatter. Rated Event Formatter extracts rated events from the Oracle NoSQL database data store and formats them into CDR files in the format required for loading them into the BRM database. For information about Rated Event Formatter, see the discussion about ECE system architecture in *BRM Elastic Charging Engine Concepts*.

To format rated events for a non- BRM system instead of the BRM database, you must configure an additional Rated Event Formatter instance.

Installing Rated Event Formatter

Rated Event Formatter is included in the ECE Server software package. Install Rated Event Formatter on a machine that is part of the ECE topology. RE Loader should be able to access the same file system as Rated Event Formatter. See "Setting Up Rated Event Loader for ECE".

Configuring Rated Event Formatter

To configure Rated Event Formatter:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.
   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.ratedEventFormatters.***Instance_Name*, where *Instance_Name* is the name of the instance you want to configure.
3. Expand **Attributes**.
4. Specify values for **name** and **partition** and for any remaining attributes you need to set for the instance.

   See Table 3–2 for attribute descriptions and default values.

Table 3–2 Configuration Parameters for a Rated Event Formatter Instance

Name	Default	Description
name	"formatter"	The name of a Rated Event Formatter instance. Name Rated Event Formatter instances consistently and uniquely (for example, **formatter1**, **formatter2**, and so on). **name** must match the name of the Rated Event Formatter node instance in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.
partition	"1"	The partition for the rated events to be processed by a Rated Event Formatter instance. **partition** must match the target BRM database schema number for the schema to which the ECE rated event is to be exported. For example, for a BRM multischema environment, BRM schema **0.0.0.1**, **partition** must be set to **1**; for BRM schema **0.0.0.2**, **partition** must be set to **2**; and so on. See "Configuring ECE for a Multischema BRM Environment" for additional information about configuring ECE for a multischema environment. You may want to name your formatter instances to correlate with the partition number. For example, one instance of Rated Event Formatter named **formatter1** can process rated events to be exported to the BRM schema **0.0.0.1** (**partition** value **1**), and another instance of Rated Event Formatter named **formatter2** can process rated events to be exported to the BRM schema **0.0.0.2** (**partition** value **2**), and so on.
dataStoreConnection	"localhost:5000"	The connection information to the Oracle NoSQL database. The connection string consists of host name and port number for connecting to an Oracle NoSQL system.
dataStoreName	"kvstore"	The data store name to be used to access an Oracle NoSQL system.
retainDuration	0	The duration in seconds that rated events must be retained in the Oracle NoSQL database after they have been processed before they can be purged. Set the value to the seconds you want to retain rated events in the Oracle NoSQL database after Rated Event Formatter has published the rated events as RE Loader records (CDR records). The default is **0**, which means that as soon as rated events are processed, they are purged immediately. For more information, see the discussion about managing persisted data in the Oracle NoSQL database in *BRM Elastic Charging Engine System Administrator's Guide*.
ripeDuration	60	The duration in seconds that rated events have existed before they can be processed. This setting must be greater than the time it takes for ECE charging servers to fully recover after failure. Delaying the processing of rated events up to the **ripeDuration** time allows time for resolving any duplicate rated events that may have been persisted to the Oracle NoSQL database. The **ripeDuration** value is the minimum number of seconds rated event information must be stored in the Oracle NoSQL Database before the Rated Event Formatter can read it.

Table 3–2 (Cont.) Configuration Parameters for a Rated Event Formatter Instance

Name	Default	Description
checkPointInterval	4	The time range in seconds used by the Rated Event Formatter instance to read a set of rated events at a repeated time interval. Valid values must be the following: ■ Less than or equal to the value of **ripeDuration** ■ Evenly divisible by the number of threads configured for **threadPoolSize** This is the number of seconds Rated Event Formatter waits before reading a batch of rated event information. If rated event information in a batch has not yet met the value of **ripeDuration**, the Rated Event Formatter does not read it.
threadPoolSize	4	The number of threads used by the Rated Event Formatter instance to process a set of rated events for each time range defined by **checkPointInterval**. Valid values are greater than zero and up to any number the system resources allow. Tune this value to the expected workload in the deployed environment.
pluginPath	n/a	The path to the JAR library that contains the reader plug-in implementation. A custom plug-in has a modified path to the JAR library.
pluginName	n/a	The class name with the package path for the formatter plug-in object to be called by Rated Event Formatter. Rated Event Formatter is configured by default to use the BrmCdrPluginDirect plug-in. If you are using the BRM database to process rated events, the *pluginName* is: **oracle.communication.brm.charging.ratedevent.formatterplugin.BrmCdrPlugin** If you are using a third-party system to process rated events, the *pluginName* is: **oracle.communication.brm.charging.ratedevent.custom.CustomPlugin**
logFormatterWorker	false	Whether to enable (**true**) or disable (**false**) logging for the worker thread pool.

5. Change directory to the *ECE_home***/oceceserver/bin** directory.
6. Start ECC:

   ```
   ./ecc
   ```

7. Stop and restart any Rated Event Formatter instances that you configured.

   Each instance reads its configuration information by name.

   For information about stopping and starting Rated Event Formatter instances, see the discussion about starting and stopping ECE in *BRM Elastic Charging Engine System Administrator's Guide*.

Troubleshooting Rated Event Formatter Processing

If you suspect a problem with how Rated Event Formatter processes rated events, look in the *ECE_home***/oceceserver/logs/rated-event-formatter.log** file for errors. The log file

contains information about how many rated events are processed, how many are purged, and any errors during rated-event processing.

Configuring the Rated Event Formatter Output

Rated events can be processed in different formats with a BRM database or a third-party system.

To process rated events in a BRM database, you must configure the BrmCdrPluginDirect plug-in. The BrmCdrPluginDirect plug-in formats ECE-rated events into BRM format.

To process rated events using a third-party system, you must configure a custom plug-in. The custom plug-in formats ECE rated events in various formats.

For more information about the BrmCdrPluginDirect plug-in, see the discussion about ECE system architecture in *BRM Elastic Charging Engine Concepts*.

To configure the BrmCdrPluginDirect plug-in:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.
   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.brmCdrPlugins.***Instance_Name*, where *Instance_Name* is the name of the instance that you want to configure.
3. Expand **Attributes**.
4. Specify values for the following attributes as needed:

Table 3–3 BrmCdrPluginDirect Plug-in Configuration Entries

Name	Default	Description
tempDirectoryPath	"/tmp/tmp"	The directory path for the BrmCdrPluginDirect plug-in to store temporary files while processing the rated event objects.
doneDirectoryPath	"/tmp/done"	The directory path for the BrmCdrPluginDirect plug-in to store completed CDR files from processing the rated event objects.
doneFileExtension	".done"	The file extension for the completed CDR files created by the BrmCdrPluginDirect plug-in.

5. Stop and start the Rated Event Formatter instance associated with the BrmCdrPluginDirect plug-in instance.

Setting Up Rated Event Loader for ECE

Rated Event Formatter extracts preprocessed rated events from the Oracle NoSQL database data store over TCP/IP, and the BrmCdrPluginDirect plug-in stores the completed CDR files in the directory you specify. RE Loader then picks up the CDR files from that directory for processing.

RE Loader and Rated Event Formatter can reside on the same machine or on separate machines, but they must both be able to access the same file system. Specify the output directory of the BrmCdrPluginDirect plug-in as the input directory of RE Loader. You must configure the RE Loader Batch Controller to pick up the CDR files.

For information about installing and configuring RE Loader to load preprocessed EDRs into the BRM database, see *BRM Configuring Pipeline Rating and Discounting*. The BRM documentation refers to Pipeline Manager when describing how to configure RE Loader to load EDRs into the BRM database; that information also applies to ECE. RE Loader processes ECE-rated events similar to how it processes pipeline-rated events.

ECE writes all event fields to CDRs. If you want RE Loader to load only specific event fields into the BRM database, you must modify the RE Loader control files to indicate which event fields you want it to load. Control files with ECE control data are included with ECE. You copy and merge these files onto the BRM system. For information about editing control files, see the Oracle database utilities documentation at the following location:

`http://docs.oracle.com/cd/B10501_01/server.920/a96652/ch05.htm#1004643`

Configuring ECE for Rerating

You configure ECE for rerating so that it can rerate usage events when rerating is run in BRM. You might run rerating for various reasons (for example, if one of your existing charge offers was replaced between the last and next billing cycles). For information about rerating in the BRM charging system, see the discussion about rerating in *BRM Setting Up Pricing and Rating*. For information about how ECE rerates usage events, see the discussion about understanding charging scenarios in *BRM Elastic Charging Engine Concepts*.

BRM and ECE work together for rerating. You run rerating jobs from BRM and BRM sends *charging requests* to ECE for rerating the usage events. ECE rerates the usage events for the specified customers in the rerate jobs and then sends the rerated events back to BRM. For the overall rerating process to work, you configure various components on the ECE and BRM side.

For information about configuring ECE for rerating, see the following topics:

- Configuring the Rerating Acknowledgment Queue
- Configuring BRM for ECE Rerating

> **Important:** For ECE to perform rerating, EM Gateway must be running.

After all components are configured, you can run rerating in BRM using the **pin_rerate** utility. For instructions about running rerating by using the **pin_rerate** utility, see *BRM Setting Up Pricing and Rating*.

For information about handling rerating errors in ECE, see the discussion about troubleshooting ECE in *BRM Elastic Charging Engine System Administrator's Guide*.

Configuring the Rerating Acknowledgment Queue

The rerating acknowledgment queue is used to send rerating-related acknowledgments from ECE to BRM; it is an Oracle AQ database queue on the BRM system.

The rerating acknowledgment queue is configured during installation. See the discussion about post-installation tasks in *BRM Elastic Charging Engine Installation Guide* for information about creating this queue. You can change the acknowledgment queue configuration.

To configure the acknowledgment queue:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.
   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.connectionConfigurations.oracleQueueConnection**.
3. Expand **Attributes**.
4. Set the **ackQueueName** attribute to the name of the acknowledgment queue for posting rerating-related acknowledgment events.

Configuring BRM for ECE Rerating

To configure BRM for ECE rerating:

1. Enable ECE rerating in BRM.

   For information, see the discussion about enabling ECE rerating in *BRM Setting Up Pricing and Rating*.
2. Configure the CM for ECE rerating.

   For information, see the discussion about configuring the CM for ECE rerating in *BRM Setting Up Pricing and Rating*.
3. Verify that the acknowledgment queue (Oracle Advanced Queuing (AQ) database queue) is created.

   Note: The acknowledgment queue is created by an ECE post-installation script. For more information, see the discussion about post-installation tasks in *BRM Elastic Charging Engine Installation Guide*.
4. Verify that EM Gateway is configured and running.

   See "Reconfiguring External Manager Gateway".

   If EM Gateway is not running, do the following:
   a. On the driver machine, go to the *ECE_home*/**oceceserver/bin** directory.
   b. Start ECC:

      ```
      ./ecc
      ```
   c. Run the following command:

```
start emGateway
```

5. Verify that BRM Gateway is configured and running.

   See "Configuring BRM Gateway".

   If BRM Gateway is not running, run the following command:

   ```
   start brmGateway
   ```

You can now run rerating using the **pin_rerate** utility. For more information, see the discussion about rerating and the **pin_rerate** utility in *BRM Setting Up Pricing and Rating*.

Enabling ECE to Rate Events during Account Migration

In BRM, account migration consists of transferring the data associated with accounts from a source BRM database schema to a target BRM database schema. Account migration is performed by Account Migration Manager (AMM). For more information about account migration, see the discussion about migrating accounts in *BRM System Administrator's Guide*.

To enable ECE to rate events during account migration:

1. Configure the definition of your system's AMM controllers for ECE:
   a. Open the *BRM_home*/**sys/amt/Infranet.properties** file in a text editor.
   b. Verify that the following entry is set to **true**:

      ```
      controller_1_event_generation=true
      ```

   c. If more than one controller is defined in the file, ensure that each controller's **controller**_*controllerNumber*_**event_generation=** entry is set to **true**.
   d. Save and close the file.

      The change takes effect the next time the **pin_amt** utility is run. For more information, see the discussion about that utility in *BRM System Administrator's Guide*.
2. Configure ECE to use the AMM acknowledgment queue. See "Configuring ECE to Use the AMM Acknowledgment Queue" for more information.
3. Verify that Customer Updater is correctly configured for your system and running.

   ECE receives AMM business events from the BRM Account Synchronization Data Manager (DM) database queue through Customer Updater. Because account migration involves multiple database schemas, Customer Updater must be configured to support all the database schemas in your system. It must also be configured to send AMM-related acknowledgments from ECE to your system's AMM acknowledgment queue.

   For information about configuring Customer Updater, see the following:

   - Configuring Customer Updater for a BRM Multischema Environment
   - Configuring ECE for a Multischema BRM Environment
4. Verify that BRM Gateway is correctly configured for your system and running.

   See "Configuring BRM Gateway".
5. Verify that Rated Event Formatter is correctly configured for your system and running.

See "Configuring Rated Event Formatter".

After you enable ECE to rate events during account migration, the AMM business events listed in Table 3–4 automatically notify ECE that an account migration job is occurring. They enable ECE to track the status of the migration, to notify BRM if ECE fails to clear its rated event data store before the migration begins (a migration prerequisite), to continue rating events while the accounts are migrated, and to update its information about the target database schema for successfully migrated accounts.

Table 3–4 ECE Response to AMM Business Events

AMM Business Event	ECE Response
HoldCDRProcessing	1. Gets the migration job ID, the source database schema, and the target database schema from this event. 2. Queries the BRM database for the list of accounts that belong to the migration job. 3. Waits for all existing rated events associated with those accounts to be extracted from the Oracle NoSQL database data store. 4. Does one of the following: If the extraction succeeds: – Assigns the IN_ACCOUNT_MIGRATION status to the accounts. – Updates their target database schema information. – Sends an ACKHoldCDRProcessing acknowledgment to the BRM acknowledgment queue. – Continues rating incoming usage events for the migrated accounts but does not extract them from the Oracle NoSQL database data store. If the extraction fails, ECE sends a NACKHoldCDRProcessing acknowledgment to BRM, and BRM does not migrate the accounts.
MigrateAcct	Sends an ACKMigrateAcct acknowledgment to the AMM acknowledgment queue.
MigrateSource	Sends an ACKMigrateSource acknowledgment to the AMM acknowledgment queue.
MigrateDestination	Sends an ACKMigrateDestination acknowledgment to the AMM acknowledgment queue.
ResumeCDRProcessing	1. Gets the migration job ID, the source database schema, and the target database schema from this event. 2. Queries the BRM database for the list of accounts that belong to the migration job. 3. Removes the IN_ACCOUNT_MIGRATION status from those accounts. 4. Loads all the rated events that were generated while the accounts' status was IN_ACCOUNT_MIGRATION into the new target database schema.

For more information about the events in Table 3–4, see the discussion about AMM business events in *BRM System Administrator's Guide*.

Configuring ECE to Use the AMM Acknowledgment Queue

The AMM acknowledgment queue is used to send AMM-related acknowledgments from ECE to BRM; it is an Oracle AQ database queue on the BRM system.

To configure ECE to use the AMM acknowledgment queue:

1. Access the ECE MBeans:
   a. Log on to the driver machine.

b. Start the ECE charging servers (if they are not started).

c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.

d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

   The **eceTopology.conf** file also contains the host name and port number for the node.

e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.

2. Expand **charging.connectionConfigurations.oracleQueueConnection**.

3. Expand **Attributes**.

4. Set the **amtAckQueueName** attribute to the fully qualified name of the acknowledgment queue on which the **pin_amt** utility listens to AMM-related acknowledgment events:

   schema.*ammAcknowledgmentQueue*

   where:

   - *schema* is the name of the BRM database schema in which the AMM acknowledgment queue resides.
   - *ammAcknowledgmentQueue* is the name of the AMM acknowledgment queue.

   For example:

   ```
   PIN01.AMM_ACK_QUEUE
   ```

Configuring External Manager Gateway for High Availability

To configure EM Gateway for high availability:

1. Open the *BRM_home*/**sys/cm/pin.conf** file in a text editor.

2. Add additional EM Gateway pointers to the file to use for failover. For example:

   ```
   -cm ece_real_time_sync_db_no 0.0.9.8
   -cm em_group ece PCM_OP_ECE_PUBLISH_EVENT
   -cm em_pointer ece ip emGateway_host1 emGateway_port1
   -cm em_pointer ece ip emGateway_host1 emGateway_port2
   -cm em_pointer ece ip emGateway_host2 emGateway_port3
   -cm em_pointer ece ip emGateway_host2 emGateway_port4
   ```

   where

   - *emGateway_host1* is the name or IP address of the server on which one instance of EM Gateway is running.
   - *emGateway_port1* is the primary port number for EM Gateway on host 1.
   - *emGateway_port2* is a backup port number for EM Gateway on host 1.
   - *emGateway_host2* is the name or IP address of the server on which another instance of EM Gateway is running. (EM Gateway can run on multiple hosts.)
   - *emGateway_port3* is the primary port number for EM Gateway on host 2.
   - *emGateway_port4* is a backup port number for EM Gateway on host 2.

3. Set the value of the following entry to 2 or higher:

```
- cm cm_op_max_retries 2
```

This entry specifies the maximum number of times an opcode is retried in the CM. The default is **1**. For high availability, the value must be at least **2**.

4. Save and close the file.
5. Restart the CM.

   See the discussion about starting and stopping the BRM system in *BRM System Administrator's Guide*.

For more information, see the discussion about high availability with the Elastic Charging server in *BRM Elastic Charging Engine Concepts*.

Configuring ECE for a Multischema BRM Environment

This section describes configuration tasks associated with configuring ECE for a multischema BRM environment.

For a multischema BRM environment, each schema publishes business events to its own Oracle Advanced Queuing (AQ) database queue for ECE to consume. Configure one ECE Customer Updater for each AQ database queue. See "Configuring Customer Updater for a BRM Multischema Environment" for instructions.

For a multischema BRM environment, each Rated Event Formatter instance can be configured to process rated events belonging to a specific BRM schema. See "Configuring Rated Event Formatter" for instructions.

The preceding points cover the configuration you need to perform in ECE for a multischema environment. The Customer Updater instance connected to the specific schema performs the initial extract and load of customer data from the BRM schema into ECE.

4

Network Integration for Online Charging Using Diameter Gateway

This chapter provides information about network integration for online charging using Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE) Diameter Gateway.

Before reading this chapter, you should be familiar with ECE concepts and architecture. See the following chapters in *BRM Elastic Charging Engine Concepts*:

- ECE Overview
- ECE System Architecture

Overview of Network Integration Using Diameter Gateway

The following steps summarize how to set up network integration for online charging using Diameter Gateway, which enables Diameter Gateway to do the following:

- Receive Gy, Sp, and Sy Diameter requests from Diameter clients and translate them into ECE requests.
- Submit ECE requests to ECE charging servers for credit-control processing.
- Receive ECE request responses and translate them into respective Gy, Sp, and Sy Diameter message responses.
- Send Diameter message responses to Diameter clients.
- Consume JMS notifications from the ECE notification queue, construct Diameter notification messages from them, and send the notification messages to the appropriate Diameter clients.

> **Important:** Before sending requests to ECE using Diameter Gateway, you must first implement ECE with PDC and BRM. See "Integrating ECE with PDC" and "Integrating ECE with BRM" for instructions.

1. Install ECE.

   Diameter Gateway is included in the ECE Server software package.

2. Add Diameter Gateway node instances required for your topology and configure each instance.

   See the discussion about post-installation tasks in *BRM Elastic Charging Engine Installation Guide* for instructions on how to add and configure Diameter Gateway node instances.

3. For all request types you receive from the network, ensure that your credit-control request (CCR) message formats adhere to the attribute value pair (AVP) fields that Diameter Gateway supports and requires.
4. For Gy interface Diameter requests, ensure that you have done the following:
   - Loaded your event definitions (which includes request specification data) from PDC into ECE.
   - Edited your mediation specification file and loaded it into ECE.

     The mediation specification enables Diameter Gateway to associate each Gy interface Diameter request with its respective usage-request builder.

     See "Network Integration for Gy Interface Requests".
5. Configure notifications for Diameter Gateway.

   Diameter Gateway listens for notifications on the ECE (JSM) notification queue (for push notifications from Elastic Charging Server).

   You must enable the notification types to be generated in the ECE system. See "Configuring Notifications for Diameter Gateway".

   You can set tuning parameters for how Diameter Gateway sends notifications to Diameter clients. For information, see the discussion about setting Diameter Gateway node properties in the post-installation tasks chapter of *BRM Elastic Charging Engine Installation Guide*.

   If a Diameter client fails or becomes unavailable before receiving a notification message from a Diameter Gateway instance, Diameter Gateway can route the notification message to another available Diameter peer. For information, see "Configuring Alternative Diameter Peers for Notifications".

   See "Configuring Notifications for Charging" for information about configuring the JMS credentials in ECE for the ECE notification queue.
6. Start the Diameter Gateway nodes.

   When the Diameter Gateway nodes start, they automatically join the Coherence cluster gaining access to ECE caches and invocation services that it uses to send requests to ECE. At startup, the Diameter Gateway instances read from the ECE notification queue for notifications.

   See the discussion about starting and stopping ECE in *BRM Elastic Charging Engine System Administrator's Guide* for information about starting Diameter Gateway nodes.

Network Integration for Sp and Sy Interface (Policy) Requests

This section provides information about network integration for policy requests using Diameter Gateway.

Given that the technical implementation of Sp has not been defined by the 3GPP standards body, Diameter Gateway uses the Sh interface as the implementation to request and subscribe to policy-related information in the ECE server.

Diameter Gateway uses the ECE policy API for retrieving Sp and Sy information from ECE charging servers and sends the information to the Policy and Charging Rules Function (PCRF).

The following Sp (implemented as Sh) and Sy interface policy request types are processed by Diameter Gateway (using the ECE policy-request builders).

Sy:

- Spending-Limit-Report-Request (SLR/SLA)
- Subscribe-Notification-Request

Sp/Sh:

- User-Data-Request (UDR)
- Subscribe-Notifications-Request (SNR)
- Push-Notification-Request (PNR)

Diameter Gateway manages notification subscriptions (when the PCRF subscribes and unsubscribes) for notifications due to Sy and Sp related updates.

Diameter Gateway listens for notifications on the ECE (JSM) notification queue (for push notifications from the Elastic Charging Server). For policy-driven charging, when changes occur to policy counters (balances) or to policy-related subscriber preferences that are associated with products that have an active policy session, ECE charging servers publish asynchronous notifications to the JMS notification queue. Diameter Gateway subscribes for receiving the policy notifications at startup and processes them as follows:

- From the ECE spending-limit JMS notifications it receives, Diameter Gateway creates Sy (Spending-Status-Notification-Request (SNR)) messages for all subscribed sessions and routes them to the appropriate Diameter clients.
- From the ECE subscriber-preferences JMS notifications it receives, Diameter Gateway creates Sp/Sh (Push-Notification-Request (PNR)) messages for all subscribed sessions and routes them to the appropriate Diameter clients.

For information about how Diameter Gateway processes policy requests, see the overview of Elastic Charging Engine and see the discussion about understanding charging scenarios in *BRM Elastic Charging Engine Concepts*.

For information about how Diameter Gateway uses the ECE policy management APIs to retrieve Sy-interface and Sp-interface data from the ECE server, see "Integrating Policy Clients with ECE".

To enable Diameter Gateway to create ECE requests for policy-driven charging, you must configure notifications for Diameter Gateway. See "Configuring Notifications for Diameter Gateway". You can configure alternate Diameter peers for each peer to which a Diameter Gateway instance connects for routing notifications. See "Configuring Alternative Diameter Peers for Notifications" for more information.

Ensure that your policy CCR message formats adhere to the to the well-known AVP fields of the 3GPP standard for Sh and Sy policy requests.

For information about how Diameter Gateway complies with standard AVPs for the Sy and Sh interfaces, see the Diameter Sy Protocol and Diameter Sh Protocol chapters of the *BRM Elastic Charging Engine Diameter Gateway Protocol Implementation Conformance Statement*.

Network Integration for Gy Interface Requests

This section provides information about network integration for Gy interface online charging requests using Diameter Gateway.

The following Gy interface credit-control request types are processed by Diameter Gateway (using ECE usage-request builders):

- Session-based requests
  - Initiate
  - Update
  - Terminate
  - Cancel
- Price enquiry
- Direct debit
- Refund

For Gy interface credit-control requests, you must do the following for Diameter Gateway to process the requests successfully:

- Present Gy interface request types inside of a Multiple-Service Credit Control (MSCC) group.

  MSCC AVPs are part of the CCR and Diameter Gateway expects each Gy interface request type to be included in the MSCC group even if the request contains only a single service. Contain the following Gy interface request types in a MSCC group:

  - Initiate
  - Update
  - Terminate
  - Cancel
  - Price enquiry
  - Direct debit
  - Refund

  For more information about MSCC requests and ECE, see "Using Multiple Services Credit Control (MSCC)".

- Add network attributes for all event attributes in the event definition that apply to usage-request charging operations.

  Diameter Gateway uses the network specification and corresponding network attributes to dynamically populate the event attributes of ECE requests with the CCR AVP data of your incoming Diameter request.

  See "Constructing Usage Requests".

- Edit your mediation specification file and load your mediation specification into ECE.

  The mediation specification enables Diameter Gateway to associate each Gy interface Diameter request with its respective usage-request builder.

  See "Editing the Mediation Specification File".

For information about how Diameter Gateway complies with standard AVPs for the Gy interface, see the *BRM Elastic Charging Engine Diameter Gateway Protocol Implementation Conformance Statement*.

Diameter Gateway uses incremental based accounting behavior when processing usage requests.

Diameter Gateway listens for notifications on the ECE (JSM) notification queue (for push notifications from the Elastic Charging Server). From the ECE

reauthorization-request JMS notifications it receives, Diameter Gateway creates Gy reauthorization-request (RAR) messages and sends them to Diameter clients running the applicable active Gy sessions.

The Diameter Gateway uses ECE usage-request builders to construct request and response messages for Gy interface request types.

Constructing Usage Requests

Diameter Gateway constructs usage requests automatically provided the required request specification data (that pertains to the requests you want to send to the ECE charging servers) has been published to ECE from PDC by way of the Pricing Updater.

Diameter Gateway includes a usage-request builder for constructing usage requests (as well as different builders for building other requests ECE supports, such as balance query requests, and top-up requests, and so on). When you start Diameter Gateway nodes, the usage-request builder reads the request specification data and sends requests that adhere to the specifications.

When you perform network enrichment of the event definition for your events in PDC, you add network attributes for all event attributes in the event definition that apply to usage-request charging operations. Diameter Gateway uses the network specification and corresponding network attributes to dynamically populate the event attributes of ECE requests with the CCR AVP data of your incoming Diameter request.

You can have Diameter Gateway dynamically populate some fields using the event-attribute to network-attribute you map in PDC and you can have Diameter Gateway explicitly populate other fields using your own custom extension code (for example, when using the Pre-OCS extension, you can explicitly populate the ECE payload for fields using your Pre-OCS extension mechanism).

For information about mapping event attributes to network attributes, see the discussion about network enrichment of event definitions in *PDC User's Guide*.

Editing the Mediation Specification File

The mediation specification enables Diameter Gateway to associate each Diameter request with its respective usage-request builder. Diameter Gateway uses the mediation specification to derive which product and event type combination applies to an incoming Diameter request, allowing it to select the request specification that applies to the event to be rated.

Diameter Gateway reads the mediation specification to automatically create usage requests by dynamically mapping the values of the AVPs in the Diameter request to their corresponding payload items that are defined in request specifications.

You configure Diameter Gateway to base its selection of a request specification on the presence of any combination of these four AVPs in the Diameter request:

- Service-Context-Id AVP
- Service-Identifier AVP
- Rating-Group AVP
- Event-Timestamp AVP

From the preceding four AVP field values, Diameter Gateway derives the following three fields which uniquely identify the request specification to use for building the ECE request:

- ProductType

- EventType
- Version

You can configure Diameter Gateway to base its selection of a request specification on the presence of a custom AVP by using the Diameter Gateway Request-Received extension. You use the Request-Received extension to modify one of the four AVP values (Service-Context-Id AVP, Service-Identifier AVP, Rating-Group AVP, or Event-Timestamp AVP) in the Diameter request so that a different Diameter mediation mapping is produced (for a ProductType, EventType, and Version). For information about the Diameter Gateway Request-Received extension, see *BRM Elastic Charging Engine Extensions*.

To edit the mediation specification:

1. Create a mediation specification file or edit an existing one.

   A sample mediation specification file is available at *ECE_home*/**oceceserver/sample_data/config_data/specifications/ece_end2end/diameter_mediation.spec**.

   It is recommended to create only one mediation specification file to represent your mediation specification. You can have only one mediation specification loaded in the ECE cluster and the last one loaded takes precedence.

2. In the mediation specification file, add a row (in the table) for each event to be rated that specifies the following information:

   - Rating-Group AVP

     The Rating-Group AVP value sent in the Diameter message.

     Null is an acceptable value if the field is not expected to be present on the CCR.

   - Service-Context-Id AVP

     The Service-Context-Id AVP value sent in the Diameter message.

     Null is an acceptable value if the field is not expected to be present on the CCR.

   - Service-Identifier AVP

     The Service-Identifier AVP value sent in the Diameter message.

     Null is an acceptable value if the field is not expected to be present on the CCR.

   - ProductType

     The product type you have defined for the event in its associated request specification.

   - EventType

     The event type you have defined for the event in its associated request specification.

   - Version

     The version number of the request specification that you want to apply to the event.

   Define the Service-Identifier, Rating-Group, and Service-Context-Id for each request specification defined in the mediation table.

For each received Diameter request, Diameter Gateway correlates the Service-Context-Id, Service-Identifier, Rating-Group, and Event-Timestamp AVP values (that you defined in the mediation specification) to the usage-request builder that applies to the event to be rated (for the applicable version, product type, and event type that you defined in the event definition (request specification).

3. (Optional) In the **ValidFrom** field of the table, set a future date and time when you want Diameter Gateway to recognize a newly deployed request specification.

   For example, to have requests processed according to a new specification on April 16, 2015, you would enter:

   ```
   |   ValidFrom
    | "2015-04-16T12:01:01"
   ```

   You can also specify a time zone. For example,

   ```
   |   ValidFrom
    | "2015-04-16T12:01:01 PST"
   ```

   If a time zone is not sent, then the **ValidFrom** field is assumed as UTC.

4. Save the mediation specification file with a **.spec** suffix (for example, **diameter_mediation.spec**) into the directory where you save your configuration data.

5. Verify that the directory specified in the *ECE_home*/**ocecеserver/config/management/migration-configuration.xml** file is the directory where you save your configuration data.

6. Load the mediation specification file into the ECE server by running the **configLoader** utility:

   ```
   start configLoader
   ```

   The utility deploys your mediation specification to the ECE cluster. Any earlier mediation specification that was in the ECE cluster is overwritten.

   Any time you deploy a new version of a mediation specification (or request specification) into the repository, Diameter Gateway re-creates its in-memory usage-request builder map and begins using the mapping definitions (to send requests that adhere to the specifications) provided that the **ValidFrom** date is reached.

7. Perform a rolling restart of Diameter Gateway node instances.

Network Integration for Gy Balance Query Requests

This section provides information about network integration for balance query requests using Diameter Gateway.

Diameter Gateway uses custom AVPs for querying for remaining-balance customer data; these Oracle AVPs have an ORA- prefix.

For a balance query, the CC-Request-Type AVP in the CCR must be set to **4** (EVENT_REQUEST) and the Requested-Action AVP must be set to **5** (which is an undefined value in the 3GPP standard specification).

For information about the AVPs that Diameter Gateway requires for balance queries, see the section about Gy Balance Query Request/Response AVPs in *BRM Elastic Charging Engine Diameter Gateway Protocol Implementation Conformance Statement*.

For information about the data types for custom balance-query AVP fields, see the *ECE_home*/**oceceserver/config/diameter/dictionary_main.xml** file.

Network Integration for Gy Top-Up Requests

This section provides information about network integration for top-up requests using Diameter Gateway.

Diameter Gateway exposes a custom event request for top-up operations that does the following:

- Credits the specified balances, optionally setting valid-from and valid-to dates
- Optionally extends the validity of existing balances credited by the top-up
- Returns that the top-up succeeded or failed
- Returns updated balance information in the top-up response

Diameter Gateway uses custom AVPs for processing top-up requests; these Oracle AVPs have an ORA- prefix.

For a top-up, the CC-Request-Type AVP in the CCR must be set to **4** (EVENT_REQUEST) and the Requested-Action AVP must be set to **4** (which is an undefined value in the 3GPP standard specification).

Diameter Gateway uses custom AVPs for processing top-up requests; these Oracle AVPs have an ORA- prefix.

For information about the AVPs that Diameter Gateway requires for top-up requests, see the section about Gy Top-up Request/Response AVPs in *BRM Elastic Charging Engine Diameter Gateway Protocol Implementation Conformance Statement*.

For information about the data types for custom top-up-request AVP fields, see the *ECE_home*/**oceceserver/config/diameter/dictionary_main.xml** file.

Sending Multiple-Service Credit Control Requests from Diameter Gateway

Diameter Gateway supports Multiple-Service Credit Control (MSCC) requests in which a Diameter application performs credit control for multiples services within the same session.

Diameter Gateway only supports MSCC requests for usage request processing (all usage-request charging operations must be contained in an MSCC group even if the request contains only a single service).

Configuring Subscriber ID Lookups

When multiple subscriber ID types come in on the CCR message, not all subscription identifiers may be provisioned for your ECE implementation. For example, you might have separate online charging systems for handling different subscription services. You can now configure Diameter Gateway to look up customer public user identity information based only on the subscription identifier types for which you have provisioned your ECE implementation.

The possible customer subscription IDs that pertain to various customer services are defined by the Subscription-Id grouped AVP in the CCR message. Multiple subscription identifier types can be provided in the group's Subscription-Id-Type AVP field. The customer may have all of the following subscription identifiers for various networks on which the customer uses services: MSISDN, IMSI, SIP, NAI, PRIVATE.

For Diameter Gateway to look up customer public user identity information based on your subscription-identifier-type configuration, do the following:

1. Open your mediation specification file, **diameter_mediation.spec**.

   diameter_mediation.spec is in the directory specified by the **configObjectsDataDirectory** parameter in the *ECE_home*/**oceceserver/config/management/migration-configuration.xml** file.

2. Where multiple subscription types are expected in the CCR for the event to be rated, locate the row that specifies the rating group, service identifier, and service context ID for the event.

   Your subscription-identifier-type configuration is relevant for the combination of the given Service-Context-Id, Service-Identifier, and Rating-Group AVP values specified in the row for the event to be rated.

3. In the **Subscription-Id-Type** column for that row, enter the subscription-identifier-type configuration of your choice.

   For each received CCR Diameter message that includes multiple subscriber ID types, Diameter Gateway uses your subscription-identifier-type configuration for looking up the public user identity.

   The subscription-identifier-type configuration options are as follows:

   - For Diameter Gateway to perform a customer lookup by using *only one subscription ID type*, enter the full string name of that Subscription-Id-Type.

     Enter the name exactly as it is defined in the RFC specification (in capitals) and enclose it with quotation marks.

     The possible values you can enter in the **Subscription-Id-Type** column for the Subscription-Id-Type are as follows (values in bold):

     – **"END_USER_E164"**

       The identifier is in international E.164 format (for example, MSISDN), according to the ITU-T E.164 numbering plan defined in [E164] and [CE164].

     – **"END_USER_IMSI"**

       The identifier is in international IMSI format, according to the ITU-T E.212 numbering plan as defined in [E212] and [CE212].

     – **"END_USER_SIP_URI"**

       The identifier is in the form of a SIP URI, as defined in [SIP].

     – **"END_USER_NAI"**

       The identifier is in the form of a Network Access Identifier, as defined in [NAI].

     For example, if you enter **"END_USER_NAI"** in the **Subscription-Id-Type** column for that event, Diameter Gateway uses only the subscription identifier type END_USER_NAI to perform a customer public user identity lookup for those events and ignores all other subscription identifier types that may be included in the CCR for those events.

```
DiameterMediationTable {
    Service-Context-Id | Service-Identifier | Rating-Group | ProductType |
EventType | Version | Subscription-Id-Type | ValidFrom |
    "gy.service@example.com" | "1" | "10" | "VOICE" | "V_USAGE" | 1.0 |
"END_USER_NAI" | "2012-12-31T12:01:01 PST" |
```

```
    "gy.service@example.com" | "1" | "11" | "DATA" | "D_USAGE" | 1.0 |
"END_USER_IMSI" | "2012-12-31T12:01:01 PST" |
}
```

- For Diameter Gateway to perform a customer lookup by using a subscription ID type determined by *the order that you list subscription ID types* in the mediation specification, enter a comma-delimited list in the order that Diameter Gateway is to resolve the subscription ID type.

  The following example shows a comma-delimited list for which Diameter Gateway first looks up the public user identity of the customer based on the SIP URI subscription identifier, and secondly based on the IMSI. In this case Diameter Gateway ignores all other subscription ID types that may be included in the CCR.

```
DiameterMediationTable {
    Service-Context-Id | Service-Identifier | Rating-Group | ProductType |
EventType | Version | Subscription-Id-Type | ValidFrom |
    "gy.service@example.com" | "1" | "12" | "DATA" | "D_USAGE" | 1.0 |
"END_USER_SIP_URI, END_USER_IMSI" | "2012-12-31T12:01:01 PST" |
}
```

- For Diameter Gateway to perform a customer lookup by using the first subscription ID type that is read in the CCR (all other subscription ID types that may be included in the CCR are ignored), leave the **Subscription-Id-Type** column blank. This type of configuration is shown in the fourth row of the sample mediation specification.

```
DiameterMediationTable {
    Service-Context-Id | Service-Identifier | Rating-Group | ProductType |
EventType | Version | Subscription-Id-Type | ValidFrom |
    "gy.service@example.com" | "1" | "13" | "SMS" | "S_USAGE" | 1.0 | "" |
"2012-12-31T12:01:01 PST" |
}
```

4. Save the mediation specification file.
5. Run the **configLoader** utility to load your mediation specification in the ECE cluster:

   ```
   start configLoader
   ```

   When your mediation specification is loaded, the earlier version of your mediation specification (that was in the ECE cluster) is overwritten and Diameter Gateway uses the configuration of the newly loaded mediation specification.

Your subscription-identifier-type configuration is used by Diameter Gateway for all usage-charging operation types: Initiate, Update, Terminate, PriceEnquiry, BalanceQuery, TopUp, Debit, and Refund.

To troubleshoot issues that may occur with your subscription-identifier-type configuration, note the following points:

- If the subscription IDs cannot be resolved correctly with the values supplied in the **diameter_mediation.spec** file, errors are logged in the Diameter Gateway log files.
- In a DEBUGGING environment, you can enable DEBUG messages in the **log4j.properties** file as shown here:

```
log4j.logger.oracle.communication.brm.charging.ecegateway.diameter.framework=DE
BUG
log4j.logger.oracle.communication.brm.charging.ecegateway.diameter.gy=DEBUG
```

- If the subscription IDs cannot be found as configured in the **diameter_mediation.spec** file, you can expect an Errant result-code of DIAMETER_MISSING_AVP (5005) or DIAMETER_INVALID_AVP_VALUE (5004).

Adding Custom AVPs for Usage Requests

If you introduce custom AVPs (to introduce new ways for charging for your services), you define your custom AVPs in the *ECE_home*/**oceceserver/config/diameter/dictionary_custom.xml** file to define their data types.

After modifying the **dictionary_custom.xml** file, perform a rolling restart of Diameter Gateway nodes in your topology.

For AVPs that apply to usage-request processing, you add network attributes for all event attributes in the event definition so that they can be dynamically mapped to ECE payload fields by Diameter Gateway (for information about mapping event attributes to network attributes, see the discussion about network enrichment of event definitions in *PDC User's Guide*). You also put a path to your AVP field to an MSCC group block.

Configuring Notifications for Diameter Gateway

To enable Diameter Gateway to consume notifications from the ECE notification queue:

> **Note:** The following steps assume ECE is installed and required ECE post-installation tasks are completed. See *BRM Elastic Charging Engine Installation Guide* for information.

1. On the Oracle WebLogic server, verify that the ECE event notification queue (a JMS topic) has been created.

   See the discussion about ECE post-installation tasks for information about creating the ECE notification queue.

2. In ECE, verify that JMS credentials have been configured correctly so that ECE can publish notifications to the ECE notification queue.

   See "Configuring Notifications for Charging" for information about configuring JMS credentials in ECE.

3. Access the ECE MBeans:

   a. Log on to the driver machine.

   b. Start the ECE charging servers (if they are not started).

   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.

   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.

   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.

4. Expand **charging.notification**.

5. Expand **Attributes**.

6. Set the appropriate type of notification (such as top-up or advice of charge) to the appropriate value:

   - Configuring Top-up Notifications
   - Configuring Threshold Breach Notifications
   - Configuring Credit Limit Ceiling Breach Notifications
   - Configuring Credit Limit Floor Breach Notifications
   - Configuring Advice of Charge Notifications
   - Configuring Policy-Driven Charging Notifications

For information about each notification, see the NotificationConfigMBean of the BizParamConfigMBean package in *BRM Elastic Charging Engine Java API Reference*.

Configuring Alternative Diameter Peers for Notifications

Peer details are configured in Diameter Gateway to filter and route the notifications for the peers to which Diameter Gateway connects. Each Diameter Gateway instance listens to a registered peer. The connection is initiated from the peer to send the

respective notifications. If a Diameter Gateway instance sends a notification message to its peer and the peer is unavailable or the peer fails after receiving the notification message, the Diameter Gateway instance retains the notification messages and sends them to another available peer based on your alternative-peer configuration.

To configure alternative Diameter peers for notifications:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.
   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. To add a Diameter peer, expand **charging.diameterGatewayPeerConfigurations**.
3. Expand **Operations** and select **addPeer**.
4. Specify the value for the following parameter:
   - **peerName.** Enter the name of the Diameter peer.
5. Click the **addPeer** button.

   The peer is added to Diameter Gateway.
6. Expand **charging.diameterGatewayPeerConfigurations.***Peer_Name*, where *Peer_Name* is the name of the Diameter peer.
7. Expand **Attributes**.
8. For each peer connected to Diameter Gateway, configure alternative peers by specifying values for the following attribute:
   - **alternatePeerNames.** Enter the name of the alternative peer for the specified Diameter peer. You can specify two alternative peers for each Diameter peer. If the peer connected to Diameter Gateway fails or if it is unavailable, Diameter Gateway routes the notifications to the alternate peers configured for the peer.

Handling Requests When Charging Servers Are Unavailable

Diameter Gateway can be configured to use a *degraded mode* operating mode if the Elastic Charging Server (charging server nodes) become unavailable.

Diameter Gateway actively monitors the health of the Elastic Charging Server. If the Elastic Charging Server becomes unavailable (such as going below the charging-server health threshold), Diameter Gateway sends the DIAMETER_TOO_BUSY result code response to network requests.

See the discussion about configuring the charging-server health threshold and the discussion about checking the ongoing health of ECE charging server nodes in *BRM Elastic Charging Engine System Administrator's Guide* for more information.

5

Authentication and Accounting Using RADIUS Gateway

This chapter describes how to use Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE) RADIUS Gateway for authenticating access requests and processing accounting requests from RADIUS clients, such as terminal servers or network access servers (NAS).

Before reading this chapter, you should be familiar with ECE concepts and architecture. See the following chapters in *BRM Elastic Charging Engine Concepts*:

- ECE Overview
- ECE System Architecture

Before using RADIUS Gateway for authentication and accounting, you should be familiar with RFC-2865, RFC-2869, and RFC-2866. See *BRM Elastic Charge Engine RADIUS Gateway Protocol Implementation Conformance Statement* for information about the supported attribute-value pairs (AVPs) and messages.

Overview of Authentication and Accounting Using RADIUS Gateway

Important: Before using RADIUS Gateway for authentication or accounting, you must first implement ECE with Pricing Design Center (PDC) and Oracle Communications Billing and Revenue Management (BRM) to load event definitions from PDC into ECE and to load customer data and data keys from BRM into ECE. See "Integrating ECE with PDC" and "Integrating ECE with BRM" for instructions.

You use RADIUS Gateway for authenticating access requests and processing accounting requests for online charging when your customers use your terminal server or NAS to connect to ECE. RADIUS Gateway does the following when it receives a request from the RADIUS client:

1. Translates the request into an ECE request.
2. Submits the ECE request to the ECE server.
3. Receives the ECE response from the ECE server and translates it into a RADIUS message response.
4. Sends the RADIUS message response to the RADIUS client.

The following steps summarize how to set up ECE for authentication and accounting using RADIUS Gateway:

1. Add additional RADIUS Gateway nodes required for your topology and configure each instance.

   See the discussion about configuring the ECE system in *BRM Elastic Charging Engine System Administrator's Guide* for instructions on how to add RADIUS Gateway nodes and specifying RADIUS Gateway node properties.

2. (Optional) Customize the RADIUS data dictionary to include custom vendor-specific attributes.

   See the discussion about configuring the ECE system in *BRM Elastic Charging Engine System Administrator's Guide* for instructions on how to customize the RADIUS data dictionary.

3. Load your event definitions (which include request specification data) from PDC into ECE. See "Loading Pricing Data into ECE" for more information.

4. Load customer data and data keys from BRM into ECE. See "Loading Data from BRM into ECE" for more information.

5. Load the RADIUS mediation specification data.

   See the discussion about configuring the ECE system in *BRM Elastic Charging Engine System Administrator's Guide* for instructions on how to load the RADIUS mediation specification data.

6. Map RADIUS network attributes to event attributes.

   See the discussion about configuring the ECE system in *BRM Elastic Charging Engine System Administrator's Guide* for instructions on how to map the RADIUS network attributes to event attributes.

7. Start the RADIUS Gateway nodes.

   When the RADIUS Gateway nodes start, they automatically join the Coherence cluster gaining access to ECE caches and invocation services that it uses to send requests to ECE.

   See the discussion about starting and stopping ECE in *BRM Elastic Charging Engine System Administrator's Guide* for instructions on how to start RADIUS Gateway nodes.

About RADIUS Gateway Authentication

RADIUS Gateway queries the ECE server for the AVPs received in the access requests from RADIUS clients and authenticates the user based on the information retrieved from the ECE server.

RADIUS Gateway supports the following authentication mechanisms for querying the ECE server and authenticating access requests:

- **Password Authentication Protocol (PAP).** An authentication protocol that uses the user name and password to validate users. See "Authenticating Access Requests by Using PAP" for more information on how RADIUS Gateway performs the PAP authentication.
- **Challenge Handshake Authentication Protocol (CHAP).** An authentication protocol that authenticates a user to a network entity; for example, the Web. This protocol ensures that the server sends a challenge to the RADIUS client after the RADIUS client establishes a network connection to access the Web server. See "Authenticating Access Requests by Using CHAP" for more information on how RADIUS Gateway performs the CHAP authentication.

- **Extensible Authentication Protocol (EAP).** An authentication protocol that supports multiple authentication mechanisms for authenticating network access; for example, EAP-Message Digest 5 (MD5). See "Authenticating Access Requests by Using EAP" for more information on how RADIUS Gateway performs the EAP authentication.

Authenticating Access Requests by Using PAP

You use PAP to authenticate access requests based on the clear-text user name and user password. Only Access-Request requests are considered for PAP authentication: other messages are ignored. The PAP authentication is performed based on the User-Name and User-Password AVP values in the Access-Request request.

The PAP authentication process is as follows:

1. RADIUS Gateway receives the Access-Request request from the RADIUS client.
2. RADIUS Gateway authenticates the RADIUS client using the **sharedsecret** password that you provided during installation.

   Note: If RADIUS clients are represented by using an IP address range, ensure that all the RADIUS clients within the IP address range use the same **sharedsecret** password.

3. RADIUS Gateway translates the Access-Request request into an ECE query.
4. RADIUS Gateway sends the query with the User-Name AVP value to the ECE server to validate the user name.
5. If the user name is not found in the ECE server, the ECE server returns a failed response. RADIUS Gateway translates the failed response into the **Access-Reject** message and returns it to the RADIUS client.
6. If a match for the user name is found, RADIUS Gateway sends a query with the User-Password AVP value in the request to the ECE server to validate the user password.
7. The ECE server returns a password. If the password is encrypted, RADIUS Gateway decrypts the password using a data key loaded from BRM before validating the user password. The data key to be used is identified using the key ID in the password returned by the ECE server.
8. If the user password in the ECE server matches the User-Password AVP value in the query, the ECE server returns a success response. RADIUS Gateway translates the success response into the **Access-Accept** message and returns it to the RADIUS client.
9. If the user password does not match, the ECE server returns a failed response. RADIUS Gateway translates the failed response into the **Access-Reject** message and returns it to the RADIUS client.

Sample Access-Request Request for PAP Authentication

```
Code: Access-Request(1)
 Identifier: 0
 Length: 120
 Authenticator: 0x7D564C041FD183A4DBA037E03E3244F3
 User-Name: alias#1006
 User-Password: 0x41FD183A4335037E03E3244F3123123
```

```
NAS-IP-Address: 128.1.2.3
NAS-Port-Type: 1034
Service-Type: 2
```

Authenticating Access Requests by Using CHAP

You use CHAP to authenticate access requests by validating the identity of the RADIUS client using Access-Challenge messages. The CHAP authentication is performed based on the CHAP-Password and CHAP-Challenge AVP values in the Access-Request request. RADIUS Gateway uses the State AVP value in the Access-Request request or the **noOfChallenges** value that you configured in ECE to carry out the number of Access-Challenge messages for a given authentication session.

At any time in a given authentication session, RADIUS Gateway can also request the RADIUS client to send an Access-Challenge message. The CHAP authentication uses encrypted passwords for authentication and the Access-Challenge message can be requested for authentication by RADIUS Gateway at any time. Therefore, the CHAP authentication process is considered more secure than the PAP authentication process.

The CHAP authentication process is as follows:

1. The RADIUS client encrypts a clear-text user password by using the CHAP identifier and CHAP-Challenge AVP value and sends it in the CHAP-Password AVP in an Access-Request request.
2. RADIUS Gateway authenticates the RADIUS client using the **sharedsecret** password that you provided during installation.

   Note: If RADIUS clients are represented by using an IP address range, ensure that all the RADIUS clients within the IP address range use the same **sharedsecret** password.

3. RADIUS Gateway translates the Access-Request request into an ECE query.
4. RADIUS Gateway sends the query with the User-Name AVP value to the ECE server to validate the user name.
5. If the user name is not found in the ECE server, the ECE server returns a failed response. RADIUS Gateway translates the failed response into the **Access-Reject** message and returns it to the RADIUS client.
6. If a match for the user name is found, the ECE server returns the password associated with the user name. If the password is encrypted, RADIUS Gateway decrypts the password into a clear-text password using the data key loaded from BRM before validating the password. The data key to be used is identified using the key ID in the password returned by the ECE server.
7. RADIUS Gateway generates an MD5 hash value using the password, CHAP-Challenge AVP, and CHAP identifier (which is the first byte of the CHAP-Password AVP), and compares it with the CHAP-Password AVP value in the Access-Request request.
8. If the values do not match, RADIUS Gateway returns a failed response. RADIUS Gateway returns an Access-Reject message to the RADIUS client.
9. If the MD5 hash value and the CHAP-Password AVP value match, RADIUS Gateway returns a success response.
10. RADIUS Gateway sends an Access-Challenge message to the RADIUS client.

RADIUS Gateway uses the State AVP value in the Access-Request request to determine the number of Access-Challenge messages to be sent to the RADIUS client. For example, if the State AVP value is 0, RADIUS Gateway directly returns the Access-Accept message. If the State AVP value is 1, RADIUS Gateway sends only one Access-Challenge message to the RADIUS client.

11. If the State AVP value is null or if the value is not set, RADIUS Gateway calculates a random number between one and the maximum number of challenges configured in ECE and sends the Access-Challenge messages to the RADIUS client.
12. The RADIUS client responds with a value calculated through the MD5 hash function.
13. RADIUS Gateway checks the response against its calculation of the expected hash value.
14. If the values match, RADIUS Gateway repeats the Access-Challenge messages based on the State AVP value or the number calculated by RADIUS Gateway.
15. If the values do not match, RADIUS Gateway returns an Access-Reject message to the RADIUS client.

Sample Access-Request Request for CHAP Authentication

```
Code: Access-Request(1)
 Identifier: 0
 Length: 144
 Authenticator: 0x7D564C041FD183A4DBA037E03E3244F3
 CHAP-Password: 0x423423432412ADA123CC1123124123
 Chap-Challenge="0xFBFCE5676F94433682718EF97F8AB24900"
 NAS-IP-Address: 127.0.0.8
 State: 0
 NAS-Port-Type: 1816
 Service-Type: 2
```

Authenticating Access Requests by Using EAP

You use EAP to authenticate users using different authentication mechanisms. EAP includes password-based authentication methods and secure certificate-based authentication methods. The EAP authentication is performed based on the EAP-Message AVP value in the Access-Request request. RADIUS Gateway supports the following EAP authentication methods:

- **EAP-Tunneled Transport Layer Security (TTLS).** Authentication between RADIUS Gateway and the RADIUS client uses a secured connection in two phases. In the first phase, RADIUS Gateway and the RADIUS client exchange authentication certificates for establishing the secured connection. In the second phase, RADIUS Gateway authenticates the RADIUS client by using different authentication mechanisms, such as EAP-PAP, EAP-CHAP, and EAP-MD5. These authentication mechanisms use the attributes in the Access-Request request to perform the authentication. You can also configure a custom EAP authentication mechanism by using the RADIUS Gateway extension points. For information about RADIUS-Request processing extension points, see *BRM Elastic Charge Engine Extensions*.
- **EAP-Non-TTLS.** Authentication between RADIUS Gateway and the RADIUS client uses a configured list of EAP authentication mechanisms. The EAP-Non-TTLS authentication process is as follows:

1. RADIUS Gateway performs a standard check on the Access-Request request received from the RADIUS client.
2. RADIUS Gateway sends the EAP-Type AVP in the Access-Challenge message that contains the value corresponding to the first EAP type configured.
3. If the RADIUS client returns NAK, RADIUS Gateway sends the next EAP type in the configured list in the Access-Challenge message. RADIUS Gateway continues this process until the RADIUS client responds with an Access-Accept message or until the end of the configured list is reached. In that case, RADIUS Gateway sends an Access-Reject message.

RADIUS Gateway, by default, supports only the EAP-MD5 authentication mechanism in the EAP-Non-TTLS method. To use a different authentication method, use the CustomAuth and CustomEAPChallenge extension points. The CustomEAPChallenge extension point sends the initial EAP challenge to the RADIUS client. The CustomAuth extension point performs the authentication and returns the authentication result. Based on the result received, RADIUS Gateway sends the appropriate RADIUS response to the RADIUS client. For information about the CustomAuth and CustomEAPChallenge extension points, see *BRM Elastic Charge Engine Extensions*.

Sample Access-Request Request for EAP-MD5 Authentication

```
Code: Access-Request(1)
 Identifier: 0
 Length: 120
 Authenticator: 0x7D564C041FD183A4DBA037E03E3244F3
 User-Name: BOB
 NAS-IP-Address: 127.0.0.1
 Calling-Station-Id: 02-00-00-00-00-01
 Framed-MTU: 1400
 NAS-Port-Type: 19
 Connect-Info: CONNECT 11Mbps 802.11b
 Service-Type: 2
 EAP-Message: 0x0200000801424F42
 Message-Authenticator: 0x4FB1186DDA9643CED0CD13D59ECD9D4E
```

About RADIUS Gateway Accounting

RADIUS Gateway processes the accounting requests to track information about customer usage. For example, RADIUS Gateway tracks when customers log in to a network for using the services and when customers log out of the network. The information tracked by RADIUS Gateway is used for statistical purposes, network monitoring, and billing the customers based on the duration of the sessions or the type of services used.

To track customer usage information, RADIUS Gateway uses the network mapping definitions in ECE and maps the accounting requests received from the RADIUS clients to the usage requests with the corresponding operation types configured in ECE.

See the following topics for information on the different types of accounting requests received from the RADIUS clients:

- About Accounting-Start and Accounting-Stop Requests
- About Accounting-On and Accounting-Off Requests
- About Accounting-Interim-Update Requests

The RADIUS Gateway accounting process is as follows:

1. At the start of accounting or the start of a user session, the RADIUS client sends an accounting request to RADIUS Gateway. The Acct-Status-Type AVP value in the request indicates the start of accounting or start of a session for the user.
2. RADIUS Gateway processes the request and records the information as either an accounting-on record or an accounting-start record in ECE, based on the accounting request received.
3. RADIUS Gateway returns an Accounting-Response message to the RADIUS client to acknowledge the accounting-start or accounting-on request.
4. While the session is active, the RADIUS client sends periodic updates on the data usage to RADIUS Gateway through accounting requests with the Acct-Status-Type AVP set to Interim-Update.
5. RADIUS Gateway processes the requests and records the information as accounting-interim-update records in ECE.
6. RADIUS Gateway returns Accounting-Response messages to the RADIUS client to acknowledge the interim-update requests.
7. At the end of accounting or the end of the user session, the RADIUS client sends an accounting request that contains the Acct-Status-Type AVP value indicating the end of accounting or the end of the user session.
8. RADIUS Gateway processes the request and records the information as either an accounting-off record or an accounting-stop record in ECE, based on the accounting request received.
9. RADIUS Gateway returns an Accounting-Response message to the RADIUS client to acknowledge the accounting-off or accounting-stop request. At any time, if the RADIUS client does not receive an Accounting-Response message, it continues to send accounting requests until it receives a response.

About Accounting-Start and Accounting-Stop Requests

When a client is configured to use RADIUS accounting, the RADIUS client sends an Accounting-Start request, which specifies the start of a session for delivering a service, and an Accounting-Stop request, which specifies the end of the session that was started for delivering a service, to RADIUS Gateway. The Accounting-Start request describes the type of service being delivered and the user who is using that service. The Accounting-Stop request describes the type of service that was delivered. The Accounting-Stop request might also contain statistics, such as elapsed time, input and output octets, or input and output messages. The RADIUS client uses the Acct-Status-Type AVP to specify the start of a session and to specify the end of a session.

The following AVPs must be present in an Accounting-Start or Accounting-Stop request:

- Acct-Session-Id

> **Note:** The Accounting-Start and Accounting-Stop requests for a given session must have the same Acct-Session-Id AVP.

- Acct-Status-Type
- NAS-IP-Address or NAS-Identifier

- User-Name
- The AVP that you configured to derive the product type in ECE. For more information on the **avpName** and **vendorId** parameters, see the discussion about configuring RADIUS Gateway nodes in *BRM Elastic Charging Engine System Administrator's Guide*.

For an Accounting-Start request, the Acct-Status-Type AVP must be set to 1. When a RADIUS client sends the Accounting-Start request, the RADIUS client indicates that the user service session has started. When RADIUS Gateway receives the Accounting-Start request, RADIUS Gateway records the information contained in the request for billing purpose and returns the Accounting-Response message to the RADIUS client.

For an Accounting-Stop request, the Acct-Status-Type AVP must be set to 2. When a RADIUS client sends the Accounting-Stop request, the RADIUS client indicates that the user service session has ended. When RADIUS Gateway receives the Accounting-Stop request, RADIUS Gateway records the information contained in the request for billing purposes and returns the Accounting-Response message to the RADIUS client.

The RADIUS client continues to send the Accounting-Start or Accounting-Stop requests until it receives the Accounting-Response message.

Sample Accounting-Start Request for Accounting

```
[Code: Accounting-Request(4)
 Identifier: 0
 Length: 94
 Authenticator: 0x30303030303030303030303030303030
 Acct-Session-Id: 123456
 Acct-Status-Type: 1
 NAS-Identifier: telco.org
 User-Name: alias#5000
 Service-Type: 1]
Radius Response Packet
[Code: Accounting-Response(5)
 Identifier: 0
 Length: 20
 Authenticator: 0x00000000000000000000000000000000]
```

Sample Accounting-Stop Request for Accounting

```
[Code: Accounting-Request(4)
 Identifier: 1
 Length: 87
 Authenticator: 0x30303030303030303030303030303030
 Acct-Session-Id: 123456
 Acct-Status-Type: 2
 Acct-Input-Octets: 10
 Acct-Output-Octets: 18
 Acct-Session-Time: 200
 NAS-Identifier: telco.org
 User-Name: alias#5000
 Service-Type: 1]
Radius Response Packet
[Code: Accounting-Response(5)
 Identifier: 1
 Length: 20
 Authenticator: 0x00000000000000000000000000000000]
```

About Accounting-On and Accounting-Off Requests

When a client is configured to use RADIUS accounting, the RADIUS client sends an Accounting-On request, which specifies the start of accounting, and an Accounting-Off request, which specifies the end of accounting, to RADIUS Gateway. The RADIUS client uses the Acct-Status-Type AVP to specify the start of accounting and to specify the end of accounting.

The following AVPs must be present in an Accounting-On or Accounting-Off request:

- Acct-Status-Type
- NAS-IP-Address or NAS-Identifier
- The AVP that you configured to derive the product type in ECE. For more information on the **avpName** and **vendorId** parameters, see the discussion about configuring RADIUS Gateway nodes in *BRM Elastic Charging Engine System Administrator's Guide*.

For an Accounting-On request, the Acct-Status-Type AVP must be set to 7. When a RADIUS client sends the Accounting-On request, the RADIUS client indicates that it is ready for service. When RADIUS Gateway receives the Accounting-On request, RADIUS Gateway closes or terminates any open accounting session associated with that RADIUS client before the RADIUS client indicated it was ready for service.

For an Accounting-Off request, the Acct-Status-Type AVP must be set to 8. When a RADIUS client sends the Accounting-Off request, the RADIUS client indicates that it is going out of service. When RADIUS Gateway receives the Accounting-Off request, RADIUS Gateway closes or terminates all the open accounting sessions associated with that RADIUS client.

Sample Accounting-On Request for Accounting

```
[Code: Accounting-Request(4)
 Identifier: 4
 Length: 68
 Authenticator: 0x30303030303030303030303030303030
 Acct-Session-Id: 131
 Acct-Status-Type: 7
 NAS-Identifier: telco.org
 User-Name: alias#5000
 Service-Type: 1
[Code: Accounting-Response(5)
 Identifier: 4
 Length: 20
 Authenticator: 0x00000000000000000000000000000000]
```

Sample Accounting-Off Request for Accounting

```
[Code: Accounting-Request(4)
 Identifier: 5
 Length: 68
 Authenticator: 0x30303030303030303030303030303030
 Acct-Session-Id: 131
 Acct-Status-Type: 8
 NAS-Identifier: telco.org
 User-Name: alias#5000
 Service-Type: 1
Radius Response Packet
[Code: Accounting-Response(5)
 Identifier: 5
 Length: 20
```

```
 Authenticator: 0x00000000000000000000000000000000]
```

About Accounting-Interim-Update Requests

During a session, the RADIUS client periodically sends Accounting-Interim-Update requests, which specify the current session duration and current data usage, to RADIUS Gateway. The RADIUS client uses the Acct-Status-Type AVP to specify the interim update.

The following AVPs must be present in an Accounting-Interim-Update request:

- Acct-Session-Id

> **Note:** The Accounting-Interim-Update requests for a given session must have the same Acct-Session-Id AVP.

- Acct-Status-Type
- NAS-IP-Address or NAS-Identifier
- User-Name
- The AVP that you configured to derive the product type in ECE. For more information on the **avpName** and **vendorId** parameters, see the discussion about configuring RADIUS Gateway nodes in *BRM Elastic Charging Engine System Administrator's Guide*.

When periodic Accounting-Interim-Update requests are sent for the same active session, the identifier in each Accounting-Interim-Update request must be unique. If the identifier is the same, RADIUS Gateway considers only the first request received with that identifier and ignores other requests.

For an Accounting-Interim-Update request, the Acct-Status-Type AVP must be set to 3. When a RADIUS client sends the Accounting-Interim-Update request, the RADIUS client indicates that the session is active. When RADIUS Gateway receives the Accounting-Interim-Update request, RADIUS Gateway records the information contained in the request for billing purposes and returns the Accounting-Response message to the RADIUS client.

The RADIUS client continues to send Accounting-Interim-Update requests until it receives the Accounting-Response message.

Sample Accounting-Interim-Update Request for Accounting

```
[Code: Accounting-Request(4)
 Identifier: 0
 Length: 95
 Authenticator: 0x30303030303030303030303030303030
 Acct-Session-Id: 123456
 Acct-Status-Type: 3
 Acct-Input-Octets: 6
 Acct-Output-Octets: 10
 NAS-Identifier: telco.org
 User-Name: alias#5000
 Service-Type: 1]
Radius Response Packet
[Code: Accounting-Response(5)
 Identifier: 0
 Length: 20
```

```
Authenticator:0x00000000000000000000000000000000]
```

6

Integrating ECE with Offline Mediation Controller

This chapter provides information about integrating Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE) with Oracle Communications Offline Mediation Controller.

Before reading this chapter, you should be familiar with ECE concepts and architecture. See the following chapters in *BRM Elastic Charging Engine Concepts*:

- ECE Overview
- ECE System Architecture

This chapter assumes you are familiar with integration points between ECE and Offline Mediation Controller. See "About Integration Points for Offline Mediation Controller".

Overview of Integrating ECE with Offline Mediation Controller

Offline Mediation Controller integrates with ECE for offline network interaction. Offline Mediation Controller mediates offline charging requests from the network and passes them to the ECE system for charging by using the ECE charging API. Offline Mediation Controller uses a cartridge pack for connecting to the ECE system and submitting usage requests. For information about the Offline Mediation Controller ECE cartridge pack, see *Oracle Communications Offline Mediation Controller Elastic Charging Engine Cartridge Pack User Guide*.

The following steps summarize what you need to do to integrate ECE with Offline Mediation Controller:

> **Important:** Before integrating ECE with Offline Mediation Controller, you must first integrate ECE with PDC and BRM. See "Integrating ECE with PDC" and "Integrating ECE with BRM" for instructions.

1. Install the Offline Mediation Controller software.

   Offline Mediation Controller needs an installation of ECE to link the libraries and configuration for the client-side operations.

2. Install the ECE Cartridge Pack and create a node chain containing the ECE Distribution Cartridge.

When the ECE Distribution Cartridge starts, it automatically joins the Coherence cluster which gives Offline Mediation Controller access to ECE caches and invocation services which are required to send usage requests.

3. Send sample input for the preconfigured services to be sent to ECE as usage requests.

Sending Offline Charging Requests to ECE

To send offline charging requests to ECE, Offline Mediation Controller uses the Elastic Charging Client. The Elastic Charging Client (client library) must be installed on the same server where Offline Mediation Controller is installed and where the ECE Cartridge Pack is deployed. Offline Mediation Controller configuration files refer to the ECE libraries from the location where the Elastic Charging Client is installed.

When the Elastic Charging Client starts, it automatically joins the Coherence ECE cluster which gives Offline Mediation Controller access to ECE caches and invocation services which are required to send usage requests.

See *Oracle Communications Offline Mediation Controller Elastic Charging Engine Cartridge Pack User Guide* for instructions on how to connect Offline Mediation Controller to ECE and submit usage requests to ECE.

About Suspense Management for Offline Charging

For information about how suspense management is handled for offline charging, see *Oracle Communications Offline Mediation Controller Suspending and Recycling Call Data Records User's Guide*. ECE does not directly handle suspense management of CDRs for offline charging.

Part III

Implementing Charging Configurations

Part III describes how to implement business rules and charging-related configurations in Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE). It contains the following chapters:

7

Overview of Charging Configurations

This chapter is an overview of the usage-charging related configurations you must implement for Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE), such as charging business rules and runtime configurations.

Understanding Usage-Event Definitions

The usage events for which you charge your customers must be defined in the ECE system. You define chargeable usage events by defining the events in BRM and adding additional information required for processing usage requests in the event definition in Pricing Design Center (PDC). For more information, see the discussion about enriching event definitions in *PDC User's Guide*.

For information about how Diameter charging applications use request specification data in the event definition for building usage requests, see "Integrating Charging Clients with ECE".

Understanding Usage-Charging Business Rules

Most of the configuration that influences how usage requests are charged is done in PDC by defining your ECE pricing components. You can configure usage-charging business rules at the ECE system level to influence the usage-charging flows of specific charging operations. For example, you can configure how ECE will round a charging result for currency and non-currency resources.

For information about configuring usage-charging business rules, see "Configuring Business Rules for Charging".

Understanding Runtime Charging Options

You must set runtime configuration options that influence how ECE processes data when processing usage requests. For example, you must configure taxation. You can also configure ECE to publish notifications for sending data to other applications in the charging system. For information about configuring runtime charging options, see "Configuring Charging Runtime Options".

8

Configuring Notifications for Charging

This chapter provides instructions for configuring notifications published by Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE).

About Configuring Notifications

You can configure ECE to publish notifications.

All notifications are disabled by default. You must enable the publishing of notifications. See "Enabling External Notifications in ECE" for information.

You use the JMX interface, such as JConsole, to enable specific types of notifications. See "Configuring Notifications for Online Charging" for information.

The allowed values for configuring notification types are follows:

- **NONE**: No notification is sent.
- **ASYNCHRONOUS**: An asynchronous notification is sent.
- **PIGGYBACK**: Events are included as an in-session notification on the usage response message (send as a block on the usage response).
- **ASYNC_PIGGYBACK**: Both asynchronous and in-session notifications are sent.

Only Advice of Charge (AoC) service events and threshold breach service events can be configured for in-session notifications (PIGGYBACK). All other service events must be configured for ASYNCHRONOUS notifications when notifications are enabled for them.

For ECE to publish external notifications, configure the JMS credentials for the JMS server on which the notification queue (JMS topic) resides. See "Configuring JMS Credentials for Publishing External Notifications".

Enabling External Notifications in ECE

To publish external notifications of any type, ECE must be enabled for publishing external notifications.

To verify that ECE is enabled for publishing external notifications, do the following:

1. Open the *ECE_home*/**oceceserver/config/charging-cache-config.xml** file, where *ECE_home* is the directory in which you installed ECE.
2. For the ServiceContext module, verify that the following entry is used for the cache-store configuration:

```
<init-param>
  <param-name>cache-store</param-name>
```

```
<param-value>oracle.communication.brm.charging.notification.internal.coherence.
AsynchronousNotificationPublisher
                    </param-value>
```

ECE is enabled for publishing external notification.

3. Save the file.

Configuring JMS Credentials for Publishing External Notifications

For ECE to publish external notifications, it needs the JMS credentials for each JMS WebLogic server on which an ECE notification queue (JMS topic) resides in your system.

To configure the JMS credentials for publishing external notifications:

1. Open the *ECE_home*/**oceceserver/config/JMSConfiguration.xml** file.
2. Locate the **<MessagesConfigurations>** section.
3. Specify values for the parameters in the following **JMSDestination name** sections:
   - **NotificationQueue:** Read by the ECE charging nodes.
   - **BRMGatewayNotificationQueue:** Read by BRM Gateway.
   - **DiameterGatewayNotificationQueue:** Read by Diameter Gateway.

   Note: Do not change the value of the **JMSDestination name** parameter.

   Each **JMSDestination name** section contains the following parameters:

   - **HostName:** Deprecated but remains in ECE for backward compatibility. To specify the host names of the WebLogic servers on which the JMS topics reside, use the **ConnectionURL** parameter, which takes precedence over **HostName**.
   - **Port:** Deprecated but remains in ECE for backward compatibility. To specify the port numbers on which the WebLogic servers reside, use the **ConnectionURL** parameter, which takes precedence over **Port**.
   - **UserName:** Specify the user for logging on to the WebLogic server.

     This user must have write privileges for the JMS topic.
   - **Password:** Specify the password for logging on to the WebLogic server.

     When you install ECE, the password you enter is encrypted and stored in the keystore. If you change the password, you must run a utility to encrypt the new password before entering it here. See the discussion about encrypting new passwords in *BRM Elastic Charging Engine System Administrator's Guide*.
   - **ConnectionFactory:** Specify the connection factory used to create connections to the JMS topic on the WebLogic server to which ECE publishes notification events.

     You must also configure settings in Oracle WebLogic Server for the connection factory. See "Configuring a WebLogic Server Connection Factory for a JMS Topic".

- **QueueName:** Specify the JMS topic that holds the published external notification messages.
- **Protocol:** Deprecated but remains in ECE for backward compatibility. To specify the wire protocol used by the WebLogic servers, use the **ConnectionURL** parameter, which takes precedence over **Protocol**.
- **ConnectionURL:** Specify one or more URLs that applications can use to connect to the JMS WebLogic servers on which your ECE notification queues (JMS topics) reside.

  Note: When this parameter contains values, it takes precedence over the deprecated **HostName, Port,** and **Protocol** parameters.

  Use the following URL syntax:

  `[t3|t3s|http|https|iiop|iiops]://address[,address]. . .`

  where:

  – **t3**, **t3s**, **http**, **https**, **iiop**, or **iiops** is the wire protocol used.

  For a WebLogic server, use **t3**.

  – *address* is *hostlist*:*portlist*.

  – *hostlist* is *hostname*[,*hostname*.]

  – *hostname* is the name of a WebLogic server on which a JMS topic resides.

  – *portlist* is *portrange*[+*portrange*.]

  – *portrange* is *port*[-*port*.]

  – *port* is the port number on which the WebLogic server resides.

  Examples:

  ```
  t3://hostA:7001
  t3://hostA,hostB:7001-7002
  ```

  The preceding URL is equivalent to all the following URLs:

  ```
  t3://hostA,hostB:7001+7002
  t3://hostA:7001-7002,hostB:7001-7002
  t3://hostA:7001+7002,hostB:7001+7002
  t3://hostA:7001,hostA:7002,hostB:7001,hostB:7002
  ```

  Note: If multiple URLs are specified for a high-availability configuration, an application randomly selects one URL and then tries the others until one succeeds.

- **ConnectionRetryCount:** Specify the number of times a connection is retried after it fails.

  This applies only to clients that receive notifications from BRM.
- **ConnectionRetrySleepInterval:** Specify the number of milliseconds between connection retry attempts.

- **InitialContextFactory:** Specify the name of the initial connection factory used to create connections to the JMS topic queue on each WebLogic server to which ECE will publish notification events.
- **RequestTimeOut:** Specify the number of milliseconds in which requests to the WebLogic server must be completed before the operation times out.
- **KeyStorePassword:** If SSL is used to secure the ECE JMS queue connection, specify the password used to access the SSL keystore file.
- **KeyStoreLocation:** If SSL is used to secure the ECE JMS queue connection, specify the full path to the SSL keystore file.

4. Save and close the file.

Configuring a WebLogic Server Connection Factory for a JMS Topic

To configure a WebLogic server connection factory for a JMS topic:

1. Log on to a WebLogic server on which one of your JMS topics resides.
2. Log in to WebLogic Server Administration Console.
3. From the JMS modules list, select the connection factory that applies to the JMS topic used for ECE notifications.
4. In the **Transactions** tab, set the **Transaction Timeout** to **2147483647**.

For information about setting up a JMS topic on a WebLogic server, see the Oracle WebLogic Server documentation.

Considerations for Using a JMS Provider Other Than Oracle WebLogic Server

If you use a JMS provider other than Oracle WebLogic Server for publishing ECE notifications, do the following on the driver machine:

- Copy the other JMS provider's client JARs to the *ECE_home***/oceceserver/lib** directory.
- Rename the other JMS provider's JAR file to **wlthint3client.jar**.
- Update the *ECE_home***/oceceserver/config/JMSConfiguration.xml** file to specify the InitialContextFactory and protocol information of the other JMS provider.

Deploying JMS Configuration Setting Updates onto a Running System

After configuring JMS configuration settings on the driver machine, you can deploy the updates onto a running ECE system.

To deploy JMS configuration settings onto a running ECE system:

1. Log on to the driver machine.
2. Change directory to the *ECE_home***/oceceserver/bin** directory.
3. Start ECC.

   ```
   ./ecc
   ```

4. Run the **sync** command to deploy the ECE installation onto server machines:

   ```
   groovy:000> sync
   ```

   The **sync** command copies the relevant files of the ECE installation (which includes your JMS configuration setting updates), onto the server machines you have defined to be part of the ECE cluster.

5. Open the topology file (*ECE_home*/**oceceserver/config/eceTopology.conf**) and define the following nodes at the top of the file (before the charging server nodes):
   - Updater nodes
   - Gateway nodes
   - Formatter nodes

   These processes must be restarted before restarting the charging server nodes.
6. Run the **rollingUpgrade** command to perform a rolling restart of ECE nodes that are currently running on your topology.

   ```
   groovy:000> rollingUpgrade
   ```

   One by one (in the order they are listed in the topology file), each currently running node is brought down, updated, and joined back to the cluster.

   When you run the **rollingUpgrade** command with no parameters specified, all running nodes are updated (charging server nodes, data-loading utility nodes, data updater nodes, and so on) except for simulator nodes (nodes that have the role **simulator**).

Configuring Notifications for Online Charging

See the following topics to configure the notification that supports the charging business rule you want to configure:

- Configuring Top-up Notifications
- Configuring Threshold Breach Notifications
- Configuring Credit Limit Ceiling Breach Notifications
- Configuring Credit Limit Floor Breach Notifications
- Configuring Advice of Charge Notifications
- Configuring Policy-Driven Charging Notifications

For more information about configuring each notification, see the NotificationConfigMBean of the BizParamConfigMBean package in *BRM Elastic Charging Engine Java API Reference*.

Configuring Top-up Notifications

This section describes how to configure top-up notifications.

See the discussion about notifications in *BRM Elastic Charging Engine Concepts* for a description of top-up notifications.

To configure top-up notifications:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

The **eceTopology.conf** file also contains the host name and port number for the node.

e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.

2. Expand **charging.notification**.
3. Expand **Attributes**.
4. Specify one of the following values for the **topUpNotificationMode** attribute:
   - To disable the generation of top-up notifications, enter **NONE**.

     An external notification is *not* generated when the **TOP_UP_NOTIFICATION_EVENT** service event is created.
   - To generate an external notification and send it to a notification queue (JMS topic) when the **TOP_UP_NOTIFICATION_EVENT** service event is created, enter **ASYNCHRONOUS**.

Configuring Threshold Breach Notifications

This section describes how to configure threshold breach notifications.

See the discussion about notifications in *BRM Elastic Charging Engine Concepts* for a description of threshold breach notifications.

To configure threshold breach notifications:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.

   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.notification**.
3. Expand **Attributes**.
4. Specify one of the following values for the **thresholdBreachNotificationMode** attribute:
   - To disable the generation of threshold breach notifications, enter **NONE**.

     An external notification is *not* generated when the **THRESHOLD_BREACH_EVENT** service event is created.
   - To generate an external notification and send it to a notification queue (JMS topic) when the **THRESHOLD_BREACH_EVENT** service event is created, enter **ASYNCHRONOUS**.
   - To generate an in-session notification and return it in the usage response message when the **THRESHOLD_BREACH_EVENT** service event is created, enter **PIGGYBACK**.

- To generate both an external and in-session notification when the **THRESHOLD_BREACH_EVENT** service event is created, enter **ASYNC_PIGGYBACK**.

5. Expand **charging.server**.
6. Expand **Attributes**.
7. Specify one of the following values for the **thresholdBreachNotificationMode** attribute:
   - To generate the threshold breach notification only when the session is terminated, enter **ON_TERMINATE**.
   - To generate the threshold breach notification when the customer's balance breaches the credit threshold value during the session, enter **ONGOING**.

Configuring Credit Limit Ceiling Breach Notifications

This section describes how to configure credit limit ceiling breach notifications.

See the discussion about notifications in *BRM Elastic Charging Engine Concepts* for a description of credit limit ceiling breach notifications.

To configure credit limit ceiling breach notifications:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home***/ocecesever/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.

   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.notification**.
3. Expand **Attributes**.
4. Specify one of the following values for the **creditCeilingBreachNotificationMode** attribute:
   - To disable the generation of credit limit ceiling breach notifications, enter **NONE**.

     An external notification is *not* generated when the **CREDIT_CEILING_BREACH_EVENT** service event is created.
   - To generate an external notification and send it to a notification queue (JMS topic) when the **CREDIT_CEILING_BREACH_EVENT** service event is created, enter **ASYNCHRONOUS**.

Configuring Credit Limit Floor Breach Notifications

This section describes how to configure credit limit floor breach notifications.

See the discussion about notifications in *BRM Elastic Charging Engine Concepts* for a description of credit limit floor breach notifications.

To configure credit limit floor breach notifications:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.
   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.notification**.
3. Expand **Attributes**.
4. Specify one of the following values for the **creditFloorBreachNotificationMode** attribute:
   - To disable the generation of credit limit floor breach notifications, enter **NONE**.

     An external notification is *not* generated when the **CREDIT_FLOOR_BREACH_EVENT** service event is created.
   - To generate an external notification and send it to a notification queue (JMS topic) when the **CREDIT_FLOOR_BREACH_EVENT** service event is created, enter **ASYNCHRONOUS**.

Configuring Advice of Charge Notifications

This section describes how to configure advice-of-charge notifications.

See the chapter about notifications in *BRM Elastic Charging Engine Concepts* for a description of advice-of-charge notifications.

To configure advice-of-charge notifications:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.
   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.notification**.
3. Expand **Attributes**.
4. Specify one of the following values for the **adviceOfChargeNotificationMode** attribute:
   - To disable the generation of advice-of-charge notifications, enter **NONE**.

An external notification is *not* generated when the **ADVICE_OF_CHARGE_EVENT** service event is created.

- To generate an external notification and send it to a notification queue (JMS topic) when the **ADVICE_OF_CHARGE_EVENT** service event is created, enter **ASYNCHRONOUS**.
- To generate an in-session notification and return it in the usage response message when the **ADVICE_OF_CHARGE_EVENT** service event is created, enter **PIGGYBACK**.
- To generate both an external and in-session notification when the **ADVICE_OF_CHARGE_EVENT** service event is created, enter **ASYNC_PIGGYBACK**.

Configuring Policy-Driven Charging Notifications

This section describes how to configure notifications that are used for policy-driven charging. A client that interacts with the Policy and Charging Rules Function (PCRF), such as Diameter Gateway, consumes these notifications to retrieve data that it then sends to the PCRF.

See the chapter about notifications in *BRM Elastic Charging Engine Concepts* for a description of spending limit notifications.

To configure spending limit notifications:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**ocecerver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.
   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.notification**.
3. Expand **Attributes**.
4. Specify values for the following attributes:

   aggregatedSpendingLimitNotificationMode: Enter one of the following values:

   - To disable the generation of aggregate spending limit notifications, enter **NONE**.

     An external notification is *not* generated when the **AGGREGATED_SPENDING_LIMIT_EVENT** service event is created.
   - To generate an external notification and send it to a notification queue (JMS topic) when the **AGGREGATED_SPENDING_LIMIT_EVENT** service event is created, enter **ASYNCHRONOUS**.

   spendingLimitNotificationMode: Enter one of the following values:

   - To disable the generation of spending limit notifications, enter **NONE**.

     An external notification is *not* generated when the **SPENDING_LIMIT_EVENT** service event is created.

- To generate an external notification and send it to a notification queue (JMS topic) when the **SPENDING_LIMIT_EVENT** service event is created, enter **ASYNCHRONOUS**.

subscriberPreferenceUpdateNotificationMode: Enter one of the following values:

- To disable the generation of subscriber preference update notifications, enter **NONE**.

  An external notification is *not* generated when the **SUBSCRIBER_PREFERENCE_UPDATE_EVENT** service event is created.

- To generate an external notification and send it to a notification queue (JMS topic) when the **SUBSCRIBER_PREFERENCE_UPDATE_EVENT** service event is created, enter **ASYNCHRONOUS**.

Configuring Headers for External Notifications

Each external notification can include some general information in a header. You can configure the header to include the information (such as public user identity) that is needed by external systems for identifying and processing the notifications.

To configure the header for an external notification:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.

   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.notificationHeaderConfigurations**.
3. Expand **Operations**.
4. Select **setAttributes**.
5. Specify values for the following parameters:
   - **eventType.** Enter the service event type for the external notification. The following are the supported service event types:
     - THRESHOLD_BREACH_EVENT
     - AGGREGATED_THRESHOLD_BREACH_EVENT
     - ADVICE_OF_CHARGE_EVENT
     - CREDIT_FLOOR_BREACH_NOTIFICATION_EVENT
     - CREDIT_CEILING_BREACH_NOTIFICATION_EVENT
     - SPENDING_LIMIT_NOTIFICATION_EVENT
     - AGGREGATED_SPENDING_LIMIT_NOTIFICATION_EVENT
     - TOP_UP_NOTIFICATION_EVENT
     - RAR_NOTIFICATION_EVENT

- SUBSCRIBER_PREFERENCE_UPDATE_EVENT
- CUSTOM_EVENT
- COMMON_HEADER_ATTRIBUTES

To configure a header which is common to all the external notifications, enter COMMON_HEADER_ATTRIBUTES.

- **attributes.** Enter the name and *xpath* of the attribute that you want to add to the header in the following format:

```
AttributeName://xpath
```

where *xpath* is the path to access the attribute in the XSD file. For example, if you want to add PublicUserIdentity to the header and if it is the child element of the PublicUserIdentities element in the XSD file, then you enter the *xpath* as PublicUserIdentities/PublicUserIdentity. For *xpath* of the attributes used in external notifications, see the *ECE_home***/oceceserver/config/Notification.xsd** file.

Note: If you are configuring a header which is common to all the external notifications, enter only the attribute (for example, NotificationType, OperationType, or version) that is common to all the external notifications.

To add multiple attributes to the header, use the following format:

```
AttributeName1://xpath1,AttributeName2://xpath2,...
```

For example:

```
PublicUserIdentity://PublicUserIdentities/PublicUserIdentity,Operationtype:
//OperationType
```

6. Click the **setAttributes** button.

   The attributes are added to the notification header.

Configuring Notifications for BRM

For information about configuring notifications that ECE publishes for BRM, see Configuring Notifications for BRM.

Enriching External Notifications with Subscriber Preference Information

You can configure ECE to enrich external notifications with subscriber preference information.

BRM enables you to manage how each subscriber prefers to receive notifications from the network. For example, you can specify that a subscriber wants to receive notifications in French (Language preference) via SMS text messages (Channel preference). All subscriber preferences that are set for customers in BRM are also stored in ECE.

You can configure ECE to enrich the following types of ECE external notifications with subscriber preference information:

- Threshold breach notifications

- Aggregated threshold breach notifications
- Advice of Charge notifications
- Credit limit ceiling breach notifications
- Credit limit floor breach notifications
- Subscriber life cycle state transition notifications
- First usage validity initialization notifications

You can configure ECE to enrich each of the preceding external notifications with all subscriber preferences or with a subset of subscriber preferences.

If the same subscriber preference is defined as a customer-level preference and as a product-level (service-level) preference, ECE uses the product-level preference. If a subscriber preference is not specified at the product-level but is specified at the customer level, ECE uses the customer-level subscriber preference.

To configure ECE to enrich external notifications with subscriber preference information:

1. Get the list of subscriber preference names set in your BRM system.

   When configuring ECE to enrich the external notifications with a subset of subscriber preferences, you must enter the name of the subscriber preferences as you previously set it in your BRM system.

2. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.

   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
3. Expand **charging.notification**.
4. Expand **Attributes**.
5. Set the **subscriberPreferenceUpdateNotificationMode** attribute to **ASYNCHRONOUS**.
6. Select a notification type whose messages you want to enrich with subscriber preference information.
7. Set it to **ASYNCHRONOUS**.
8. Expand **Operations**.
9. Select **setNotificationEnrichmentConfig**.
10. Specify values for the following parameters:
    - **notificationEnrichType**: Enter **subscriberPreferences**.
    - **listOfEnrichNameValues**: Enter one of the following values:

      No value: (Default) External notifications are not enriched with subscriber preferences.

Individual subscriber preferences: External notifications are enriched with a subset of subscriber preferences. Enter the name of each preference, separated by commas. The names must match the preference names set in your BRM system.

ALL: External notifications are enriched with all the customer's subscriber preferences.

11. Click the **setNotificationEnrichmentConfig** button.

For each notification type that you have enabled to be enriched with subscriber preference information, ECE publishes subscriber preference information listed in the **SubscriberPreferences** block of the external notification messages.

The following is an example of the **SubscriberPreferences** block for a threshold breach notification that is enriched with the language subscriber preference of the customer.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Notification>
<NotificationType>THRESHOLD_BREACH_EVENT</NotificationType>
<PublicUserIdentities>
<PublicUserIdentity>6500000001</PublicUserIdentity>
</PublicUserIdentities>
<BalanceElementId>840</BalanceElementId>
<BalanceElementCode>USD</BalanceElementCode>
<CurrentBalance>-3.00</CurrentBalance>
<ThresholdAmount>-4</ThresholdAmount>
<ThresholdPercent>98.0</ThresholdPercent>
<BreachDirection>THRESHOLD_BREACH_UP</BreachDirection>
<DuplicateEvent>False</DuplicateEvent>
<SubscriberPreferences>
<SubscriberPreference PublicUserIdentity="6500000001:VOICE">
<SubscriberPreferencesInfo>
<PreferenceName>Language</PreferenceName>
<PreferenceValue>French</PreferenceValue>
</SubscriberPreferencesInfo>
</SubscriberPreference>
</SubscriberPreferences>
</Notification>
```

Sample Notification Payloads

This section shows samples of XML strings that ECE publishes into the JSM notification queue for different notification types.

THRESHOLD_BREACH_EVENT (Breach Direction Up)

The payload published for a threshold breach (breach direction up) uses the following format:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Notification>
    <CreditThresholdBreachNotification>
    <NotificationType>THRESHOLD_BREACH_EVENT</NotificationType>
<PublicUserIdentities>
    <PublicUserIdentity>123</PublicUserIdentity>
</PublicUserIdentities>
    <BalanceElementId>840</BalanceElementId>
    <BalanceElementCode>USD</BalanceElementCode>
    <CurrentBalance>-4.00</CurrentBalance>
    <ThresholdAmount>-4.5</ThresholdAmount>
```

```
    <ThresholdPercent>55.0</ThresholdPercent>
    <BreachDirection>THRESHOLD_BREACH_UP</BreachDirection>
    <AlertType>2</AlertType>
    <Reason>0x01</Reason>
    <OperationType>USAGE</OperationType>
    <SubOperationType>INITIATE</SubOperationType>
    </CreditThresholdBreachNotification>
</Notification>
```

THRESHOLD_BREACH_EVENT (Breach Direction Down)

The payload published for a threshold breach (breach direction down) uses the following format:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Notification>
    <CreditThresholdBreachNotification>
    <NotificationType>THRESHOLD_BREACH_EVENT</NotificationType>
<PublicUserIdentities>
    <PublicUserIdentity>123</PublicUserIdentity>
</PublicUserIdentities>
    <BalanceElementId>840</BalanceElementId>
    <BalanceElementCode>USD</BalanceElementCode>
    <CurrentBalance>-9.00</CurrentBalance>
    <ThresholdAmount>-8.0</ThresholdAmount>
    <ThresholdPercent>20.0</ThresholdPercent>
    <BreachDirection>THRESHOLD_BREACH_DOWN</BreachDirection>
    <AlertType>2</AlertType>
    <Reason>0x01</Reason>
    <OperationType>USAGE</OperationType>
    <SubOperationType>INITIATE</SubOperationType>
    </CreditThresholdBreachNotification>
</Notification>
```

AGGREGATED_THRESHOLD_BREACH_EVENT (Aggregated Based on Balance Element ID)

The payload published for an aggregated threshold breach (aggregated based on Balance Element ID) uses the following format:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Notification>
    <AggregatedCreditThresholdBreachNotification>
     <NotificationType>AGGREGATED_THRESHOLD_BREACH_EVENT</NotificationType>
       <PublicUserIdentities>
            <PublicUserIdentity>123</PublicUserIdentity>
       </PublicUserIdentities>
     <BalanceElementId>840</BalanceElementId>
     <BalanceElementCode>USD</BalanceElementCode>
     <CurrentBalance>-3.00</CurrentBalance>
     <ThresholdAmount>[-4.5, -3.5]</ThresholdAmount>
     <ThresholdPercent>[55.0, 65.0]</ThresholdPercent>
     <BreachDirection>THRESHOLD_BREACH_UP</BreachDirection>
     <OperationType>USAGE</OperationType>
     <SubOperationType>INITIATE</SubOperationType>
    </AggregatedCreditThresholdBreachNotification>
</Notification>
```

TOP_UP_NOTIFICATION_EVENT

The payload published for a top-up uses the following format:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Notification>
    <RARNotification>
       <NotificationType>TOP_UP_NOTIFICATION_EVENT</NotificationType>
       <PublicUserIdentities>
            <PublicUserIdentity>123</PublicUserIdentity>
            <PublicUserIdentity>456</PublicUserIdentity>
       </PublicUserIdentities>
       <ActiveSessions>
            <ActiveSessionId>SESSION1</ActiveSessionId>
            <ActiveSessionId>SESSION2</ActiveSessionId>
       </ActiveSessions>
       <ProductType>VOICE</ProductType>
       <ProductId>test</ProductId>
       <CustomerId>12345</CustomerId>
       <DuplicateEvent>True</DuplicateEvent>
       <OperationType>USAGE</OperationType>
       <SubOperationType>UPDATE</SubOperationType>
    </RARNotification>
</Notification>
```

CREDIT_CEILING_BREACH_EVENT

The payload published for a credit limit ceiling breach uses the following format:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Notification>
    <CreditCeilingBreachNotification>
    <NotificationType>CREDIT_CEILING_BREACH_EVENT</NotificationType>
    <BalanceElementId>840</BalanceElementId>
    <BalanceElementCode>USD</BalanceElementCode>
    <CurrentBalance>1.00</CurrentBalance>
    <CreditCeiling>0</CreditCeiling>
    <AlertType>2</AlertType>
    <Reason>0x01</Reason>
    <OperationType>USAGE</OperationType>
    <SubOperationType>INITIATE</SubOperationType>
   </CreditCeilingBreachNotification>
</Notification>
```

CREDIT_FLOOR_BREACH_EVENT

The payload published for a credit limit floor breach uses the following format:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Notification>
    <CreditFloorBreachNotification>
    <NotificationType>CREDIT_FLOOR_BREACH_EVENT</NotificationType>
    <BalanceElementId>840</BalanceElementId>
    <BalanceElementCode>USD</BalanceElementCode>
    <CurrentBalance>-1.00</CurrentBalance>
    <CreditFloor>0</CreditFloor>
    <AlertType>2</AlertType>
    <Reason>0x01</Reason>
    <OperationType>USAGE</OperationType>
    <SubOperationType>INITIATE</SubOperationType>
   </CreditFloorBreachNotification>
```

```
</Notification>
```

BILLING_NOTIFICATION_EVENT

The payload published for a billing notification uses the following format:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Notification>
    <BillingNotification>
       <NotificationType>BILLING_NOTIFICATION_EVENT</NotificationType>
       <CustomerId>12345</CustomerId>
       <BillingUnitId>2345</BillingUnitId>
       <ExternalReference>1</ExternalReference>
    </BillingNotification>
</Notification>
```

REPLENISH_POID_ID_NOTIFICATION_EVENT

The payload published for a replenish POID ID notification uses the following format:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Notification>
    <ReplenishPoidIdNotification>
       <NotificationType>REPLENISH_POID_ID_NOTIFICATION_EVENT</NotificationType>
       <SchemaName>1</SchemaName>
       <Quantity>10000</Quantity>
    </ReplenishPoidIdNotification>
</Notification>
```

SPENDING_LIMIT_NOTIFICATION

The payload published for a spending limit notification uses the following format:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Notification>
    <SpendingLimitNotification>
       <NotificationType>SPENDING_LIMIT_NOTIFICATION</NotificationType>
       <CustomerId>340876</CustomerId>
       <PublicUserIdentities>
           <PublicUserIdentity>9986068473</PublicUserIdentity>
           <PublicUserIdentity>login123</PublicUserIdentity>
       </PublicUserIdentities>
       <BalanceElementId>840</BalanceElementId>
       <BalanceElementCode>USD</BalanceElementCode>
       <CurrentBalance>2</CurrentBalance>
       <ConsumedReservation>3</ConsumedReservation>
       <Unit>MegaBytes</Unit>
       <Breaches>
          <OfferProfileName>Offer1</OfferProfileName>
          <LabelName>Fair Usage</LabelName>
          <StatusLabel>low qos</StatusLabel>
          <DeltaToNextThreshold>8</DeltaToNextThreshold>
       </Breaches>
       <DuplicateEvent>True</DuplicateEvent>
   </SpendingLimitNotification>
</Notification>
```

LIFE_CYCLE_TRANSITION_EVENT_TYPE

The payload published for a life cycle transition notification uses the following format:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Notification>
  <LifeCycleTransitionNotification>
     <NotificationType>LIFECYCLE_TRANSITION_NOTIFICATION_EVENT</NotificationType>
      <PublicUserIdentities>
            <PublicUserIdentity>0049100120</PublicUserIdentity>
      </PublicUserIdentities>
     <CustomerId>3135579</CustomerId>
     <ExternalReference>1</ExternalReference>
     <ProductId>3134811</ProductId>
     <ProductType>TelcoGsmTelephony</ProductType>
     <LifecycleState>103</LifecycleState>
     <ExpirationTime>1439653419867</ExpirationTime>
      <SubscriberPreferences>
     <SubscriberPreference PublicUserIdentity="316-20150813-143831-0-21484--1538921
12-slc06bui:TelcoGsmTelephony, 0049100120:TelcoGsmTelephony">
        <SubscriberPreferencesInfo>
           <PreferenceName>Language</PreferenceName>
           <PreferenceValue>French</PreferenceValue>
        </SubscriberPreferencesInfo>
       </SubscriberPreference>
     </SubscriberPreferences>
  </LifeCycleTransitionNotification>
</Notification>
```

EXTERNAL_TOP_UP_NOTIFICATION_EVENT

The payload published for a external top-up notification uses the following format:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Notification>
    <ExternalTopUpNotification>
     <NotificationType>EXTERNAL_TOP_UP_NOTIFICATION_EVENT</NotificationType>
     <PublicUserIdentities>
            <PublicUserIdentity>1000000</PublicUserIdentity>
     </PublicUserIdentities>
     <CustomerId>137826171</CustomerId>
     <ExternalReference>1</ExternalReference>
     <RequestTime>1325269800000</RequestTime>
     <Id>RECHARGE_ID1</Id>
     <BalanceImpact>
       <ProductId>137826171</ProductId>
       <ProductType>VOICE</ProductType>
       <BalanceItemImpact>
       <BalanceItemId>1</BalanceItemId>
       <BalanceElementCode>FSEC</BalanceElementCode>
       <Quantity>-10</Quantity>
       <ExtendValidityFlag>false</ExtendValidityFlag>
            <ValidFrom>1325269800000</ValidFrom>
            <ValidTo>1388514600000</ValidTo>
            </BalanceItemImpact
           </BalanceImpact>
    <SubscriberPreferences>
       <SubscriberPreference PublicUserIdentity="1000001:VOICE, 1000000:VOICE">
       <SubscriberPreferencesInfo>
         <PreferenceName>Language</PreferenceName>
         <PreferenceValue>French</PreferenceValue>
```

```
      </SubscriberPreferencesInfo>
     </SubscriberPreference>
   </SubscriberPreferences>
 </ExternalTopUpNotification>>
</Notification>
```

FIRST_USAGE_VALIDITY_EVENT_TYPE

The payload published for a first usage validity notification uses the following format:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Notification>
    <FirstUsageValidityNotification>
    <NotificationType>FIRST_USAGE_VALIDITY_INIT_NOTIFICATION_
EVENT</NotificationType>
    <CustomerId>12345</CustomerId>
    <ExternalReference>1</ExternalReference>
    <BalanceId>12345</BalanceId>
    <validity>
        <BalanceElementId>100025</BalanceElementId>
        <BalanceItemId>1</BalanceItemId>
        <ValidFrom>1325269800000</ValidFrom>
        <ValidTo>1388514600000</ValidTo>
    </validity>
     <SubscriberPreferences>
      <SubscriberPreference PublicUserIdentity="1000001:VOICE, 1000000:VOICE">
       <SubscriberPreferencesInfo>
          <PreferenceName>Language</PreferenceName>
          <PreferenceValue>French</PreferenceValue>
       </SubscriberPreferencesInfo>
      </SubscriberPreference>
     </SubscriberPreferences>
      </FirstUsageValidityNotification>
</Notification>
```

SUBSCRIBER_PREFERENCE_NOTIFICATION_EVENT

The payload published for creating a subscriber preference notification uses the following format:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Notification>
    <CreateSubscriberPreference>
     <NotificationType>SUBSCRIBER_PREFERENCE_NOTIFICATION_EVENT</NotificationType>
     <CustomerId>340876</CustomerId>
     <ProductInfo>
      <ProductId>12345</ProductId>
      <PublicUserIdentities>
       <PublicUserIdentity>9886753556</PublicUserIdentity>
       <PublicUserIdentity>login</PublicUserIdentity>
      </PublicUserIdentities>
     </ProductInfo>
     <SubscriberPreferencesInfo>
      <PreferenceName>Language</PreferenceName>
      <PreferenceValue>English</PreferenceValue>
     </SubscriberPreferencesInfo>
     <SubscriberPreferencesInfo>
      <PreferenceName>Channel</PreferenceName>
      <PreferenceValue>Email</PreferenceValue>
     </SubscriberPreferencesInfo>
```

```
    </CreateSubscriberPreference>
</Notification>
```

The payload published for modifying a subscriber preference notification uses the following format:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Notification>
    <ModifySubscriberPreference>
     <NotificationType>SUBSCRIBER_PREFERENCE_NOTIFICATION_EVENT</NotificationType>
     <CustomerId>customer1</CustomerId>
     <ProductInfo>
      <ProductId>12345</ProductId>
      <PublicUserIdentities>
       <PublicUserIdentity>9886753556</PublicUserIdentity>
       <PublicUserIdentity>login</PublicUserIdentity>
      </PublicUserIdentities>
     </ProductInfo>
     <SubscriberPreferencesInfo>
      <PreferenceName>Language</PreferenceName>
      <PreferenceValue>English</PreferenceValue>
     </SubscriberPreferencesInfo>
     <SubscriberPreferencesInfo>
      <PreferenceName>Channel</PreferenceName>
      <PreferenceValue>Email</PreferenceValue>
     </SubscriberPreferencesInfo>
    </ModifySubscriberPreference>
</Notification>
```

The payload published for deleting a subscriber preference notification uses the following format:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Notification>
    <DeleteSubscriberPreference>
     <NotificationType>SUBSCRIBER_PREFERENCE_NOTIFICATION_EVENT</NotificationType>
     <CustomerId>customer1</CustomerId>
     <ProductInfo>
      <ProductId>12345</ProductId>
      <PublicUserIdentities>
       <PublicUserIdentity>9886753556</PublicUserIdentity>
       <PublicUserIdentity>login</PublicUserIdentity>
      </PublicUserIdentities>
     </ProductInfo>
     <SubscriberPreferencesInfo>
      <PreferenceName>Language</PreferenceName>
      <PreferenceValue>English</PreferenceValue>
     </SubscriberPreferencesInfo>
     <SubscriberPreferencesInfo>
      <PreferenceName>Channel</PreferenceName>
      <PreferenceValue>Email</PreferenceValue>
     </SubscriberPreferencesInfo>
    </DeleteSubscriberPreference>
</Notification>
```

ADVICE_OF_CHARGE_NOTIFICATION_EVENT_TYPE

The payload published for advice of charge notification uses the following format:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Notification version="3.0.0.0.0">
```

```
  <AdviceOfChargeNotification>
     <NotificationType>ADVICE_OF_CHARGE_NOTIFICATION_EVENT</NotificationType>
      <PublicUserIdentity>CUSTOMER1</PublicUserIdentity>
      <Consumptions>
     <BalanceElementId>840</BalanceElementId>
     <ConsumptionQuantity>10.0</ConsumptionQuantity>
     <ConsumptionUnit>Money{cur=USD}</ConsumptionUnit>
      </Consumptions>
      <Grants>
     <BalanceElementId>840</BalanceElementId>
     <GrantQuantity>1.0</GrantQuantity>
     <GrantUnit>Money{cur=USD}</GrantUnit>
      </Grants>
      <CurrentBalance>
     <BalanceElementId>840</BalanceElementId>
     <CurrentBalanceQuantity>100.0</CurrentBalanceQuantity>
     <CurrentBalanceUnit>Money{cur=USD}</CurrentBalanceUnit>
      </CurrentBalance>
      <ChargeInfo>
     <OperationType>USAGE</OperationType>
     <SubOperationType>UPDATE</SubOperationType>
      </ChargeInfo>
  </AdviceOfChargeNotification>
</Notification>
```

9

Configuring Business Rules for Charging

This chapter provides instructions for configuring usage-charging business rules for Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE).

About Usage-Charging Business Rules

This section describes the different usage-charging business rules you can configure.

For information about configuring usage-charging business rules, see "Configuring Usage-Charging Business Rules".

About Reservation Validity

ECE authorizes and reserves a balance or *active reservation* for a session request. ECE bases the active reservation on the requested service units of the session request (Initiate or Update request types). ECE sends the validity time for the active reservation or *reservation validity* back to the network mediation client. The reservation validity specifies how long a session can continue before the client must ask for a reauthorization of resources for further usage.

ECE supports a configurable *reservation expiration*, which specifies how long a session can continue before the client must report the consumed usage to ECE.

ECE uses the used service units to record the reserved balance as a consumed reservation. If the network mediation client does not communicate the used service units within the reservation expiration, ECE considers the reserved balances to be available balances for subsequent session requests. The reserved balances that are available are cleaned up as part of administrative processes when the session terminates.

Setting the reservation expiration time too low takes network activity to report usage. Setting the value too high increases the risk of revenue leakage if the customer uses up the balance before the reservation expiration time expires.

You configure the reservation validity as a systemwide setting.

ECE sends the reservation expiration and reservation validity information in the usage response.

The validity time is set for each session and gets reset upon each interim request received for the session.

After the validity time for a session expires, any resources reserved are released and become available to other active sessions for the same product.

For information about configuring reservation validity, see "Configuring Reservation Validity and Expiration".

About Reservation Quota

When ECE receives usage requests in which the REQUESTED_UNITS block is not set (the Requested-Service-Units AVP value is missing), ECE uses the systemwide quota you specify. For each product and rateable usage metric (RUM) combination, you configure a systemwide *initial* quota and a systemwide *incremental* quota for reservation. For usage requests that have an Initiate operation type, ECE uses the systemwide initial quota for the reservation. For usage requests that have an Update operation type, ECE uses the systemwide incremental quota for the reservation.

For information about configuring reservation quota, see "Configuring Reservation Quota for Products".

For information about setting the reservation quota when using incremental or cumulative accounting, see "Using Incremental or Cumulative Accounting for Usage Requests".

About Minimum Quantity for Reservation

You can configure a minimum quantity for reservation for charging events.

Some charging events cannot be charged in fragments. For example, you cannot charge for half of an SMS message. In this case, you would set a minimum quantity reservation of **1** for charging an SMS event.

If the customer does not have enough balance to reserve the minimum quantity of the charging event, ECE sends a usage response for the request as INSUFFICIENT_RATED_QUANTITY.

For information about configuring minimum quantity reservation, see "Configuring a Minimum Quantity for Reservation".

About Advice of Promotion

ECE provides the capability to provide Advice of Promotion (AoP) information enabling the operator to notify the customer that a better price could be obtained for the service they are about to use. ECE provides the AoP information as defined by 3GPP (TS22.086). The operator can choose to use this information in whatever way is relevant to the specific service being offered. For example, a network operator can send the AoP information in an Interactive Voice Response (IVR) pre-call announcement for a Voice service.

The AoP function can be used to notify a customer of a preferential price that would be available within the configured time window. If a reduced rate is available within this window, the AoP will be returned with an indication of the price to which the new rate would be applicable.

To support AoP, ECE determines whether a better rate for using a service is available near the time that the customer's usage request is received. ECE relays that information back to the network mediation system, which passes the message on to the customer.

ECE implements AoP as follows:

1. A customer makes a request to initiate a session, to debit a specific or calculated amount of a resource, or to generate a price estimation for using a resource.

2. The ECE charging server calculates the charge for the request.
3. If AoP is enabled, ECE adds a time offset to the start and end time of the request and recalculates the charge using the offset time period (the new start and end time).

   You configure how much of a time offset to use. See the discussion about usage-charging configuration in *BRM Elastic Charging Engine System Administrator's Guide* for instructions.
4. If the recalculated charge is less expensive for the customer, ECE sends the information about potential savings back to the network mediation system in the usage response.

ECE applies AoP when AoP is configured at the ECE system level. Configure AoP at the system level by using the configuration service. See "Configuring Advice of Promotion" for information about configuring AoP.

Note the following details about AoP:

- AoP is a systemwide configuration (it is not configured on a per-charge offer basis).
- The default configuration of AoP gives advice based on time.
- When applying AoP, ECE uses charge offers and discount offers that are eligible when the request is received to recompute the charge for the offset time period. If a different charge offer or discount offer applies to the future offset time period, AoP may advise a promotion where there is none or may not advise a promotion when a promotion is available.

When using AoP, ensure that your rate plans have tiered consumption configured accurately so as to prevent a credit breach of noncurrency balance elements.

About Rounding Charging Results

By default, ECE uses the rounding rules configured in Pricing Design Center (PDC) for a currency or noncurrency resource to round the balance impact amount for processing stages like charging, discounting, and taxation. Rounding rules can be different for each processing stage. For information on configuring the rounding rules for resources in PDC, see the PDC Help.

However, you can also configure rounding in ECE for a currency or noncurrency resource by specifying rules for how the ECE rounds the balance impact amount for processing stage like charging, discounting, and taxation. The rounding rules you specify are applied at the system level so the same rule is applied across all processing stages. The rounding is also applied to reverse-rating calculations.

If a rounding rule is not configured in PDC for a resource, ECE uses the rules configured at the system level to round the balance impact amount. For information about resource rounding, see the discussion about resource rounding in *BRM Setting Up Pricing and Rating*.

The rules you can specify for rounding are as follows:

- Scale

  You can specify a rule for how many digits to the right of the decimal point to allow. The default scale is **2**.
- Rounding mode

You can specify a rule for the rounding behavior according to the Java math rounding enum. The default is **HALF_UP**.

Go to the Java SE technical documentation web site for information about using the Java math rounding enum:

`http://docs.oracle.com/javase/6/docs/api/java/math/RoundingMode.html`

For information about configuring rounding for currency and noncurrency resources, see "Configuring Rounding for a Currency Resource" and "Configuring Rounding for a Noncurrency Resource".

Example of Currency Rounding for a Charge

If you allow two digits to the right of the decimal point and you round down towards zero (**DOWN** rounding mode), ECE takes a calculated charge of 0.509 USD and rounds it to 0.50 USDs.

Example of Noncurrency Rounding for a Charge

If you allow zero digits to the right of the decimal point and you round towards positive infinity (**UP** rounding mode), ECE takes a charge of 0.509 bonus point and rounds the value to 1 bonus point.

Examples of Currency Rounding for Discounts

If you allow zero digits to the right of the decimal point and you round down towards zero (**DOWN** rounding mode), ECE takes a discount of -2.5 USD and rounds the value to -2 USD.

If you allow zero digits to the right of the decimal point and you round towards negative infinity (**FLOOR** rounding mode), ECE takes a discount of -2.5 USD and rounds the value to -3 USD.

If you allow two digits to the right of the decimal point and you round down towards zero (**DOWN** rounding mode), ECE takes a discount of -0.075 USD and rounds the value to -0.07 USD.

About Reverse Rating When Rating Is Based on Multiple RUMs

When ECE applies reverse rating for a service in which events are rated by using multiple RUMs (for example, when applying reverse rating for a session event where both Occurrence and Duration are RUMs by which the event is measured), fractional values of the authorized resource quantity may result. You can enable ECE to round up the authorized resource quantities to the nearest whole number.

Rounding up the authorized resource quantity may result in customers exceeding their credit limits.

If your business requires that your customers must be able to use *all* of their balances, configure ECE to round up the authorized resource quantity.

For information about configuring rounding for reverse rating when rating is based on multiple RUMs, see "Configuring Rounding for Reverse Rating on Multiple RUMs".

About a Tolerance for Policy-Tier Threshold Breaches for Policy-Driven Charging

Unlike credit limits, which prevent revenue loss and are strictly enforced by Oracle Communications Billing and Revenue Management (BRM), policy tier thresholds must be crossed to trigger implementation of business rules, such as reduced quality of service (QoS) for subscribers who download an excessive amount of data.

In both cases, BRM calculates authorization amounts as a subscriber nears a limit or threshold:

- For credit limits, BRM authorizes the remaining balance and sends a final unit indicator (FUI), which makes the entire remaining balance available for use. If the balance is not topped up before it is completely consumed, the session is terminated.
- For policy tier thresholds, BRM cannot authorize an amount above the threshold, even if the subscriber's credit balances are sufficient to cover the charges. Instead, BRM authorizes the remaining balance up to the policy threshold but does not send an FUI. Therefore, only about 80 percent of the remaining balance is made available for use. The session ends when the remaining balance becomes so small that the service can no longer be supported.

To enable subscribers to continue using a service as they near a policy tier threshold, you must configure a breach tolerance for the threshold. A *breach tolerance* is the amount of usage exceeding the threshold for which a customer can be authorized. It permits BRM to authorize a usage amount that crosses the policy tier threshold. When the threshold is crossed, the service continues under a new business rule, such as lower QoS for larger download totals.

For example, suppose the network sends a usage request for 200 MB, but adding that to a subscriber's current 1.9 GB policy counter balance will cause the balance to breach a 2 GB policy tier threshold. In this case, BRM does one of the following:

- **Without Breach Tolerance:** If a breach tolerance is not configured, BRM makes only about 80 MB available to prevent the usage from exceeding the policy tier threshold. When usage reduces the 80 MB balance to the point that the remaining balance cannot support the service, the session ends.
- **With Breach Tolerance:** If a breach tolerance of 100 or more MB is configured, BRM authorizes the entire 200 MB request. This enables the subscriber's usage to cross the 2 GB policy tier threshold by 100 MB. As soon as the policy tier threshold is crossed, a change in the quality of service is triggered, and the service continues under the new policy.

You can set a breach tolerance for each balance element used in a policy counter. You decide what tolerance value is appropriate for your business needs.

For information about configuring a tolerance for credit limit breaches, see "Configuring a Tolerance for Policy-Tier Threshold Breaches for Policy-Driven Charging".

Configuring Usage-Charging Business Rules

This section provides instructions for configuring usage-charging business rules in the ECE runtime system.

For information about the different usage-charging business rules, see "About Usage-Charging Business Rules".

To configure usage-charging business rules, see the following topics:

- Configuring Reservation Validity and Expiration
- Configuring Reservation Quota for Products
- Configuring a Minimum Quantity for Reservation
- Configuring Advice of Promotion

- Configuring Rounding for a Currency Resource
- Configuring Rounding for a Noncurrency Resource
- Configuring Rounding for Reverse Rating on Multiple RUMs
- Configuring a Tolerance for Policy-Tier Threshold Breaches for Policy-Driven Charging
- Configuring Systemwide Consumption Rules for Balances
- Defining Systemwide Credit Profiles
- Redirecting a Subscriber Session to a Service Portal
- Configuring ECE to Generate Midsession Rated Events

For more information about each usage-charging attribute, see the **BizParamConfigMBean** package in *BRM Elastic Charging Engine Java API Reference*.

Configuring Reservation Validity and Expiration

For information about reservation validity, see "About Reservation Validity".

To configure the reservation validity and expiration:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.

   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.reservationConfig**.
3. Expand **Attributes**.
4. Specify values for the following attributes:
   - **validityTime**: Enter the amount of time, in seconds, to use as the reservation validity. This specifies how long a session can continue before the client must ask for a reauthorization. The default value is **3600** (one hour).
   - **reservationDuration**: Enter the amount of time, in seconds, to use as the reservation expiration. This specifies how long a session can continue before the client must report the consumed usage to ECE. The default value is **3600** (one hour).

For detailed information, see ReservationConfigMBean in the **BizParamConfigMBean** package of *BRM Elastic Charging Engine Java API Reference*.

Configuring Reservation Quota for Products

For information about reservation quota, see "About Reservation Quota".

To configure the reservation quota for products:

1. Access the ECE MBeans:

a. Log on to the driver machine.

b. Start the ECE charging servers (if they are not started).

c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.

d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**ocecerver/config/eceTopology.conf** file.

The **eceTopology.conf** file also contains the host name and port number for the node.

e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.

2. Expand **charging.reservationConfig**.

3. Expand **Operations**.

4. For *each product* that you offer, do the following:

a. Select **setDefaultReservationQuota**.

b. Specify values for the following parameters:

productType: Enter the name of the product defined in the ECE request specification data (for example, VOICE or SMS).

rum: Enter the *name* of the attribute defined in the ECE request specification data.

Important: Though the parameter name is **rum**, its value must be the *attribute name* specified in the REQUESTED_UNITS block of the request specification data, not the RUM name. For example, if you send attribute **INPUT_VOLUME** in the usage request, enter **INPUT_VOLUME** as the **rum** attribute's value.

---

initialQuota: Enter the initial quota for this product-RUM combination. The value must be decimal-compliant (Java BigDecimal). ECE uses this value to populate the REQUESTED_UNITS blocks of all Initiate-type usage requests whose Requested-Service-Units AVP value is missing.

incrementalQuota: Enter the incremental quota for this product-RUM combination. The value must be decimal-compliant (Java BigDecimal). ECE uses this value to populate the REQUESTED_UNITS blocks of all Update-type usage requests whose Requested-Service-Units AVP value is missing.

unit: Enter the unit of measurement for the quota, such as seconds, minutes, events, or megabytes.

c. Click the **setDefaultReservationQuota** button.

Configuring a Minimum Quantity for Reservation

For information about minimum quantity for reservation, see "About Minimum Quantity for Reservation".

To configure the minimum quantity for reservation:

1. Access the ECE MBeans:

a. Log on to the driver machine.

b. Start the ECE charging servers (if they are not started).

c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.

d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

The **eceTopology.conf** file also contains the host name and port number for the node.

e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.

2. Expand **charging.reservationConfig**.
3. Expand **Operations**.
4. Select **setMinAuthorizedQuota**.
5. Specify values for the following parameters:
   - **productType**: Enter the name of the product for which you are setting a minimum quantity reservation. Enter the name as it is defined in the ECE request specification data (for example, VOICE or SMS).
   - **rum**: Enter the *name* of the attribute defined in the ECE request specification data.

     Important: Though the parameter name is **rum**, its value must be the *attribute name* specified in the REQUESTED_UNITS block of the request specification data, not the rateable usage metric (RUM) name. For example, if you send attribute **INPUT_VOLUME** in the usage request, enter **INPUT_VOLUME** as the **rum** attribute's value.

   - **minAuthorizeQuota**: Enter the minimum amount of the specified unit that can be reserved for this product-RUM combination.
   - **unit:** Enter the unit of measurement for the quota, such as seconds, minutes, events, or megabytes.
6. Click the **setMinAuthorizedQuota** button.

Configuring Advice of Promotion

You configure Advice of Promotion (AoP) as a systemwide setting.

See "About Advice of Promotion" for information about how ECE supports AoP.

To configure Advice of Promotion:

1. Access the ECE MBeans:

a. Log on to the driver machine.

b. Start the ECE charging servers (if they are not started).

c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.

d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

The **eceTopology.conf** file also contains the host name and port number for the node.

e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.

2. Expand **charging.server**.

3. Expand **Attributes**.
4. Specify values for the following attributes:
   - **aopEnabled**: Enter **true** to enable AoP or **false** to disable AoP.
   - **aopVariance**: Enter an amount of time in the ISO 8601 duration format (for example, **PT10M**, which specifies 10 minutes).

     ECE uses the time you specify to offset the start and end times of the request and recalculate the charge for the offset period.

     For more information about the duration format, see the ISO 8601 documentation.

Configuring Rounding for a Currency Resource

You can configure rounding for a currency resource by specifying rules for how the rating engine rounds the balance impact amounts it calculates. See "About Rounding Charging Results" for information about rounding rules.

The rules you specify are applied as a systemwide configuration for rounding the balance impact amounts of all charges, discounts, and chargeshares. The rounding is also applied to all reverse rating calculations and taxation calculations.

To configure rounding for a currency resource:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.
   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.server**.
3. Expand **Attributes**.
4. Specify values for the following attributes:
   - **currencyScale**: Enter the number of digits you allow to the right of the decimal point for a calculated impact amount.

     For example, enter **2** if you allow two digits to the right of the decimal point.

     The default is **2**.
   - **currencyRoundingMode**: Enter the rounding mode that determines the rounding behavior by entering the string representation of the Java math rounding enum.

     For more information, see the Java SE technical documentation web site:

     `http://docs.oracle.com/javase/6/docs/api/java/math/RoundingMode.html`

     For example, enter **UP** to round up away from zero or **DOWN** to round down towards zero.

The default is **HALF_UP**.

Configuring Rounding for a Noncurrency Resource

You can configure rounding for a noncurrency resource by specifying rules for how the rating engine rounds the balance impact amounts it calculates. See "About Rounding Charging Results" for information about rounding rules.

The rules you specify are applied as a systemwide setting and apply for rounding the balance impact amounts of all charges, discounts, and chargeshares. The rounding is also applied to all reverse rating calculations and taxation calculations.

To configure rounding for a noncurrency resource:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.
   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.server**.
3. Expand **Attributes**.
4. Specify values for the following attributes:
   - **nonCurrencyScale**: Enter the number of digits you allow to the right of the decimal point for a calculated impact amount.

     For example, enter **2** if you allow two digits to the right of the decimal point.

     The default is **2**.
   - **nonCurrencyRoundingMode**: Enter the rounding mode that determines the rounding behavior by entering the string representation of the Java math rounding enum.

     For more information, see the Java SE technical documentation web site:

     `http://docs.oracle.com/javase/6/docs/api/java/math/RoundingMode.html`

     For example, enter **UP** to round up away from zero or **DOWN** to round down towards zero.

     The default is **HALF_UP**.

Configuring Rounding for Reverse Rating on Multiple RUMs

When ECE performs the reverse rating service in which events are rated by using multiple RUMs, fractional values may result for the authorized resource. You configure whether to round up the fractional value of the impacted resource quantity as a systemwide setting. ECE authorizes an additional RUM unit when the quantity is a fraction.

To configure whether to round up the fractional value of the authorized resource quantity by authorizing an additional RUM unit:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.
   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.server**.
3. Expand **Attributes**.
4. Set the **reverseRateUseAllBalances** attribute to one of the following values:
   - To round up the fractional value of the authorized balance quantity, enter **true**.

     This option allows customers to use all balances even if they might exceed their credit limits by a small amount.
   - To disallow the fractional value of the authorized balance quantity to be rounded up, enter **false**.

     This option does not allow customers to exceed their credit limits.

   The default is **false**.

Configuring a Tolerance for Policy-Tier Threshold Breaches for Policy-Driven Charging

You can configure a tolerance for policy-tier threshold breaches when you implement policy-driven charging. See "About a Tolerance for Policy-Tier Threshold Breaches for Policy-Driven Charging" for information.

To configure a tolerance for policy-tier threshold breaches:

1. Before charging servers are started, open *ECE_home*/**oceceserver/config/management/charging-settings.xml** and uncomment the following lines:

```
<toleranceConfigMappingGroup  config-class="java.util.ArrayList">
          <toleranceConfig
   config-class="oracle.communication.brm.charging.appconfiguration.
   beans.policy.ToleranceConfig"
              balanceElementId="12345" tolerance="1.25"/>

        <toleranceConfig
   config-class="oracle.communication.brm.charging.appconfiguration.
   beans.policy.ToleranceConfig"
               balanceElementId="34567" tolerance="3"/>
 </toleranceConfigMappingGroup>
```

2. Save the file.
3. On the driver machine, change directory to the *ECE_home*/**oceceserver/bin** directory.

4. Start Elastic Charging Controller (ECC):

   ```
   ./ecc
   ```

5. Start your charging servers:

   ```
   start server
   ```

6. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   c. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.
   d. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
7. Expand **charging.policyConfig**.
8. Expand **Operations**.
9. Select **setPolicyTolerance**.
10. For *each* balance element (policy counter) to which offer profiles apply in your system, do the following:
    a. Specify values for the following parameters:

       beid: Enter the balance element ID of the balance element.

       tolerance: Enter the RUM units allowed to exceed the authorized usage quantity that ECE returns to the network for a specified charging session.

       The value must be greater than **0**. Base it on your business needs.

       Your customers can use all balances and exceed their policy-tier threshold limits by the specified number of RUM units.
    b. Click the **setPolicyTolerance** button.

Configuring Systemwide Consumption Rules for Balances

When more than one validity-based subbalance is available for a usage request, consumption rules determine from which balance bucket ECE is to consume first. For example, if a customer has several groups of free minutes that expire at different times, you use consumption rules to indicate which minutes to use first, based on the validity period start time and end time. Consumption rules are typically configured at the balance element level when you define pricing in the pricing application such as PDC. Consumption rules can also be configured at the customer balance level by the customer and subscription management components of the BRM system. For information about configuring consumption rules in PDC and BRM, see the PDC documentation and the BRM documentation.

When ECE receives a usage request for which no consumption rules are configured, ECE applies its own systemwide consumption rules for processing the usage request.

To configure ECE systemwide consumption rules:

1. Access the ECE MBeans:
   a. Log on to the driver machine.

b. Start the ECE charging servers (if they are not started).

c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.

d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

   The **eceTopology.conf** file also contains the host name and port number for the node.

e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.

2. Expand **charging.server**.
3. Expand **Attributes**.
4. Set the **systemConsumptionRule** attribute to one of the following systemwide consumption rules:
   - **EARLIEST_START**
   - **LATEST_START**
   - **EARLIEST_EXPIRATION**
   - **LATEST_EXPIRATION**
   - **EARLIEST_START_LATEST_EXPIRATION**
   - **EARLIEST_START_EARLIEST_EXPIRATION**
   - **LATEST_START_LATEST_EXPIRATION**
   - **LATEST_START_EARLIEST_EXPIRATION**
   - **EARLIEST_EXPIRATION_EARLIEST_START**
   - **EARLIEST_EXPIRATION_LATEST_START**
   - **LATEST_EXPIRATION_EARLIEST_START**
   - **LATEST_EXPIRATION_LATEST_START**
   - **NONE**: When the attribute is set to **NONE**, the default consumption rule is not configured, and the order for consuming balances is undefined.

   By default, this attribute is set to **EARLIEST_START_EARLIEST_EXPIRATION**.

Defining Systemwide Credit Profiles

Credit profiles define the credit limits, credit floors, and credit threshold combinations that can be associated with balance elements. You design credit profiles in BRM and can customize them for individual customers. Credit profiles are applied to individual customers as part of the registration process, when customers purchase your products.

ECE must have systemwide (or default) credit profiles. Systemwide credit profiles are used by ECE when it receives a usage request for which the customer's credit profile has not been defined for the customer. ECE uses systemwide credit profiles to establish the credit for customers who have no credit profile.

You define the systemwide credit profiles in BRM. You must include definitions for the ECE systemwide credit profiles for currency (ID = **0**) and for noncurrency (ID = **2**).

Redirecting a Subscriber Session to a Service Portal

Service providers can redirect a subscriber session to a service portal, a server outside of the online charging system, where specific services can be offered to the subscriber.

During an online charging session, if a subscriber is about to deplete funds for the use of a service, the subscriber can be redirected to a web site to top up the account. You can configure ECE to send service portal addresses back to credit-control clients. Credit-control clients use the information for redirecting a subscriber session to the service portal applicable to the business scenario.

ECE derives the service portal address (to send back to credit-control clients) based on configurable instructions that you define in *redirection rules*. Your redirection rules can be based on any of the following customer conditions (typically based on a combination of them):

- Whether the customer has insufficient funds
- Whether the customer has an inactive account
- Whether the customer is roaming or not roaming
- Whether the customer belongs to a specific customer segment (for example, customer accounts associated with a BRM business profile for which the payment type is Prepaid or Postpaid or the subscription type is Voice or Data.)

Each redirection rule can send the session to a different service portal.

For example, you might configure two redirection rules for the following business scenarios:

- Given a customer with an account using a *prepaid* payment type who is roaming, redirect the subscriber to **http://myPrePaidRoamingRedirect.com**.
- Given a customer with an account using a prepaid payment type who is *not* roaming, redirect the subscriber to **http://myPrePaidHomeNetworkRedirect.com**.

After ECE derives the service portal addresses and address types based on your redirection rules, ECE sends the address back to the credit-control client.

To redirect a subscriber session to a service portal:

1. Create your redirection rules in a text editor and save the file.

   If you have multiple redirection rules, you must separate them by semicolons and save them as a single line. The single-lined redirection configuration should contain all of the redirection rules for the business scenarios that require redirecting subscriber sessions to applicable service portals.

   See "Creating Redirection Rules".

2. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**ocecerver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.

   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
3. Expand **charging.redirectionConfiguration**.
4. Expand **Attributes**.
5. Set the **redirectionRule** attribute to a copy of your redirection-rule configuration.

The default value is an empty string.

If no rule is provided, no redirection is done. ECE terminates the session.

ECE begins using the redirection-rule configuration at runtime.

If your redirection rule uses incorrect syntax, ECE logs the **Rule Evaluation Failed** error at runtime in the charging-server node log files (**ecs** log files) and leaves the redirection rule field in the usage response empty.

Tip: Modifying a redirection-rule configuration in the JConsole window may be error prone because you cannot see the entire rule. Modifying a redirection-rule configuration in the file where you created it is recommended. Pressing **Ctrl + A** in the Value column of the **redirectionRule** variable selects all contents.

Creating Redirection Rules

A redirection rule contains conditions that must be met for the subscriber session to be redirected to a service portal.

Your redirection configuration might contain a Voice redirection rule and a Data redirection rule for redirecting subscribers to service portals relevant to those services.

You must use allowed redirection-rule conditions.

The following scenario:

When the customer is roaming
The redirect address is http://RedirectRoaming.com
The redirect address type is URL

Is redirected by using the following redirection rule:

```
"( (@fui AND @roamingRequest) => [redirect_type:"URL",redirect_address:
"http://RedirectRoaming.com"];
```

The following scenario:

When the customer is Postpaid
And the customer is roaming
The redirect address is http://RedirectRoaming.com
The redirect address type is URL
The redirection must be performed within 900 seconds

Is redirected by using the following redirection rule:

```
"( (@fui AND ({business_profile([name:"POSTPAID"])} == "true" )) AND
@roamingRequest) => [redirect_type:"URL",redirect_address:
"http://RedirectRoaming.com",redirect_validity:"900"]"
```

Table 9–1 shows redirection-rule conditions that you can use to create redirection rules.

Table 9–1 ECE Redirection-Rule Conditions

ECE Redirection-Rule Conditions	Description
@fui	Checks in the charging result if the customer has insufficient funds (finds the Final Unit Indicator in the service context). @fui is required.
{business_profile([name:"*BusinessProfileName*"])} == "true") For example: `{business_profile([name:"POSTPAID"])} == "true" )`	Accesses a business profile by looking up a business profile name and comparing its value to true. Valid values for *BusinessProfileName* are names of attributes you defined in the attribute-value pairs of your BRM business profiles.
@roamingRequest	Checks if the request is for a customer who is roaming. **@roamingRequest** denotes roaming. **!@roamingRequest** denotes not roaming. The check is done on the value of the following Diameter credit-control-request fields: ■ GGSN-MCC-MNC-3GPP ■ IMSI-MCC-MNC-3GPP **Note:** These fields are not provisioned in ready-to-use event specifications. You must provision these network fields when you enrich your event definitions in PDC.

{request_attribute([name:"*FieldName*"])}	Reads a payload field from a usage request.
	For example, the following condition reads the simple attribute 3GPP-IMSI-MCC-MNC from the payload of the usage request:
	`{request_ attribute([name:"3GPP-IMSI-MCC-MNC"])}`
	Use this construct to use any request attribute field as a condition in your redirection rule. For example, if you want subscribers to be directed to a different URL if they have a 1234 cell phone ID, you might use the condition:
	`{request_attribute([name:"CELL_ID"])} == "1234")`

@productType	Retrieves the product type. For example, a redirection rule using this condition: `( (@productType == 'DATA') AND ( {request_ attribute(name:"GGSN-MCC-MNC- 3GPP"])} == "1234") ) => [redirect_type:"URL",redirect_ address:"myDataTopUpRedirect.com"]`

Redirection-Rule-Configuration Syntax

You configure one or multiple redirection rules in a single-lined redirection configuration with each redirection rule separated by semicolons.

The syntax for a redirection rule is the following:

((*redirection_condition* **AND** *redirection_condition***) AND** *redirection_condition***) => [redirect_type:"***redirect_type***",redirect_address:"***redirect_address***",redirect_validity:"***redirect_validity***"];**

where:

- *redirection_condition* is a condition that must be met for ECE to send the specified redirect type, redirect address, and redirect validity in the ECE usage response. See Table 9–1 for accepted redirection-rule conditions.
- *redirect_type* is the type of the service portal address (for example, **URL**)
- *redirect_address* is the service portal address (for example, a web site address)
- *redirect_validity* is the time, in seconds, that the subscriber being redirected has to complete the task that must be done at the service portal. The value you enter here overrides the default reservation validity time of ECE. If you do not specify a redirect validity in your reservation rule, then the default reservation validity time of ECE is sent back to the credit-control client.

When you design your redirection rules, it can be helpful to create a user scenario for each and show the translation in a table, as shown in the following examples.

Example Redirection Rules

The following is an example of redirection rules.

> **Tip:** For visual clarity, this example shows a carriage return after each redirection rule. Your redirection-rule configuration would be one line comprised of these four redirection rules *separated only by semicolons*.

```
"( (@fui AND ({business_profile([name:"POSTPAID"])}  == "true" )) AND  @roamingRequest) =>
[redirect_type:"URL",redirect_address:"http://myPostPaidRoamingRedirect.com",redirect_
validity:"900"];

( (@fui AND ({business_profile([name:"POSTPAID"])}  == "true" )) AND  !@roamingRequest) =>
[redirect_type:"URL",redirect_address:"http://myPostPaidHomeNetworkRedirect.com",redirect_
validity:"900"];

( (@fui AND ({business_profile([name:"PREPAID"])}  == "true" )) AND  @roamingRequest) => [redirect_
type:"URL",redirect_address:"http://myPrePaidRoamingRedirect.com"];

( (@fui AND ({business_profile([name:"PREPAID"])}  == "true" )) AND  !@roamingRequest) =>
[redirect_type:"URL",redirect_address:"http://myPrePaidHomeNetworkRedirect.com"]"
```

The four redirection rules support redirecting subscribers who have depleted funds in their account to a service portal for these scenarios:

- Given a subscriber with an account using a postpaid payment type who is roaming, redirect the subscriber to **http://myPostPaidRoamingRedirect.com** and allow the subscriber to use network resources for 900 seconds.
- Given a subscriber with an account using a postpaid payment type who is *not* roaming, redirect the subscriber to **http://myPostPaidHomeNetworkRedirect.com** and allow the subscriber to use network resources for 900 seconds.

- Given a subscriber with an account using a *prepaid* payment type who is roaming, redirect the subscriber to **http://myPrePaidRoamingRedirect.com**.
- Given a subscriber with an account using a prepaid payment type who is *not* roaming, redirect the subscriber to **http://myPrePaidHomeNetworkRedirect.com**.

Configuring ECE to Generate Midsession Rated Events

By default, ECE generates a rated event for a network session only when a Diameter terminate operation ends the session. You can also configure ECE to generate a rated event whenever a Diameter update operation occurs during the network session. Such events are called *midsession rated events*. For more information, see the discussion about midsession rated events in *BRM Elastic Charging Engine Concepts*.

To generate midsession rated events, you enable the feature and then define conditions, called *triggers*, that initiate the generation of such events. Triggers are based on one or more of the following criteria:

- Duration (for example, every 4 hours that a session is active)
- Quantity (for example, whenever downloaded data totals 70 MB or more)
- Time of day (for example, daily at 23:00:00 during the life of the session)

Each trigger is associated with a service-event pair. If an ongoing session meets the trigger conditions at the time an update operation occurs, a midsession rated event for the specified service is generated.

> **Note:** Trigger conditions are examined only during update operations. If a trigger condition is "every 200 MB" but an update operation does not occur until the total is 288 MB, the rated event is for 288 MB, not 200 MB. The same applies to duration criteria.

For example, the following code triggers the generation of a rated **/data_usage** event for a **DATA** service's ongoing network session if at least one of the following conditions is true:

- The combined values of the event's **input_volume** and **output_volume** fields total 70 MB or more.
- The current time minus the time the last midsession rated event was generated is greater than or equal to 7 hours.
- The current time is greater than or equal to 11 p.m.

```
<midSessionCdrConfiguration
config-class="oracle.communication.brm.charging.appconfiguration.beans.midsessionc
dr.MidSessionCdrConfiguration"
  midSessionCdrEnabled="true">

<productConfigurationGroup config-class="java.util.ArrayList">
  <productLifecycleConfiguration
   config-class="oracle.communication.brm.charging.appconfiguration.beans.
   midsessioncdr.MidSessionCdrConfiguration"

   productType="DATA">

   <eventConfigurationGroup config-class="java.util.ArrayList">
    <eventConfiguration
     config-class="oracle.communication.brm.charging.appconfiguration.beans.
```

```
     midsessioncdr.MidSessionCdrConfiguration"

     eventType="DATA_USAGE">

      <triggerConfiguration
       config-class="oracle.communication.brm.charging.appconfiguration.beans.
       midsessioncdr.MisSessionCdrTriggerConfiguration"
       durationunit="HOURS"
       durationvalue="7"/>

      <triggerConfiguration
       <!-- Use ";" to separate fields. Values in the fields are summed. -->
       quantiytfields="input_volume;output_volume"
       quantityunit="MEGABYTES"
       quantityvalue="70"/>

      <triggerConfiguration
       config-class="oracle.communication.brm.charging.appconfiguration.beans.
        midsessioncdr.MidSessionCdrConfiguration"
        timeofday="23:00:00"/>

    </eventConfiguration>
   </eventConfigurationGroup>
  </productLifecycleConfiguration>
</productConfigurationGroup>
</midSessionCdrConfiguration>
```

Note:

- All conditions in a **TriggerConfiguration** block must be met (criteria are assumed to be joined by AND).
- If a trigger contains multiple **TriggerConfiguration** blocks, the conditions in only one block must be met (blocks are assumed to be joined by OR).

To configure ECE to generate midsession rated events:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.

   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.midSessionCdrConfiguration**.
3. Expand **Attributes**.
4. Set the **midSessionCdrEnabled** attribute to **true**.
5. Define trigger conditions for one or more service-event pairs:

a. Expand **Operations**.

b. Click **addOrUpdateMidSessionCdrTriggerDetails**.

c. Specify values for the parameters listed in Table 9–2:

Table 9–2 Parameters for Defining Midsession Rated Event Triggers

Parameter	Description
productType	Name of the service for which you are creating the trigger (for example, **"DATA"**).
eventType	Name of the event for which you are creating the trigger (for example, **"DATA_USAGE"**).
triggerName	Name of the trigger you are defining.
qtyFields	Name of one or more event fields to which a quantity condition applies (for example, **"input_volume; output_volume"**). Use a semicolon (**;**) to separate field names. Values in the fields are summed.
qtyUnit	Unit of measure for conditions based on quantity (for example, **"MEGABYTES"**).
qtyValue	Total quantity of the unit that triggers event generation (for example, **"70"**).
durationUnit	Unit of measure for conditions based on duration (for example, **"HOURS"**).
durationValue	Amount of the unit that triggers event generation (for example, **"70"**).
timeOfDay	A particular time of day in a 24-hour clock at which to generate the event (for example, **"23:00:00"**, which indicates 11 p.m.). Use the *hh*:*mm*:*ss* format.

A trigger with one **TriggerConfiguration** block is created for the specified service-event pair. All conditions in the block (quantity, duration, time of day) must be met to generate a midsession rated event.

d. (Optional) Do one of the following:

To define another trigger, click the plus sign in the panel's upper right corner, and repeat step 4 for a *different* service-event pair.

To add a **TriggerConfiguration** block to the current trigger, click the plus sign in the panel's upper right corner, and repeat step 4 for the *same* service-event pair.

10

Configuring Charging Runtime Options

This chapter provides instructions for configuring charging runtime options for Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE).

About Charging Runtime Options

This sections describes the charging runtime options you configure in ECE:

- About Taxation
- About Notifications
- About Server-Initiated Reauthorization Requests (RAR)
- About Debit Request History
- About Open-Session Management for Network Element Failures

About Taxation

ECE supports a fixed rate tax (a flat-rate taxation which is also known as GST or VAT).

You can apply a tax on both charges and alterations (discounts).

For information about configuring taxation, see "Configuring Taxation".

About Notifications

For information about notifications you can configure, such as those for online charging (typically for use by the online network mediation system), as well as those used for sending data to other applications in the charging system, see the discussion about notifications in *BRM Elastic Charging Engine Concepts*.

About Server-Initiated Reauthorization Requests (RAR)

ECE supports server-initiated reauthorization requests (RAR).

ECE can perform server-initiated reauthorization during an ongoing session. This allows you to update a session in response to changes that occur to a customer's product offerings or balance (for example, a change to a charge offer or to a Friends and Family promotion). When ECE notifies the network, the network sends a reauthorization request, and, if there is a change in the charge, ECE can base the reauthorization on the new charge.

A server-initiated reauthorization can be triggered by the following:

- Changes to offers, such as the creation, modification, or deletion of a subscriber's charge offer or alteration offer
- Changes to balances that affect rating (for example, a resource that expires mid-session, a resource that becomes available from a top-up, or changes to the customer balance due to an accounts receivable action)
- Changes to promotions, such as changes to Friends and Family or a Special Day offer
- Changes to charge sharing or alteration sharing groups. For example, a new member is added to the group or a member is removed mid-session

For example:

1. A subscriber is in a call session. The subscriber adds the called number of that session to a Friends and Family list.
2. Because a Friends and Family discount might change the charge amount, ECE sends a request to the network.
3. In response, the network sends a reauthorization request.
4. ECE sends a reauthorization, using the Friends and Family charge amount.

Note: A reauthorization request is not triggered by a top-up or by rerating when resources are added to a sharing group owner's account.

For information about configuring server-initiated RAR in ECE, see "Configuring Support for Server-Initiated Reauthorization Requests".

About Debit Request History

For a debit request, ECE returns a correlation ID in the usage response and stores the correlation ID in a debit map for each product. If a refund request is later received for the debit request, ECE uses the correlation ID to validate the refund request (refund requests are valid only when they are associated with a debit request correlation ID).

The debit request information in the debit map is transient data, and you can configure the number of debit requests to be retained per product. By default, debit request information is stored for 10 debit request operations (for each product) at any given time. For example, if there are 10 entries in the debit map and a new debit request is received, the entry for the oldest debit request is deleted from the debit map and an entry for the new debit request is added.

For information about configuring debit request history, see "Configuring Debit Request History".

About Open-Session Management for Network Element Failures

When a network element associated with active sessions in ECE fails, ECE receives an accounting on/off request from the network element. You can configure ECE to cancel or terminate active sessions when processing accounting on/off requests. See "Configuring Open-Session Management for Network Element Failures" for instructions.

Configuring Charging Runtime Options

This section provides instructions for configuring charging-related runtime options:

- Configuring Taxation
- Configuring the Return of Remaining Balances in Usage Responses
- Configuring ECE to Align the Validity Start and End of Conditional Balance Impacts
- Configuring Support for Server-Initiated Reauthorization Requests
- Configuring Debit Request History
- Configuring Open-Session Management for Network Element Failures

Configuring Taxation

To configure taxation:

> **Note:** For taxation to work in the ECE runtime environment, you must set the following mandatory parameters.

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.

   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.taxation**.
3. Expand **Operations**.
4. Click **addTaxDetails**.
5. Specify values for the following parameters:

> **Important:** These parameters are mandatory. You must set all of them when configuring taxation.

- **taxCode**: Enter the tax code used by the charge offer or discount offer to which the tax applies.

  The tax code is used by charge offers and discount offers to point to the tax rate that must be applied when a usage request is processed for the charge offer or discount offer.

  Enter the same tax code entered in PDC when the taxation section of the charge offer and discount offer was defined.

- **taxRate**: Enter the tax rate to apply.

For example, entering **0.20** applies a 20% tax on the total usage impact.

- **taxGlId**: Enter the General Ledger ID used for the tax impact.

6. Specify an additional **taxCode**, **taxRate**, and **taxGlId** value for each charge offer or discount offer to which a tax applies.

Configuring the Return of Remaining Balances in Usage Responses

You can configure ECE to return the customer's remaining-balance information in the usage response (as an in-session notification). For example, you could use the information to send customers a low-balance notification when they are about to use up all of their available balance for a service or they reach a balance amount set in your system to trigger such notifications.

ECE sends remaining-balance information for initiate and update usage requests.

The remaining-balance information that ECE returns pertains to all resources that were impacted by the session; that is, the resources to which the session applied balance impacts.

For charge distribution scenarios (charge sharing), ECE returns the remaining-balance information for the resources impacted by the sharer's usage.

For detailed information about the remaining-balance information returned in the usage response, see the documentation for oracle.communication.brm.charging.messages.usage (for remaining balance information) in *BRM Elastic Charging Engine Java API Reference*.

To configure ECE to return remaining-balance information in the usage response:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**ocecesever/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.

   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.server**.
3. Expand **Attributes**.
4. Set the **remainingBalanceCalcMode** attribute to one of the following values:
   - **NONE**: (Default) Sends no remaining-balance information in usage responses.

     ECE does not calculate the remaining balance.

   - **CURRENT_BALANCE**: Sends remaining-balance information for the current balance, excluding the credit limit, in the usage response. Use this option to notify your customers of their plain vanilla remaining balance.

     ECE calculates the remaining balance by adding all sub-balances valid for the session, including the consumed reserved amount of ongoing sessions. The remaining balance is calculated as follows:

     ```
     remaining balance = sum of for valid sub-balances of (current balance +
     ```

```
consumed reserved amount)
```

- **UPTO_CREDIT_LIMIT**: Sends remaining-balance information capped at the credit limit in the usage response. Use this option to notify your customers of the credit limit up to which you allow them to use the balance.

  ECE calculates the remaining balance by adding all sub-balances valid for the session, including the consumed reserved amount of ongoing sessions (the consumed reservation of the balances ECE reserved for ongoing sessions) and subtracts that value from the credit limit.

  ECE calculates the remaining balance as follows:

  ```
  remaining balance = {credit limit - sum of for valid sub-balances of
  (current balance + consumed reserved amount)}
  ```

Configuring ECE to Align the Validity Start and End of Conditional Balance Impacts

When you design your pricing components in PDC, you can create charges for which conditional balance impacts are configured. See *PDC User's Guide* for information about conditional balance impacts.

You can configure ECE to align the validity start and validity end of a conditional balance impact based on the period start set in PDC.

For example, if a customer activates a conditional balance impact that is valid for three days and period start is set to ALIGN_WITH_OFFER_START, and the purchased charge offer is no longer valid after one day, this configuration specifies whether the conditional balance impact can still be used after the purchased charge offer validity has ended. If ECE does not align the validity end of the conditional balance impact with the validity end of the purchased charge offer, the balance could be used by another charge offer.

To configure ECE to align the validity start and end of conditional balance impacts:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.
   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.server**.
3. Expand **Attributes**.
4. Set the **alignRecurringImpactsToOffer** attribute to **true**.

   At run time, if this is set to **true** and ECE receives a usage request for which a conditional balance impact applies, ECE compares the validity start and end of the conditional balance impact with the usage validity start and end of the associated charge offer that the customer purchased. If the validity start or end of the conditional balance impact breaches the validity start or end of the associated

purchased charge offer, ECE aligns both the validity start and end of the conditional balance impact with those of the charge offer.

Configuring Support for Server-Initiated Reauthorization Requests

To enable server-initiated reauthorization requests (RAR):

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.
   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.notification**.
3. Expand **Attributes**.
4. Set the **rarNotificationMode** attribute to **ASYNCHRONOUS**.

   This enables RAR notifications, which are required for server-initiated reauthorization requests. ECE generates an external notification and sends it to a notification queue (JMS topic) when the **RAR_NOTIFICATION_EVENT** service event is created. When specific condition changes occur during a session, ECE generates a RAR notification to inform the network to request a reauthorization.
5. Under the **ECE Configuration** node, expand **charging.server**.
6. Expand **Attributes**.
7. Set the **offerEligibilitySelectionMode** attribute to **PERIOD**.

   In **PERIOD** mode, ECE selects applicable charge offers valid any time between the start and end time of the session to determine charges for events. You use this mode when implementing server-initiated reauthorization requests so that ECE can rate based on changes to a customer's subscription, such as the purchase of a promotional offer, during the session.

   By default, this parameter is set to **END_TIME**.

   In **END_TIME** mode, ECE selects charge offers valid at the end time of the session to determine charges for events. **END_TIME** mode must be used when using a version of BRM that does not support **PERIOD** mode.

Note: Events rated in PERIOD mode might result in a different charge from the charge calculated when the event is rerated. This happens because the event is rerated using only the pricing applicable at the event end time.

Configuring Debit Request History

To configure the debit request history:

1. Access the ECE MBeans:

a. Log on to the driver machine.

b. Start the ECE charging servers (if they are not started).

c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.

d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

   The **eceTopology.conf** file also contains the host name and port number for the node.

e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.

2. Expand **charging.server**.
3. Expand **Attributes**.
4. Set the **debitRefundSessionEvictionSize** attribute to the maximum number of debit requests to log in the debit map at one time.

   This is the number of debit requests to keep in history so that refunds can be made against them.

Configuring Open-Session Management for Network Element Failures

To configure how to manage open sessions in ECE when a network element associated with those open sessions fails:

1. Access the ECE MBeans:

   a. Log on to the driver machine.

   b. Start the ECE charging servers (if they are not started).

   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.

   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.

   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.

2. Expand **charging.server**.
3. Expand **Attributes**.
4. Set the **accountingOnOffMode** attribute to one of the following values:

   - **TERMINATE**: Active sessions that have a state of Initiated are terminated when an accounting on/off request is processed.
   - **CANCEL**: Active sessions in ECE that have a state of Initiated are canceled when an accounting on/off request is processed.

Part IV

Integrating ECE with Client Programs

Part IV describes how to integrate Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE) with client programs such as network charging functions. It contains the following chapters:

11

Overview of Integrating Client Applications with ECE

This chapter is an overview of how client applications and network charging functions integrate with Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE).

ECE Integration with Client Applications

You can integrate ECE with client applications so that the Elastic Charging Server (charging server nodes) can query data and perform usage processing.

Your client applications use ECE client libraries (the Elastic Charging Client) for connecting to ECE and executing operations for usage requests (charging) and query requests.

Your client applications are part of the Oracle Coherence grid (ECE cluster) but they do not cache ECE business data.

For usage request submitted by charging clients, Elastic Charging Server fetches request-related data, such as the customer and pricing data, performs required preprocessing of the data, rates (calculates the charge), updates the customer balance and active session, and then builds a response. Elastic Charging Server sends the response back to your client application. See "Integrating Charging Clients with ECE".

For query requests submitted by query clients, Elastic Charging Server fetches the data in the query request and builds a response, sending the response back to your client applications. See "Integrating Query Clients with ECE".

Top-up clients, such as those in a Third-Party top-up system, can integrate with ECE. Top-up clients execute operations for update requests (for example, a top-up updates the customer account balance). See "Integrating Top-Up Clients with ECE" for information.

Policy clients, such as the Policy and Charging Rules Function (PCRF), can integrate with ECE. See "Integrating Policy Clients with ECE".

You can also write client applications for management requests to search for open sessions related to network elements that have failed. For management requests, the Elastic Charging Server searches across the Coherence grid and executes the action on the search results.

ECE offers services that expose its client APIs so that client applications can obtain information from ECE for further processing. See "ECE API Reference" for information about ECE public APIs.

When you install the ECE Server software, the ECE SDK is installed in *ECE_home*/**ocecesdk**. The ECE SDK contains client libraries that enable your client applications to connect to the ECE cluster and build different types of requests. The ECE SDK includes sample programs that demonstrate how to use the ECE client APIs. See "About the ECE Sample Programs" for information.

12

About the ECE Sample Programs

This chapter describes how to use the sample programs included in the Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE) SDK to call the ECE APIs.

About the Sample Programs

ECE SDK includes a set of sample programs that demonstrate how to use the ECE API for sending requests to ECE.

You use these sample programs in the following ways:

- Use the sample programs as code samples for calling the ECE APIs.
- Use the sample programs as code samples for writing custom applications.
- Run sample programs to send requests to ECE and receive responses.

  The sample programs print information about the messages exchanged.
- Use the sample program scripts to get an idea of the configuration and dependencies that are required for integrating the ECE client into your build system (Maven, Ant, and so on).

Look at the sample program source to see how the program works. For example, if you want to write a program that sends a unit-based debit request to ECE, examine the SampleDebitRefund sample program source to:

- View the methods to use in your code.
- How to use the libraries and calls.

The samples are supported on the Linux and Oracle Solaris platforms. Compile the sample programs using the shell script provided for each sample program. See "Compiling and Running the Sample Programs" for more information.

Finding the Sample Programs

The SDK sample programs are found in the *ECE_home*/**ocecesdk/source/oracle/communication/brm/charging/sdk** directory.

Table 12–1 shows the ECE SDK software directory structure: *ECE_home* is the directory in which ECE Server software is installed.

Table 12–1 Elastic Charging Engine Sample Program Directories

Directory	Description
ECE_home/**ocecesdk/bin**	Directories that contain shell scripts for compiling and running various types of sample programs.
ECE_home/**ocecesdk/bin/extensions**	Shell scripts for extension-implementation sample programs.
ECE_home/**ocecesdk/bin/notification**	Shell scripts for notification sample programs.
ECE_home/**ocecesdk/bin/plugin**	Shell scripts for BrmCdrPluginDirect plug-in sample programs.
ECE_home/**ocecesdk/bin/policy**	Shell scripts for policy sample programs.
ECE_home/**ocecesdk/bin/query**	Shell scripts for query sample programs.
ECE_home/**ocecesdk/bin/update**	Shell scripts for update sample programs.
ECE_home/**ocecesdk/bin/usage**	Shell scripts for usage sample programs.
ECE_home/**ocecesdk/config**	Configuration files common to all sample programs.
ECE_home/**ocecesdk/config/extensions**	Configuration files for extension-implementation sample programs.
ECE_home/**ocecesdk/source** **Note:** The full path showing the Java project directory structure to the sample programs is: *ECE_home*/**ocecesdk/source/oracle/communication/brm/charging/sdk**	All Java sample programs.
ECE_home/**ocecesdk/source/.../sdk/extensions**	Source files for extension-implementation sample programs (for pre-request-processing and post-request-processing). Includes the data loader used for extensions.
ECE_home/**ocecesdk/source/.../sdk/notification**	Source files for notification sample programs.
ECE_home/**ocecosdk/source/.../sdk/plugin**	Source files for BrmCdrPluginDirect plug-in sample programs.
ECE_home/**ocecesdk/source/.../sdk/policy**	Source files for policy sample programs.
ECE_home/**ocecesdk/source/.../sdk/query**	Source files for query sample programs.
ECE_home/**ocecesdk/source/.../sdk/update**	Source files for update sample programs.
ECE_home/**ocecesdk/source/.../sdk/usage**	Source files for usage sample programs.

Descriptions of the Sample Programs

Each sample program includes these supporting files:

- Source files to view or modify for your own applications
- Shell scripts to compile and run the sample programs

The sample programs use the generic **.ecc** script **sdk_production_loader.ecc**.

All of the sample programs can work with the ready-to-use sample data included with the ECE Server software installation (the sample data you load using ECE data-loading utilities).

> **Note:** The ECE sample programs do not work well with data you load using the simulator **loader** utility.

The following tables list the sample programs, the **.ecc** script that applies to each, the shell script that complies and runs the sample program, and the description of the sample program.

- Table 12–2, " ECE Sample Programs for Usage Requests"
- Table 12–3, " ECE Sample Programs for Update Requests"
- Table 12–4, " ECE Sample Programs for Policy Requests"
- Table 12–5, " ECE Sample Programs for Query Requests"
- Table 12–6, " ECE Sample Programs for Extension Implementations"
- Table 12–7, " ECE Sample Programs for Notifications"
- Table 12–8, " ECE Sample Programs for Custom Plug-in"

For descriptions of the methods the sample programs use, see the documentation for **oracle.communication.brm.charging.sdk** in *BRM Elastic Charging Engine Java API Reference*.

Table 12–2 lists the usage sample programs.

Table 12–2 ECE Sample Programs for Usage Requests

Sample Program	ECC Script	Shell Script	Description
SampleAccountingOnOff	sdk_production_loader.ecc	sample_accounting_on_off.sh	Simulates an accounting on/off request being sent from the mediation client.
SampleDataSession	sdk_production_loader.ecc	sample_data_session.sh	Simulates a simple data session, including an INITIATE, an UPDATE, and a TERMINATE request.
SampleDebitRefundSession	-	sample_debit_refund_session.sh	Shows how to send debit and refund requests with multiple values in unit-based and amount-based mode. See "Usage Example for SampleDebitRefundSession".
SampleGenericSession	-	sample_generic_session.sh	Simulates any kind of voice or data session.
SampleGprsSession	-	sample_gprs_session.sh	Simulates a GPRS session.
SampleIncrementalUsageRequestLauncher	sdk_production_loader.ecc	sample_incremental_usage_request.sh	Simulates a voice session with incremental mode.

Table 12–2 (Cont.) ECE Sample Programs for Usage Requests

Sample Program	ECC Script	Shell Script	Description
SampleMultipleServicestLauncher	sdk_production_loader.ecc	sample_multiple_service.sh	Shows how to send usage requests for the Multiple Services Credit Control (MSCC) case (multiple sub-requests are sent in a single usage request).
SamplePriceEnquiry	sdk_production_loader.ecc	sample_price_enquiry.sh	Sends a price enquiry request.
SampleReAuthRequest	sdk_production_loader.ecc	sample_RAR.sh	Sample program that shows the generation of reauthorization request (RAR) messages. Also shows how to consume notification messages. This portion of the code is for illustration only and is disabled.
SampleStartUpdateAccountingRequestLauncher	-	sample_start_update_accounting_request.sh	Simulates a sample usage session including a START_ACCOUNTING, UPDATE_ACCOUNTING, and TERMINATE request. For example, a usage session for a DSL data download in a postpaid scenario.
SampleUsageRequestLauncher	sdk_production_loader.ecc	sample_usage_request.sh	Allows you to send custom voice usage requests. Customer ID, number of requests to send, request type (INITITATE/UPDATE...) and duration must be given as arguments (for example, sample_usage_request.sh run 6500000000 2 TERMINATE 120).
SampleVoiceSession	sdk_production_loader.ecc	sample_voice_session.sh	Simulates a simple voice session, including an INITIATE, an UPDATE, and a TERMINATE request.

Table 12–3 lists the update sample programs.

Table 12–3 ECE Sample Programs for Update Requests

Sample Program	ECC Script	Shell Script	Description
SampleExternalTopUpRequestLauncher	-	sample_external_topup_notification_request.sh	Shows how third- party systems can perform direct top ups in ECE.

Table 12–4 lists the policy sample programs.

Table 12–4 ECE Sample Programs for Policy Requests

Sample Program	ECC Script	Shell Script	Description
SamplePolicySessionRequestLauncher	-	sample_policy_session_request.sh	Simulates a policy session. Shows how to send a policy request to ECE that requests both Sp and Sy information.
SampleSpendingLimitReportRequestLauncher	-	sample_spending_limit_report_request.sh	Simulates a policy Sy query request. Shows how to send a request to retrieve policy counter status information.
SampleSubscribeNotificationRequestLauncher	-	sample_subscribe_notification_request.sh	Simulates a policy Sp query request. Shows how to send a request to retrieve the value for a specified set of subscriber preferences and subscribe for receiving notifications when the values of the preferences change. For example, shows how to retrieve the channel a subscriber prefers for receiving policy-related notifications (SMS or email) or the language in which the subscriber prefers the notification to be written (French, English).
SampleSubscriberPreferenceUpdateRequestLauncher	-	sample_subscriber_preference_update_request.sh	Simulates a policy-related update request. Shows how to update the subscriber preferences in ECE.
SampleUserDataRequestLauncher	-	sample_user_data_request.sh	Simulates a policy Sp query request without subscription. Shows how to send a request to retrieve the values for subscriber preferences that are configured for the product of a customer.

Table 12–5 lists the query sample programs.

Table 12–5 ECE Sample Programs for Query Requests

Sample Program	ECC Script	Shell Script	Description
SampleAuthenticationQuery	sdk_production_loader.ecc	sample_auth_query_request.sh	Sends an authentication query request.
SampleBalanceQueryRequestLauncher	sdk_production_loader.ecc	sample_balance_query_request.sh	Sends a balance query request.

Table 12–6 lists the extension sample programs.

Table 12–6 ECE Sample Programs for Extension Implementations

Sample Program	ECC Script	Shell Script	Description
-	-	sample_extensions_loader	Data loader for extension implementations
-	-	build_deploy_extension	Sample extension implementation
-	-	tax_configuration.spec	Sample extension implementation
-	-	tax_configuration_data.csv	Sample extension implementation

Table 12–7 lists the notifications sample programs.

Table 12–7 ECE Sample Programs for Notifications

Sample Program	ECC Script	Shell Script	Description
SampleDurableJmsClient	-	sample_durable_jms_client.sh	Simulates a durable JMS client
SampleJmsClient	-	sample_jms_client.sh	Simulates a JMS client
SampleJmsServer	-	sample_jms_server.sh	Simulates a JMS server

Table 12–8 lists the custom plug-in sample programs.

Table 12–8 ECE Sample Programs for Custom Plug-in

Sample Program	ECC Script	Shell Script	Description
SampleRatedEventFormatterCustomPlugin	-	build_deploy_plugin.sh	Builds and deploys the sample plug-in SampleRatedEventFormatterCustomPlugin.

Compiling and Running the Sample Programs

You can compile and run a sample program with the shell script provided for that specific sample program as shown in Table 12–2.

To compile and run a sample program:

1. Open the *ECE_home*/**ocecesever/config/eceTopology.conf** file.
2. Un-comment the line where the **sdkCustomerLoader** node is defined.
3. Change directory to *ECE_home*/**ocecesever/bin**
4. Load the ECE runtime environment:

```
$ ./ecc 'load sdk_production_loader.ecc'
```

5. Change directory to *ECE_home*/**ocecesdk/sample_program_directory**, where the *sample_program_directory* is the subdirectory that contains the shell script for compiling and running the sample program you want to run:

Directory	Description
ECE_home/**ocecesdk/bin/extensions**	Shell scripts for extension-implementation sample programs.
ECE_home/**ocecesdk/bin/notification**	Shell scripts for notification sample programs.
ECE_home/**ocecesdk/bin/policy**	Shell scripts for policy sample programs.
ECE_home/**ocecesdk/bin/plug-in**	Shell scripts for BrmCdrPluginDirect plug-in sample program
ECE_home/**ocecesdk/bin/query**	Shell scripts for query sample programs.
ECE_home/**ocecesdk/bin/update**	Shell scripts for update sample programs.
ECE_vome/**ocecesdk/bin/usage**	Shell scripts for usage sample programs.

For example, to compile and run the **sample_voice_session.sh** sample program (the sample program for sending a voice session usage request to ECE), you must be in the *ECE_home*/**ocecesdk/bin/usage** directory.

6. Compile the sample program:

   $ sh ./*scriptname***.sh build**

   You must compile the sample program once.

   For example, to compile the **sample_voice_session.sh** sample program, enter:

   ```
   $ sh ./sample_voice_session.sh build
   ```

7. Run the sample program.

   $ sh. /*scriptname***.sh run**

   Some programs require that you enter parameters. The **run** command output gives you information about what parameters are required.

   If you run the **run** command with no parameters, the command uses default parameter values from the SDK scripts.

8. When you are done with the sample program, shut down the ECE runtime environment:

   ```
   $ ecc stop server
   ```

About Sample Program Parameters

To determine which parameter values you need to use for running a sample program, you can use the help option of the sample script. All of the sample scripts have a help option which prints the usage of the sample program.

About Sample Program Methods

For descriptions of the methods the sample programs use, see the documentation for **oracle.communication.brm.charging.sdk** in *BRM Elastic Charging Engine Java API Reference*.

Usage Example for SampleDebitRefundSession

Below is a usage example for running the SampleDebitRefundSession sample program.

To see the usage of any sample program, type no arguments at the command line.

Usage:

sample_debit_refund_session.sh build | run | defaultrun *userId requestType correlationId* [*BALANCE_ELEMENT_ID*,*AMOUNT BALANCE_ELEMENT_ID*,*AMOUNT* ...] [*TOTAL,IN,OUT TOTAL,IN,OUT* ...]

where:

- **build** compiles the related SDK source files
- **run** runs the SDK program (debit refund) according to the parameters you provide.

  You must supply all parameters in the command line.

  Alternatively, you can provide no parameters. If you provide no parameters, the SDK script will invoke the ECE sample program with default parameter values.
- **defaultrun** builds and runs the SDK program.

  No parameters are required. The program uses the default parameter specified inside the shell script.
- Order of parameters are fixed and if one optional parameter is provided then *all values of other optional parameters* must be supplied.
- requestType is either DEBIT_AMOUNT, REFUND_AMOUNT, DEBIT_UNIT, or REFUND_UNIT.
- In case of DEBIT_AMOUNT or REFUND_AMOUNT, BALANCE_ELEMENT_ID is the well-known ISO code for balance elements, such as 840 for US Dollars, or 95 for Free Minutes.
- In case of DEBIT_UNIT or REFUND_UNIT, TOTAL,IN and OUT has to be specified with numbers in MB (MegaBytes).

Example:

#debit $10 for USD and 25 seconds Free Minutes

```
sample_debit_refund_session.sh run 650999777 DEBIT_AMOUNT CORR_ID 840,10 95,25
```

#refund $50 for USD and 5 seconds Free Minutes

```
sample_debit_refund_session.sh run 650999777 REFUND_AMOUNT CORR_ID 840,50 95,5
```

13

Integrating Charging Clients with ECE

This chapter describes how to integrate charging client applications with Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE).

Note: The information in this chapter is relevant for those using third party network mediation software (for online charging) for receiving network messages and translating them into ECE requests.

If you are using Diameter Gateway for receiving Diameter messages and translating them into ECE requests, see "Network Integration for Online Charging Using Diameter Gateway". Diameter Gateway constructs all charging-operation-type requests that ECE supports for online charging.

About Integrating ECE with Charging Diameter Applications

You can integrate ECE with network mediation systems so that they can send requests to ECE for executing charging and network operations in ECE.

Running the ECE sample programs is the best way to learn how to use the ECE API for integrating ECE with external network mediation systems. The Java API documentation is also a general resource for learning how to use the ECE API.

Third party network mediation software programs can use different types of builders to build different ECE requests. The network mediation software program receives the network request and transfers the relevant request fields from the network-request data structures into the UsageRequest Payloads by using the ECE UsageRequestBuilder APIs.

You call a constructor which is an ECE API (a public file included in the ECE SDK jar file installed on the client) to make a request (Build) and then call other usage-request classes to populate the request (Payload/createPayload). See "About Building Usage Requests" for more information.

ECE is pre-integrated with Offline Mediation Controller for offline charging; Offline Mediation Controller includes pre-made builders for building the different request types needed for offline charging (charging requests, balance query requests, and so on). See "Integrating ECE with Offline Mediation Controller".

About Building Usage Requests

This section describes how network mediation software programs (client applications) build usage requests.

About the Usage Request Builder

The usage request builder creates usage requests; it is created and used by the network mediation software program when a request from the network is received.

Network mediation software programs use the Elastic Charging Client API for instantiating the usage request builder. The Elastic Charging Client API is included in the ECE SDK.

For sample code that demonstrates how to use the usage request builder, see the sample programs included in the ECE SDK.

Using Incremental or Cumulative Accounting for Usage Requests

ECE supports incremental and cumulative based accounting behavior when processing usage requests.

Incremental accounting logic is used by the Diameter standard which supports Requested Service Units (RSU) and Used Service Units (USU) concepts. Incremental accounting logic indicates that the creator of the usage request enables the rating engine (ECE) to calculate in the active session the duration based on the units that have been used since the previous session update.

Cumulative accounting logic is used by the Radius standard which indicates that the creator of the usage request always supplies the full quantity (for example: duration, volume, meters, miles and so on) inclusive of all previous session requests.

When you create your usage request builder, specify the accounting behavior by using the **UnitReportingMode** ECE Java enum. When the usage request builder is instantiated, the enum indicates to ECE whether to use incremental or cumulative accounting behavior.

Important: When there are multiple RUMs and attributes ROUND_UP and ROUND_DOWN of quantity in the rate plan, Granted Service Units that are reported on all attributes may be rounded up or down based on the rate plan configuration.

For both incremental and cumulative accounting, you must set attributes for the Requested_Units and Used_Units blocks in the payloads of applicable operation types. For example, the Requested_Units block is defined for the payloads of Initiate and Update operation types and the Used_Units block is defined for the payloads of Update, Update Accounting and Terminate operation types.

When configuring incremental or cumulative quota for usage requests, the metric name (RUMs) must be the same as the attribute name. For example, when sending attribute INPUT_VOLUME on the usage request, the RUMs must be defined with the same name.

Diameter Gateway uses incremental based accounting only.

About Usage Request Fixed Attributes

Usage requests contain a set of well known or *fixed attributes* that must be provided. Fixed attributes are required fields directly exposed by the UsageRequest interface. Fixed attributes are applicable for all the events in ECE.

You cannot pass in null for any of the fixed attributes. For non-duration requests, you can pass the same timestamp for both **requestStart** and **requestEnd**.

Fixed attributes within a usage request include the following:

- **userIdentity**

  The userIdentity attribute is the fixed attribute name representing the public user identity of the person or entity using the product (phone number, email address and so on). It is a generic way of identifying who is being charged for the usage.

- **requestId**

  The requestId is an identifier that uniquely identifies the usage interaction. If the usage is session based, the requestId must be the same across different operation types (Initiate, Update and Terminate). The requestId is used to locate the active session associated with the charging customer.

- **requestStart**

  The requestStart is the time at which the usage started.

  For session-based usage requests, ECE observes the requestStart value for Initiate operation-type usage requests.

- **requestEnd**

  The requestEnd is the time at which the usage ended.

  If the usage interaction has no duration, such as for event-based charging, the requestStart is equal to the requestEnd.

  Note: If the payload contains a non-null **"DURATION"** attribute (either as a top-level attribute or under a Requested Service Units (RSU) and Used Service Units (USU) block, its value will override the value of the **requestEnd** attribute.

- **requestMode**

  The requestMode defines the mode of the usage request. Valid values are **OFFLINE** and **ONLINE**.

  For backward compatibility, the default value is **ONLINE**.

- **sequenceNumber**

  The sequenceNumber is the sequential session-centric attribute and is a type of subID you can apply for different types of charging within a session.

  You cannot change the name of the fixed attributes.

  Usage requests also contain configurable (dynamic) attributes. Configurable attributes are defined in the payload blocks of the event definition (request specification data defined in PDC when you enrich event definitions).

About Charging Operation Types

The ECE API is designed to receive usage requests and send usage responses for common operation types in the charging industry.

ECE usage charging supports the following operation types:

Table 13–1 Charging Operation Types Supported by ECE

Operation Type	Description
Initiate	Commencement of a session-based charging operation.
Update	Continuation of a session-based charging operation.
Terminate	Conclusion of a single non-session based charging operation.
Cancel	Complete cancellation of a session-based charging operation.
Refund_Amount	Refund a specific amount to a specific balance resource.
Refund_Unit	Refund a calculated amount, based on units consumed, to the impacted resource(s).
Debit_Amount	Debit a specific amount to a specific balance resource.
Debit_Unit	Debit a calculated amount, based on Units consumed, to the impacted resource(s).
Price_Enquiry	Generate a price estimation without any balance reservations occurring. It is used when there isn't a high probability of receiving a charging request. For example, Price_Enquiry might be called to get the price of an event charge to display in a content portal.
Start_Accounting	Begin tracking usage without incurring balance impacts.
Update_Accounting	Continue tracking usage without incurring balance impacts.
Balance_Query	Return the user balance.
Accounting_On_Off	Clean left open session and reservation for a specific network element.

Each charging operation type requires an input payload that supplies fields which are relevant to the charging operation.

The BALANCE_QUERY operation type is used for query requests. The query request is built using the Query Request Builder.

The ACCOUNTING_ON and ACCOUNTING_OFF operation types are used for management requests. Management requests are built using the Management Request Builder.

For offline charging, requests are typically submitted for a single event that represents the entire charge (using the Terminate operation type). Session-based operations such as Initiate and Update are not as common for offline charging; however, these operation types are used when using a stream protocol like Radius or Rf in which ECE is used to record the consumption of resources (quantity consumption) as the session continues.

ECE processes charging operations by forwarding usage requests to the applicable combinations of charge, alteration and distribution rate plans. ECE creates the rate plan expressions required for usage charging by using fields which are supplied in the request specification payloads.

The sample request specification files demonstrate the data ECE requires to support the charging operation types.

Building Usage Requests

Your network mediation software receives network requests and transfers the relevant network-request fields from the network-request data structures into the usage-request

payloads. The network mediation software program uses the usage-request builder APIs for transferring the relevant network-request fields.

The usage-request builder uses ECE request specification data in the ServiceSpec repository initially so that it knows the format to use to build usage requests. The same instance of a usage-request builder can create usage requests for the lifetime of the network mediation client application for a specified request specification payload definition. For example, a usage-request builder creates the same Voice/Usage usage request repeatedly until you change the request specification data for a Voice/Usage usage request.

When you change the payload definitions of an event definition (to extend its associated usage request) and deploy the new definitions into the ECE cluster, the usage-request builder will need to get the changes of the newly deployed request specification data.

Consuming ECE Notification Data

The notification framework publishes external notifications, the data in which can be used by charging clients for further processing. For example, ECE can publish an external notification when a balance breaches a credit threshold value. The charging client application can use this information for sending an email to the customer informing the customer that a credit threshold has been breached.

See the discussion about notifications in *BRM Elastic Charging Engine Concepts* for information about the external notifications published by the notification framework.

Using Multiple Services Credit Control (MSCC)

ECE supports Multiple-Service Credit Control (MSCC) requests in which a Diameter application performs credit control for multiples services within the same session.

An MSCC request is a list of sub-requests that are targeted to the same customer, share the same operation type and session id, but individually apply to different products.

When ECE receives MSCC requests, it assigns a different session ID to each of its sub-requests. Doing this allows ECE to distinguish one sub-request from another when looking up the active session associated with each sub-request.

An MSCC request results in an MSCC response containing a sub-response for each sub-request. Each sub-response contains status indicating whether the sub-request succeeded or failed.

When ECE persists rated event information for MSCC requests in the Oracle NoSQL Database, note the following points:

- Rated event information is persisted for each sub-request.
- The NoSQL key for the rated event is based on the session ID that ECE assigned (not based on the original MSCC request session ID).
- The ECE session ID in the Oracle NoSQL Database is a composite of the original usage request's session ID, the user identity and the product type, separated by underscore characters. For example:

  Original MSCC requestId: **1313b2ab-d51e-4545-8bba-25c731daf10b**

  Usage request's userIdentity: **650123555**

  usage request's product type: **VOICE**

  ECE session ID: **1313b2ab-d51e-4545-8bba-25c731daf10b_VOICE_650123555**

MSCC support does not include support for rating groups (Rating-Group AVP), credit pools (G-S-U-Pool-Reference AVP where units of the service type are pooled in a credit pool) and credit control (as described in section 5.1.2 of IETF RFC 4006).

For for an example of how to send MSCC requests to ECE, see the SampleMultipleServicesLauncher sample program in the ECE SDK. For information about how to run ECE sample programs, see "About the ECE Sample Programs".

MSCC AVPs are part of the CCR and Diameter Gateway expects each Gy interface request type to be included in the MSCC group even if the request contains only a single service. When CCR is sent without MSCC AVPs, Diameter Gateway validates only the subscriber ID in the CCR and authenticates the subscriber.

Sample Programs for Integrating with Charging Clients

See "About the ECE Sample Programs" for information about sample programs that demonstrate how to use the ECE charging API.

14

Integrating Policy Clients with ECE

This chapter describes how policy clients that interact with the Policy and Charging Rules Function (PCRF) integrate with Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE).

> **Note:** The information in this chapter is relevant for those using third party network mediation software (for online charging) for receiving policy-driven-charging requests from the network and translating them into ECE requests.
>
> If you are using Diameter Gateway for receiving Diameter messages and translating them into ECE requests, see "Network Integration for Online Charging Using Diameter Gateway". Diameter Gateway constructs all requests that ECE supports for policy-driven charging using the ECE interfaces described in this chapter.

About Integrating Policy Clients with ECE

Policy clients integrate with ECE by using the ECE policy management APIs.

The policy client uses the ECE policy Sy interface to retrieve policy counter information from ECE. The policy client, in turn, sends the policy counter information to the Policy and Charging Rules Function (PCRF) using its Diameter Sy interface. As part of initiating a policy Sy session with ECE, the policy client subscribes for receiving notifications that contain the policy counter information.

The policy client uses the ECE policy Sp interface to retrieve customer preferences information from ECE. The policy client, in turn, sends the customer preferences information to the PCRF using its Diameter Sp interface. As part of initiating a policy Sp session with ECE, the policy client subscribes for receiving notifications that contain the customer preferences information.

About the ECE Sy and Sp Interfaces

To support policy-driven charging, ECE offers policy management APIs. The ECE Sy interface allows policy clients to subscribe for and retrieve the spending limit information about policy counters from ECE. The ECE Sp interface allows policy clients to subscribe for and retrieve customer preferences information that is relevant to policy enforcement from ECE.

The following sections describe each ECE interface:

- About the ECE Sy Interface

- About the ECE Sp Interface

ECE also supports a combined ECE Sy and Sp interface that allows policy clients to retrieve and subscribe for both types of information in one policy session. A combined ECE Sy and Sp interface reduces the number of messages between ECE and policy clients. See "About a Combined ECE Sy and Sp Interface" for information.

About the ECE Sy Interface

ECE supports the Sy interface which is used by the PCRF to retrieve policy counter information. To support the Sy interface, ECE offers the following ECE Sy procedure and notification:

- Spending Limit Report Request

  Policy clients use this procedure to request the status of policy counters available in ECE and to subscribe and unsubscribe (for the PCRF) to updates of ECE policy counters.

- Spending Limit Notification

  ECE uses this notification to report statuses of requested policy counters for one or more products and also report the results of request processing.

  The policy client transfers the status information to the PCRF.

About the ECE Sp Interface

ECE supports the Sp interface which is used by the PCRF to query customer preferences. To support the Sp interface, ECE offers the following ECE Sp procedures:

- Subscribe Notification Request

  Policy clients use this procedure to retrieve customer preferences and to subscribe and unsubscribe (for the PCRF) to updates of customer preference data changes.

  The customer preferences can include the following:

  - Customer's allowed services
  - Customer's allowed Quality of Service (QoS)
  - Customer's preferred channel for receiving notifications (such as receiving an SMS or email)
  - Customer's preferred language

- Subscribe Notification Response

  ECE uses this procedure to report updates to customer-preference data to the policy client that is subscribed for the notification.

- User Data Request

  Policy clients use the User Data Request procedure only to retrieve subscriber preferences without subscribing for receiving notifications when the preferences change.

- User Data Response

  ECE uses this procedure to send subscriber-preference data to the policy client.

The policy client transfers customer preference data to the PCRF.

Querying for Extended Subscriber Preference Information in Sp Query

The PCRF can also query extended information about customers and products (services). The policy client uses the ECE policy Sp query procedure to retrieve extended customer and extended product information.

To be able to query extended information from ECE using the policy Sp query request, you must configure the extended product and customer information in ECE.

To configure the extended product and customer information:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.
   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.policyConfig**.
3. Expand **Operations**.
4. Select **setDsl**.
5. Specify values for the following parameters:
   - **alias**: Enter the alias for the extended information to use in the policy query request.
   - **dsl**: Enter the DSL to use to retrieve the information from ECE in the following format:

     get*type***([product | customer]/***attribute with arguments***)**

     For example:

     getObject(product/lifeCycleStateName)
6. Click the **setDsl** button.

   This creates a mapping between the extended information alias with the DSL used to retrieve the extended information from customers and services.

The following are the ECE ready-to-use extended-information aliases (from the *ECE_home*/**oceceserver/config/management/charging-settings.xml** file):

```
<!--
Configure policy extended information for policy query
alias - Alias for the extended info which will be referred in the policy query
dsl - Abstract dsl in <get type>(<product or customer>/<attribute with arguments>)
format eg. getObject(customer/customerId)

-->
<policyConfig
config-class="oracle.communication.brm.charging.appconfiguration.beans.policy.Poli
cyConfig">
  <policyExtendedInfoGroup config-class="java.util.ArrayList">
    <policyExtendedInfo
      config-class="oracle.communication.brm.charging.appconfiguration.beans.polic
```

```
y.PolicyExtendedInfo"
      alias="lifeCycleStateName"
      dsl="getObject(product/lifeCycleStateName)"/>
    <policyExtendedInfo

      config-class="oracle.communication.brm.charging.appconfiguration.beans.polic
y.PolicyExtendedInfo"
      alias="publicIdentifiers"
      dsl="getObject(product/publicIdentifiers)"/>
    <policyExtendedInfo

      config-class="oracle.communication.brm.charging.appconfiguration.beans.polic
y.PolicyExtendedInfo"
      alias="ratingProfileFriends"
         dsl="getRatingProfile(product/ratingProfileByName?FRIENDS)"/>
    <policyExtendedInfo

      config-class="oracle.communication.brm.charging.appconfiguration.beans.polic
y.PolicyExtendedInfo"
      alias="ratingProfileCorporate"
           dsl="getRatingProfile(customer/ratingProfileByName?CORPORATE)"/>
  </policyExtendedInfoGroup>
</policyConfig>
```

About a Combined ECE Sy and Sp Interface

ECE supports combining its ECE Sp and Sy interfaces by offering the following procedures:

- Policy Session Request

  Policy clients use this procedure for retrieving Sp and Sy information and subscribing or unsubscribing (for the PCRF) to receiving updates to Sp and Sy data. This request is a combination of the Spending Limit Report Request and the Subscribe Notification Request.

- Policy Session Response

  ECE uses this procedure to report the information requested by the Policy Session Request and provide results of request processing.

  The policy client transfers the information to the PCRF.

Sample Policy Notification Payloads

ECE publishes notifications for active policy sessions into the ECE notification queue (JMS topic), such as spending limit notifications and subscriber preference notifications. The payloads are published as XML strings in the XML structure shown by the *ECE_home***/oceceserver/config/Notification.xsd** file.

Using the Policy Management API

To use the policy management API, clients call submitPolicy API with PolicyRequest.

For detailed information about the policy management API, see the documentation for the following packages in *BRM Elastic Charging Engine Java API Reference*:

- **oracle.communication.brm.charging.brs**
- **oracle.communication.brm.charging.messages.policy**

- **oracle.communication.brm.charging.messages.query**

Sample Programs for Policy Requests

For information about running sample programs that demonstrate how to use the ECE policy management API, see "About the ECE Sample Programs".

Sample programs print the response from ECE.

15

Integrating Top-Up Clients with ECE

This chapter describes how to integrate Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE) with top-up clients, such as those in a Third-Party top-up system.

Note: The information in this chapter is relevant for those using third party network mediation software (for online charging) for receiving top-up requests from the network and translating them into ECE requests.

If you are using Diameter Gateway for receiving Diameter messages and translating them into ECE requests, see "Network Integration for Online Charging Using Diameter Gateway". Diameter Gateway constructs top-up requests for ECE.

About Integrating Third-Party Top-Up Systems with ECE

ECE can receive top-ups from Third-Party top-up systems.

The ECE top-up API interfaces directly with top-up systems. The top-up systems send the top-up amount to ECE. ECE updates the customer balance in the cache and sends the update to BRM to update the customer balance in BRM.

ECE acts only as the Balance Management Function for the external top-up systems. The ECE top-up API does not perform authentication, authorization, or accounting operations as these AAA functions are handled by other systems.

Using the Top-Up API

To use the top-up API, clients call the submitUpdate API with ExternalTopupUpdateRequest.

For detailed information about the top-up API, see the documentation for the following in *BRM Elastic Charging Engine Java API Reference*:

- **oracle.communication.brm.charging.brs**
- **oracle.communication.brm.charging.messages.update**

Top-Up API Validation Behavior

Consider the following points for how the ECE top-up API validates top-up requests:

- For a currency resource, validity should not be passed as part of the request; if it is passed, it fails with an error. The customer's balance is expected to have one valid

currency balance item/bucket with infinite validity. If no balance item/bucket is present, a new one is created with infinite validity.

- Validity extend (**ValidityExtend**) is only for non-currency resources (given that currency resources do not have a validity).
- Whenever validity needs to be set for a resource, then both validity start and validity end are mandatory to be sent as part of the request.

  To create bucket with infinite validity, set both **validityStart** and **validityEnd** to **-1**.
- If the request is to extend validity and the customer balance has multiple valid balance items/buckets, then an error response is sent.
- If the request is to create a **firstUsage** bucket, then validity start and validity end should not be set in the request, except for the FirstUsageValidityUnit.ABSOLUTE mode.
- **ValidityExtend** is not allowed on a first-usage bucket.
- If both the validity and first-usage information (such as offset and unit) are specified as part of the request, then the top-up request fails with an error.
- Top-ups from Third-party top-up systems are not allowed when ECE is in a short-lived phase of the rerating process called the CATCH_UP phase.

  If top-up requests are sent during the CATCH_UP phase of rerating, ECE sends a failed response and includes the reason code for the failure in the response message. If that occurs, you can resend the top-up and ECE will process it.
- During testing, if a top-up request is sent to ECE with an event time that is earlier than the account creation time of the account to which the top-up applies, the balance is updated with the top-up in ECE but the balance is not updated in BRM. When you set event time stamps during testing, ensure the event time of the top-up request is later than the applicable account creation time.

Preventing Duplicate Top-Up Requests

Third-party top-up systems, such as voucher management systems, are used to eliminate duplicate top-up requests sent to ECE. However, duplicate top-up requests can occur internally in ECE, for example, when an unexpected ECE server shutdown occurs.

To detect duplicate top-up requests, ECE uses a combination of the Session ID field and the Recharge Reference field in the top-up CCR message.

As part of detecting duplicate top-up requests, ECE maintains a top-up history cache of the customer. The top-up history cache maintains a number of top-up Recharge Reference message IDs for each customer at any given time. The Recharge Reference message ID is required to be unique in the top-up history cache.

For top-up requests that have the same session ID, if the Recharge Reference message ID of an incoming top-up request is available in the history, then ECE considers the request to be a duplicate.

If ECE detects a duplicate top-up request, the following occurs:

- ECE does not apply the top-up
- ECE includes the following in the top-up response message:
  - The reason code DUPLICATE_REQUEST
  - The current customer balance

If you do not configure top-up request history for your implementation, ECE may not detect internal duplicate top-up requests. If ECE cannot detect an internal duplicate top-up request, the following occurs:

- ECE applies the top-up balance to the customer balance in its customer cache (as though it were a new request) and sends the top-up to BRM.
- BRM, which stores all top-up Recharge Reference message IDs in the BRM database, detects the top-up request as a duplicate and does not apply the top-up to the customer balance in the BRM database.
- BRM throws an error in the BRM Gateway log file as **Error from BRM: ERR_ DUPLICATE**.

  You would need to manually track such errors in the BRM Gateway log file since the ECE customer balance would no longer be synchronized with the customer balance in the BRM database.

You manage the number of top-up Recharge Reference message IDs to store in the top-up history cache by using the **external.topupConfig** ECE MBean.

To configure a top-up history cache:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.
   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **charging.externalTopUpConfig**.
3. Expand **Attributes**.
4. Set the **topUpHistoryCount** attribute to the number of top-up request message IDs to store for each customer.

   The default value is **3**.

   If you change the value of this attribute (for example, from 10 to 4) and the top-up history cache already contains 10 message IDs, ECE eliminates the message IDs of the oldest 6 top ups when the next top up arrives so that only the message IDs of the 4 most recent top-up requests are stored.

Sample Programs for Top-up Requests

For information about running sample programs that demonstrate how to use the ECE top-up API, see "About the ECE Sample Programs".

16

Integrating Query Clients with ECE

This chapter describes how client applications that need to query data in Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE) can integrate with ECE.

> **Note:** If you are using Diameter Gateway for receiving Diameter messages and translating them into ECE requests, see "Network Integration for Online Charging Using Diameter Gateway". Diameter Gateway constructs balance query requests for ECE.

About Integrating Query Clients with ECE

You can write client applications to query data in ECE, such as query the login and password information of a customer, or query the customer's account balance.

About Sending Authentication Queries

Use the authentication API to query the login and password of subscribers.

Use the login and password information for:

- Implementing authentication methods outside of the ECE charging server
- Allowing subscribers to validate their login and password credentials against a charge offer to which they are subscribed

About Sending Balance Queries

Use the balance API to query subscriber balances.

Use the subscriber balance for:

- Making policy decisions
- Sending the balance information to subscribers so they can monitor their network-usage expenses, validate their credit limit, or monitor their active reservation

For information about the data returned in the ECE balance query response, see the documentation for **oracle.communication.brm.charging.messages.query** in *BRM Elastic Charging Engine Java API Reference*.

About Balance Element ID Information in the ECE Balance Query Response

ECE returns the balance element numeric ID of each balance in the ECE balance query response. ECE returns the balance element numeric ID of balances for both

SUMMARY and DETAILED balance query modes. Client applications could use this information, for example, when customer balances are stored in multiple subscriber profile repositories and it is required to map the balances between the repositories.

About Grantor Information in the ECE Balance Query Response

ECE returns grantor information in the ECE balance query response for DETAILED balance query mode. Grantor information consists of the Grantor ID and the Grantor Type. The different grantor types include purchased charge offerings, purchased alteration offerings, charge offerings, and alteration offerings.

About Sending Price Estimation Queries

Use the PriceEnquiry operation type of the charging API to query price information.

For information about the PriceEnquiry operation type, see "About Charging Operation Types".

Using the Query APIs

For detailed information about the query APIs, see the documentation for the following packages in *BRM Elastic Charging Engine Java API Reference*:

- For the ECE authentication and query API: **oracle.communication.brm.charging.messages.query**
- For the ECE PriceEnquiry of the charging API: **oracle.communication.brm.charging.brs** and **oracle.communication.brm.charging.messages**

Sample Programs for Query Requests

For information about running sample programs that demonstrate how to use the ECE query APIs, see "About the ECE Sample Programs".

Part V

Testing an ECE Implementation

Part V describes how you can test an Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE) implementation. It contains the following chapters:

17

Overview of ECE Testing Tools

This chapter describes Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE) testing tools included with the ECE Software Development Kit (SDK) and the ECE data-loading utilities.

About ECE Testing Tools

You can use the following testing tools when implementing ECE in a charging system:

- simulator

  The simulator emulates the role of a client application, such as a network mediation software program, sending requests to ECE. See "About the Simulator".

- Customer file generator

  The customer file generator enables you to create customers without using Oracle Communications Billing and Revenue Management (BRM). See "About the Customer File Generator".

- Query tool

  The query tool provides access to ECE cache content by way of CohQL, a Coherence query utility that uses an SQL-like language. See "Using the Query Tool to Test ECE".

- Data-loading utilities

  See "Using Data-Loading Utilities".

- Performance MBean

  You can use the **PerformanceMonitor** MBean to monitor the performance of your ECE deployment during testing.

  See "About the Performance MBean".

When testing ECE, you can change ECE's current time and date, without affecting the operating system time and date. See "Changing Time and Date to Test ECE".

About the ECE SDK

When you install the ECE server software, the ECE SDK is installed in the *ECE_home*/**ocecesdk** directory. *ECE_home* is the directory in which ECE is installed.

The ECE SDK contains the following:

- Client libraries that enable your applications to connect to the cluster and build usage requests.

- Sample programs that demonstrate how to use the ECE client APIs for sending requests to ECE. See "About the ECE Sample Programs" for information about using the sample programs.

About the Simulator

The simulator emulates the role of a client application, such as a network mediation software program, sending requests to ECE. The simulator allows you to send usage requests, query requests, update requests, or policy requests to ECE for processing. You can run sample workloads for testing latency and throughput of your system. For information about how the simulator works and how to use it, see "Using the Simulator to Test ECE".

About the Customer File Generator

The customer file generator enables you to create customers without using BRM.

The customer file generator creates ECE customer XML files according to provided specifications.

You run the customer file generator from Elastic Charging Controller (ECC).

The customer file generator is an ECE node specified in the ECE topology file (*ECE_home*/**ocececeserver/config/eceTopology.conf**).

For information about using the customer file generator, see "Generating Customer Data for a Workload".

About the Query Tool

The query tool, **query.sh**, provides access to ECE cache content by way of CohQL, a Coherence query utility that uses an SQL-like language.

Important: The ECE data model within the Coherence cache is subject to change. Oracle does not recommend that client applications directly use the Coherence API or the query tool for accessing ECE cache data. For querying ECE cache data, write your client applications to use the ECE APIs such as the balance query and authentication query APIs.

You can use the query tool as follows:

- For debugging
- For development
- For learning about the ECE customer domain model
- For generating reports

For information about how to use the query tool, see "Using the Query Tool to Test ECE".

About Data-Loading Utilities

See the following topics for information about data-loading utilities:

About loader

Use the **loader** utility to load sample data into ECE caches required for processing requests. The **loader** utility loads sample pricing-related configuration data, sample pricing data, and sample customer data. The **loader** utility also loads sample request specification files that are used to validate requests that are sent by the simulator. For information about how to use the **loader** utility when testing ECE, see "Using the Simulator to Test ECE".

About configLoader

Use the **configLoader** utility to load the mediation specification data required for Diameter Gateway into ECE. See "About Configuration Data".

The **configLoader** utility loads the mediation specification file into configuration objects in the ECE cluster-replicated cache. In the *ECE_home*/**oceceserver/config/management/migration-configuration.xml** file, the **configObjectsDataDirectory** parameter specifies the path to the mediation specification file that is supplied (as ready-to-use configuration) when you install ECE. The mediation specification file is loaded into the ServiceSpec repository, a cache in the cluster where the mediation specification can be accessed by any Java Virtual Machine (JVM) process that is running in the cluster.

The **configLoader** utility also deploys the RADIUS mediation specification file into the ECE cluster.

Any previous RADIUS mediation specification that was in the ECE cluster is overwritten. See "Configuring the ECE System" in *ECE System Administrator's Guide* for more information.

About customerLoader

Use the **customerLoader** utility to load customer data into ECE from XML files.

In a test environment, use the **customerLoader** utility to load sample customer data into the ECE customer cache from XML files. See "About Customer Data".

In a production environment, use the **customerLoader** utility with the **-incremental** parameter to load customer data incrementally.

> **Caution:** Do *not* run the **customerLoader** utility without the **-incremental** parameter in a production environment.

About the ECE Customer XML Data Files

ECE customer XML data files are used by the **customerLoader** utility to load customer data into ECE.

ECE customer XML data files contain customer data in ECE XML data file format.

Use the customer file generator to create ECE customer XML files. See "Generating Customer Data for a Workload" for more information.

About the ECE Credit Profile XML Data Files

ECE credit profile XML data files are used by the **customerLoader** utility to load sample system wide credit profiles into ECE.

ECE includes sample system wide credit profiles in *ECE_home*/**oceceserver/sample_data/config_data/CreditProfile.xml**.

About pricingLoader

Use the **pricingLoader** utility to load sample pricing data into the ECE pricing cache from XML files.

Important: When you use Pricing Design Center (PDC), you run the ECE *Pricing Updater* for loading pricing data into ECE. Pricing Updater loads pricing data into ECE from the Java Message Service (JMS) queue to which PDC publishes the data.

Do *not* run both the **pricingLoader** utility and Pricing Updater on the same ECE system.

Use the **pricingLoader** utility only on a standalone installation when PDC is not used.

You start the **pricingLoader** utility by running an ECC command. When started, and when configured to access the XML files, the **pricingLoader** utility loads the data in the XML files into the ECE pricing cache and then stops.

It may be useful during testing or development to manually load pricing data from the XML file. For information about configuring the **pricingLoader** utility, see "Configuring pricingLoader".

About the Pricing XML Data File

The **pricingLoader** utility uses the same XML data format as PDC. You create the XML files by using the **ImportExportPricing** utility, a PDC utility. The pricing XML data file contains pricing data exported from the PDC database. For information about creating XML files for pricing data, see *PDC User's Guide*.

About the Pricing XML Schema File

The pricing XML schema file (XSD file) is the template for the pricing XML data file. The XML data file must conform to the structure of the schema file. PDC performs the XSD validation on the pricing XML files that it generates. XSD files are available in their respective directories in the *PDC_home*/**apps/xsd** directory, where *PDC_home* is the directory in which PDC is installed. For information about pricing XML schema files, see *PDC User's Guide*.

About Sample Programs

The ECE SDK includes sample programs that demonstrate how to use the ECE API for sending requests to ECE.

You use sample programs in the following ways:

- Use the sample programs as code samples for calling the ECE APIs.
- Use the sample programs as code samples for writing custom applications.
- Run sample programs to send requests to ECE and receive responses asynchronously.

  The sample programs print information on the messages exchanged.
- Use the sample program scripts as a guide for integration of the ECE client into your build system (Maven, Ant, and so on).

For information about how to use the sample programs, see "About the ECE Sample Programs".

Using Data-Loading Utilities

This section describes ECE data-loading utilities.

Using configLoader

This section describes how to configure the **configLoader** utility and use it to load the mediation specification configuration data required for Diameter Gateway into ECE.

Configuring configLoader

To configure the **configLoader** utility:

1. Verify the configuration in the mediation specification file.

   See "Editing the Mediation Specification File".

2. Open the *ECE_home*/**oceceserver/config/management/migration-configuration.xml** file.

3. Set the **configObjectsDataDirectory** parameter to the path of the directory that contains the ECE mediation specification file.

   The **configLoader** utility loads the mediation specification file from this directory into configuration objects in the ECE cluster-replicated cache.

   By default, this parameter is set to *ECE_home*/**oceceserver/sample_data/config_data**.

4. Save and close the file.

Loading Configuration Data with configLoader

To load configuration data with **configLoader**:

1. Change directory to the *ECE_home*/**oceceserver/bin** directory.

2. Start ECC:

   ```
   ./ecc
   ```

3. If your charging servers are not running, start them:

   ```
   start server
   ```

4. Run the following command:

   ```
   start configLoader
   ```

Using pricingLoader

This section describes how to configure the **pricingLoader** utility and use it to load the sample pricing data into the ECE pricing cache.

Important: When you use Pricing Design Center (PDC), you run the ECE *Pricing Updater* for loading pricing data into ECE. Pricing Updater loads pricing data into ECE from the JMS queue to which PDC publishes the data.

Do *not* run the **pricingLoader** utility and Pricing Updater on the same ECE deployment.

Use the **pricingLoader** utility only on a standalone installation when PDC is not used.

Configuring pricingLoader

To configure the **pricingLoader** utility:

1. Open the *ECE_home*/**oceceserver/config/management/migration-configuration.xml** file.
2. Set the **pricingDataDirectory** parameter to the path of the directory that contains the ECE pricing component XML data file.

   The **pricingLoader** utility loads the pricing data in the XML data files published to this directory into ECE caches.

   By default, this parameter is set to *ECE_home*/**oceceserver/config/management/sample_data/pricing_data**.
3. Save and close the file.

Loading Pricing Data with pricingLoader

To load sample pricing data with **pricingLoader**:

1. Change directory to the *ECE_home*/**oceceserver/bin** directory.
2. Start ECC:

   ```
   ./ecc
   ```
3. If the charging server nodes are not running, start them and load configuration data:

   ```
   start server
   start configLoader
   ```
4. Run the following command, which loads the pricing data:

   ```
   start pricingLoader
   ```

Using customerLoader

This section describes how to configure the **customerLoader** utility and use it to load the sample customer data such as credit profiles, offer profiles, and product cross-reference data from XML files.

Configuring customerLoader

To configure the **customerLoader** utility:

1. Open the *ECE_home*/**oceceserver/config/management/migration-configuration.xml** file.

2. Set the **configObjectsDataDirectory** parameter to the path of the directory that contains the credit profile and offer profile XML data files.

   The **customerLoader** utility loads the sample profile data from the XML data files in this directory into the ECE customer cache.

   By default, this parameter is set to *ECE_home***/oceceserver/sample_data/config_data**.

3. Set the **productOfferingCrossRefFilePath** parameter to the path of the directory for the product-offering cross-reference XML data file.

   The **customerLoader** utility loads the product-offering cross-reference data from the XML data files in this directory into the ECE customer cache.

   By default, this parameter is set to *ECE_home***/oceceserver/config/management/sample_data/customer_data**.

4. Set the **customerDataDirectory** parameter to one of the following:

   - To load the sample customer data files in ECE, specify the path of the directory for the ECE customer XML data file.

     The **customerLoader** utility loads the customer data in the XML data files published to this directory into the ECE customer cache.

     By default, this parameter is set to *ECE_home***/oceceserver/config/management/sample_data/customer_data**.

   - To incrementally load the customer data from BRM into ECE, specify the schema from which the data has to be loaded into ECE. The schema number and the corresponding schema name are found in the **connectionconfiguration** section of the *ECE_home***/config/management/charging-settings.xml** file. For example, if the schema number is 1, the corresponding schema name is customerUpdater1.

5. Set the **customerXmlPattern** parameter to the pattern of the customer XML data file name that you want **customerLoader** to load. The files with a name that starts with this pattern are loaded.

   By default, this parameter is set to **customer**.

6. Set the **remoteWmThreads** parameter to the number of parallel work manager threads to be used for the **customerLoader** utility.

   By default, this parameter is set to **1**.

7. Set the **batchSize** parameter to the number of customers to put into the repository in one put operation.

   Use **batchSize** to optimize performance of put operations into the repository. The more customers you insert into the repository in one put operation (rather than inserting each one individually), the better the performance.

   By default, this parameter is set to **5000**.

8. Save and close the file.

Loading Customer Data with customerLoader

Note: Do *not* use this procedure to load data in a production environment.

To load customer data:

1. Change directory to the *ECE_home*/**oceceserver/bin** directory.
2. Start ECC:

   ```
   ./ecc
   ```

3. If your charging server nodes are not running, start them and load configuration data and pricing data:

   ```
   start server
   start configLoader
   start pricingLoader
   ```

4. Run the following command, which loads the sample customer data:

   ```
   start customerLoader
   ```

Loading Customer Data Incrementally with customerLoader

Note: Do *not* run the **customerLoader** utility without the **-incremental** parameter in a production environment.

For incremental loading, the schema from which the loading has to be performed must be specified. On a system with multiple schemas, run the **customerLoader** utility for each schema.

The schema number and the corresponding schema name are found in the **connectionconfiguration** section of the *ECE_home*/**config/management/charging-settings.xml** file. For example, if the schema number is **1**, the corresponding schema name is **customerUpdater1**.

To load customer data:

1. Start ECC:

   ```
   ./ecc
   ```

2. If your charging server nodes are not running, start them and load configuration data and pricing data:

   ```
   start server
   start configLoader
   start pricingLoader
   start CustomerUpdater
   ```

3. Create prepopulated tables to load only preselected customer data. For more information, see "Creating Prepopulated Distribution Tables".
4. Run the following command, which loads the customer data incrementally:

   ```
   start customerLoader -incremental customer_updater_schema_name
   ```

   where *customer_updater_schema_name* is the schema name specified for Customer Updater in the **connectionconfiguration** section of the *ECE_home*/**config/management/charging-settings.xml** file.

   For Example:

```
start customerLoader -incremental customerUpdater1
```

About the Performance MBean

You can use the **PerformanceMonitor** MBean to monitor the performance of your ECE deployment. You can monitor the CPU usage of server nodes and client during your testing.

For example, when building charging extensions, you can run ECE without your extensions and use the methods to see how much CPU time is used. You can then run ECE with your extensions, and use the methods again to see how much CPU time is used. By comparing the CPU times, you can derive the additional time spent by your extension.

The following **PerformanceMonitor** MBean methods are available:

- The startTracking CPU() method starts tracking CPU usage for the running process.
- The stopTrackingCPU() method stops tracking CPU usage for the running process. This method returns CPU utilization between 0 and 1 where **0** means 0% CPU usage and **1** means 100% CPU usage.
- The getTrackedCPU() method gets the last tracked CPU usage between [0, 1] if a previously tracked CPU usage is available. If a previously tracked CPU usage is not available, **-1** is returned.

The simulator exposes the throughput information through the getLastThroughput() method. The getLastThroughput() method gets the throughput number from the last successfully completed simulation run. If completed simulation runs do not exist or if a simulation run is in progress, **-1** is returned.

Changing Time and Date to Test ECE

You can change ECE's current time and date, without affecting the operating system time and date, to test time-sensitive functions associated with Rated Event Formatter and Diameter Gateway in ECE.

> **Caution:** Changing the time and date introduces the possibility of corrupting data. Do not change the time and date in a production database.

For example, you can change ECE's current time and date to test the following:

- Whether accounts are billed correctly. If you advance the date in BRM to the next billing cycle to test if the accounts are billed correctly, you must advance the date in ECE to the same date as set in BRM. This ensures that the events rated by ECE on that day are sent to BRM with the same date as set in BRM so that the events can be billed for the next billing cycle.
- Whether customer's spending limit is reported correctly. If a charge offer includes a conditional balance impact and the conditional balance impact is valid only for a day, you can advance the date by a day to ensure that when the Spending-Limit-Report-Request (SLR/SLA) request is received, the spending limit for the next day is reported.

- Whether events are rerated correctly. You can advance the date in ECE to store rated events in the Oracle NoSQL database data store with the future date to ensure that they are rerated when rerating is run in BRM for the future date.

To change the time and date to test ECE:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.
   e. In the editor's MBean hierarchy, expand the **ECE configuration** node.
2. Expand **charging.server**.
3. Expand **Attributes**.
4. Specify the values for the following attributes:
   - **virtualTimeMode.** Enter one of the following values:
     - **0.** Use this to reset the time to ECE's current time. Time is changed to the operating system time.
     - **1.** Use this to set the time as a constant time. Time is frozen at the specified time until this value or the time you set is changed.
     - **2.** Use this to set the time to change every second. Time is changed to the time specified, and then advances one second every second.
   - **virtualTime.** Enter the time and date in the following format: *YYYY*-*MM*-*DD*T*HH*:*mm*:*ss*.*SSS*. For example, to set the time and date to 11:30:02:600 on September 03, 2016, enter 2016-09-03T11:30:02.600.

After you change the time and date, perform testing as needed. You can also change the time and date between testing stages. After completing testing, reset the time to ECE's current time and perform database cleanup if needed.

18

Using the Simulator to Test ECE

This chapter provides instructions for using the simulator when testing an Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE) implementation.

About the Simulator

The simulator emulates the role of a client application, such as a network mediation client application, sending requests to ECE. The simulator allows you to send usage requests, query requests, or update requests to ECE for processing.

For ECE to process requests, it must have data in its customer, pricing and configuration caches. The simulator works in conjunction with the loader tool which generates and loads sample data into ECE caches. The simulator can also be used with the data migration loaders as long as the customers being loaded are compatible by having numeric, sequential IDs.

The simulator randomly chooses different types of customers when sending requests and can control the number of requests sent per second for the entire data grid. You can specify the attributes of the customers in the requests before running each sample workload. For information about the attributes you can use to set up a workload, see *BRM Elastic Charging Engine Java API Reference BRM Elastic Charging Engine Java API Reference*. For information about how to edit parameters for the sample workload, see "Configuring the Simulator".

Configuring the Simulator

The simulator configuration is loaded through the *ECE_home*/**config/management/test-tools.xml** file under the simulator configuration section. You can configure this file directly by using a text editor *before* the charging servers are started or you can configure it by using a JMX editor, such as JConsole, *after* the charging servers are started.

To configure the simulator by using a JMX editor:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.
   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. If necessary, create a simulator instance as follows:
   a. Expand **test-tools.simulators**.
   b. Expand **Operations.**
   c. Select **addSimulatorConfiguration**.
   d. In the **name** parameter, enter a name for the simulator instance.
   e. Click the **addSimulatorConfiguration** button.
3. Expand **test-tools.simulators.***Instance_Name*, where *Instance_Name* is the name of the simulator instance you want to configure.
4. Expand **Attributes**.
5. Specify values for the simulator configuration attributes you want to change.

Configuring the Simulator for an ECE Integrated Implementation

The simulator is often used on a single server or standalone ECE installation. You can configure the simulator for an ECE integrated installation in which ECE is connected to BRM and simulates requests that will send charging results to BRM.

To configure the simulator for an integrated installation:

1. Create or extend your ECE request specification file.

   You can use pre-defined or custom request specification files.
2. Point the simulator to the request specification file you want to use.

   See "Pointing the Simulator to a Request Specification File".
3. Configure the simulator to populate the payload fields of the usage requests created. See "Configuring the Simulator to Populate Payload Fields".

Pointing the Simulator to a Request Specification File

To point the simulator to the request specification file you want to use:

1. Examine the request specification loaded into ECE.

   Request specifications are uniquely identified by a combination of two parameters:

   - **Version**
   - **EventType**

2. Configure the simulator by doing the following.

   You can do this online by using a JMX editor (setting attributes of ECE **Configuration:type=test-tools.simulator** MBean) or offline by editing the **test-tools.xml** file.

   a. Set the **ProductType** to match an available **ProductType** from the request specification file (for example, TelcoGsmTelephony).

   b. Set the **EventType** to match the one in the request specification file (for example, ConvergentVoice).

   c. Set the **Version** to match the one in the request specification file (for example, **1.0**).

Based on these three parameters, the simulator identifies the target request specification when building the requests. It creates requests with the payload fields defined in the request specification file to which it points. It does not populate payload fields. To populate payload fields, see "Configuring the Simulator to Populate Payload Fields".

Configuring the Simulator to Populate Payload Fields

By default, the simulator does not populate payload fields. The created payload will not have any values, except for the default ones. The mandatory sub-blocks (those with cardinality of at least 1) are created but not populated.

You must provide values for all mandatory fields. You may also want to provide values for some optional fields, or override the default ones.

To provide payload values:

1. Using a JMX editor, for each field you want configured, call the addPayloadAttribute method of ECE **Configuration:type=test-tools.simulator** Mbean.

2. Provide the name and value of the field you want populated in the request.

   Or using a text editor, edit the **test-tools.xml** file to include the fields you want.

With the payload attributes configured, when constructing requests, the simulator examines the request specification and places all relevant attributes into the payload.

For example, if a CALLED_ID attribute of the request payload has been set and the request specification defines a CALLED_ID attribute for Initiate requests only, all Initiate requests sent by the simulator have a payload with the specified CALLED_ID, but Terminate requests do not have a CALLED_ID attribute in their payload.

Using the Simulator with BRM

When you use the simulator for generating usage for an event type, and you want to load the generated rated events into BRM, ensure that you do the following when configuring the request specification file to be used with the simulator:

1. Declare your event type name as **"USAGE"**.

2. Associate the **"USAGE"** event type with an existing BRM event type by creating an external mapping. For example:

```
EventType "USAGE" -external "/event/delayed/session/telco/gsm"
```

By doing this, you can use the simulator to generate events that can be loaded into BRM during end-to-end testing of ECE with BRM.

About Running a Sample Workload

You run a sample workload by using the simulator. Running a sample workload involves sending a specified number of requests to ECE so that ECE can process the requests. When processing requests, ECE uses data (such as customer data and pricing data) loaded in its caches. Running a sample workload can reveal information about the throughput and the latency of your hardware and network equipment.

Loading Required Cache Data

After you install ECE and start the ECE charging server processes, the ECE caches in the Coherence grid are empty. You must load data in ECE caches that are required for processing requests.

You can use the simulator loader utility (**loader**) to load data or you can use other data-loading utilities such as the **configLoader**, **customerLoader**, and **pricingLoader** to load data. The **loader** utility loads customer, pricing, and configuration sample data.

Customizing Your Sample Workload

You can customize the configuration of your sample workload, such as increase the number of requests to send to ECE, by using the configuration service.

To customize your sample workload:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**ocecerver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.
   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **test-tools.common**.
3. Expand **Attributes**.
4. Specify values for the workload attributes you want to change.

   For descriptions of each attribute, see *BRM Elastic Charging Engine Java API Reference*.

Running a Sample Workload

You use the simulator to run a sample workload.

See "Configuring the Simulator" for instructions on configuring the simulator.

To run a sample workload:

1. Change directory to the **bin** directory.

   ```
   cd ECE_home/oceceserver/bin
   ```

2. Start the Elastic Charging Controller (ECC) application.

   ```
   ./ecc
   ```

3. Start the ECE charging servers.

   ```
   > start server
   ```

4. Start the loader.

   ```
   > start loader
   ```

   The loader generates and loads the sample data and loads the request specification files.

5. Start the simulator.

   ```
   > start simulator
   ```

6. Initialize the simulator.

   ```
   > init simulator
   ```

   The simulator obtains the public user identity map so that it can provide the BRS server with user ID routing information.

7. Run the sample workload.

   ```
   > simulate simulator
   ```

   It will take a few seconds to run the workload.

8. Look in the **invocation.log** file to see statistics about running that sample workload.

About the loader Utility

The **loader** utility is a data-loading utility used only when running a sample workload using the simulator. The **loader** loads customer, pricing, and configuration sample data that the simulator is designed to use for running sample workloads. The **loader** also loads sample request specification files that are used to validate requests that are sent by the simulator.

Note: The ECE sample programs do not work well with data loaded by the simulator **loader** utility.

Generating Customer Data for a Workload

The customer file generator is a utility that enables you to create customer data without using BRM. Use the customer file generator for generating customer data used for running a workload with the simulator. For information about customer file generator, see "About the Customer File Generator".

To generate customer data for a workload using customer file generator:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**oceceserver/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.
   e. In the editor's MBean hierarchy, expand the **ECE Configuration** node.
2. Expand **test-tools.customerGenerator**.
3. Do one of the following:
   - To use simple customer structures, configure the customer file generator to use one of the available customer configurations.

     The utility programmatically creates the customer structures according to the ECE customer data XSD file.
   - To create complex customer structures, configure the customer file generator to use the customer template.

     You manually set the mandatory fields, sequencing of tags, and so on as required by the ECE customer data XSD file.
4. Start ECC.

   ```
   ./ecc
   ```

5. Start the ECE charging servers.

   ```
   > start server
   ```

6. Start the customer file generator.

   ```
   > start customerGenerator
   ```

   The customer file generator generates the customer data files according to the customer configuration or customer template you configured.
7. Start the **customerLoader** utility.

   ```
   > start customerloader
   ```

   The **customerLoader** loads the customer data files into ECE.

19

Using the Query Tool to Test ECE

This chapter provides instructions for using the query tool when testing an Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE) implementation.

> **Important:** The ECE data model within the Coherence cache is subject to change. Oracle does not recommend that client applications directly use the Coherence API or the query tool for accessing ECE cache data. For querying ECE cache data, write your client applications to use the ECE APIs such as the balance query and authentication query APIs.

About the Query Tool

The query tool **query.sh** is a wrapper around the Coherence query utility CohQL which uses an SQL-like language. The query tool provides access to ECE cache content by way of CohQL, enabling you to execute queries on ECE Coherence caches. The query tool is meant to be used for debugging purposes only.

For information about how to use CohQL, see *Oracle Coherence Developer's Guide*:

http://docs.oracle.com/cd/E24290_01/coh.371/e22837/api_cq.htm#CDDIBCDH

With access to the ECE domain model Javadoc documentation, you can use the query tool for writing scripts that interact with the ECE domain objects, constructing CohQL queries. The query tool supports all ECE caches and objects.

The query tool program is included with the ECE Server software in *ECE_home*/**oceceserver/bin** where *ECE_home* is the directory in which ECE Server software is installed.

You can use the query tool as follows:

- For debugging
- For development
- For learning about the ECE Customer domain model
- For generating reports

To learn about query tool options, use the help command:

```
$ ./query.sh -h
```

Using the Query Tool

Use the query tool **query.sh** to write scripts for interacting with ECE domain objects.

For interactive use of the query tool, see "Using the Query Tool Interactively".

For non-interactive use of the query tool, see "Using the Query Tool Non-Interactively".

Using the Query Tool Interactively

The following shows interactive use of the query tool:

```
$ ./query.sh
Coherence Command Line Tool
CohQl> select count () from Customer;
Results
1000
CohQL> select key(), value().getCode().toString() from BalanceElement
Rusults
840, "USD"
```

Using the Query Tool Non-Interactively

The following shows non interactive use of the query tool:

```
$ ./query.sh -s -c -l "select sum(getAvailableBalance(\'USD\',null).getQuantity())
from Balance"
```

Query Tool Logfile

The query tool log file is *ECE_home*/**ocecesever/logs/query_out.log**.

Query Tool Command History

The query statement history is contained in *ECE_home*/**ocecesever/bin/.cohql-history**. You can use the up and down arrows to move through the command history.

Query Examples

This section provides examples for querying data in ECE using the query tool.

Query a Customer Balance

Following is an example of how to query a customer's balance. First, you query a specific customer to find the balance ID, and then you query a specific balance to find the balance element and balance amount.

> **Tip:** You can use the same model for querying a customer's active session object.

Step 1: Query the customer to find the balance ID

To query the customer and find the balance ID:

```
$ ./query.sh
```

```
Coherence Command Line Tool
CohQl> select key(), value() from Customer where key() = "Cust#6500000001"
Results
"Cust#6500000001",
 ###################################### Customer Begin
##########################################
CustomerImpl
{ customerId='Cust#6500000001
, inTransaction='null
, defaultBalanceId='Bal#6500000001
, externalReference='1
, version=0
, profiles={Birthday=[RatingProfileValueImpl{name=NUMBER, value=2013-08-21,
 validFrom=1970-01-01T00:00:00.000Z, validTo='292278994-08-17T07:12:55.807Z}]}
, subscriberPreferences={}
, subscribedPreferences=null
, AlterationSharingAgreements ={}
, DistributionSharingAgreements ={}
, productMap={Pro#6500000001=ProductImpl{
...
, balanceId = 'Bal#6500000001'
,         profiles = {FriendsAndFamily=[RatingProfileValueImpl{name=NUMBER,
 value=6501234567, validFrom=1970-01-01T00:00:00.000Z,
 validTo='292278994-08-17T07:12:55.807Z}]}
,         subscriberPreferences = {}
,         subscribedSpendingLimitCounters = {}
,         Life cycle state = 102
,         Life cycle Expiration time = 0
,         activeSessions = {}
,         debitRefundSessions = {}
,         audited purchased charge offerings = {}
,         audited purchased alteration offerings = {}
,         audited profiles={}
,         audited UsedAlterationSharingAgreements={}
,         audited UsedDistributionSharingAgreements={}
        balance=null
}}
, balances={}
,
 billingUnits={BillingUnit#6500000001=[BillingUnitImpl
{billingUnitId=BillingUnit#6500000001},
{accountingCycle=[Triple{first=2013-08-21T11:10:16.224-07:00},
{second=2013-09-21T11:10:16.224-07:00},{third=2013-10-21T11:10:16.224-07:00},]},
{billingCycle=[Triple{first=2013-08-21T11:10:16.224-07:00},
{second=2013-09-21T11:10:16.224-07:00},{third=2013-10-21T11:10:16.224-07:00},]},
{billingFrequency=1},]}
, auditedProducts={}
, auditedProfiles={}
, audited AlterationSharingAgreements={}
, audited DistributionSharingAgreements={}
, customerRerateProcessingInfo=CustomerRerateProcessingInfoImpl
{ RerateProcessingStatus='NOT_IN_RERATING, ReratingJobId='null}
 ####################Customer End ########################################
```

Step 2: Query the balance to find the balance element

To query the balance to find the balance element, you specify two components of the associated key (composite key) that links the customer to the balance.

```
$ ./query.sh
```

```
Coherence Command Line Tool
CohQl> select value() from Balance where key().getId() = "Bal#6500000001" and
 key().getAssociatedKey() = "Cust#6500000001"
Results
[BalanceImpl{BalanceId=Bal#6500000001}
{externalRevision=0}{OwnerId=1}
{BillingUnitId=BillingUnit#6500000001}
{BillingUnit=null}ActiveReservationMap{}}
balanceItemSpecs{{USD=BalanceItemSpecImpl{beCode='USD', unit=Money{cur=USD}
, creditProfile=oracle.communication.brm.charging.config.creditprofile.internal.
CreditProfileReference@1dc79d4
, consumptionRule=EARLIEST_START}}}
balanceItems{{[BalanceItemImpl{balanceItemId=0}{currentBalance=-10000}
{balanceItemSpec=BalanceItemSpecImpl{beCode='USD', unit=Money{cur=USD}
, creditProfile=oracle.communication.brm.charging.config.creditprofile.internal.
CreditProfileReference@1dc79d4
, consumptionRule=EARLIEST_START}{validity=null}{validityRule=null}]),]
```

Query the Subscriber Base Balance Summary

Here is an example of how to summarize balance amounts across the entire subscriber base (total balance) in the grid:

```
$ ./query.sh -s -c -l "select sum(getAvailableBalance(\'USD\',null).getQuantity())
from Balance"
```

20

Testing Scenarios in an Integrated System

This chapter describes testing scenarios for testing an Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE) implementation in an integrated charging system.

Verifying That Configuration Data Is Loaded into ECE

To verify that configuration data is loaded into ECE, perform a CohQL query on the RequestSpecification cache by using the ECE query tool (**query.sh**).

Verifying That Pricing Data Is Loaded into ECE

To verify that pricing data is loaded into ECE and stored in the replicated pricing caches, perform a CohQL query on pricing related caches by using the ECE query tool.

For a list of pricing cache names, see the *ECE_home*/**oceceserver/config/charging-cache-config.xml** file.

Verifying That Customer Data Is Loaded into ECE

To verify that customer data is loaded into ECE and stored in the distributed cache, perform a CohQL query on the Customer cache by using the ECE query tool.

Verifying That Rated Events Are Published to Oracle NoSQL Database

To verify that rated events are successfully published to the Oracle NoSQL database, you can do the following:

- In the Oracle NoSQL database data store directory, check the size of the data files. If the file size grows, this indicates that rated events were put into the Oracle NoSQL database.
- Check Oracle NoSQL log files; a performance-related log file has statistics about the operations such as the read and write operations.

You cannot verify that a particular rated event was published to the Oracle NoSQL database.

Disabling the Publishing of Rated Events to Oracle NoSQL Database

Some types of testing may not require publishing rated events to the Oracle NoSQL database.

To disable the publishing of rated events to the Oracle NoSQL database:

1. Open the *ECE_home*/**oceceserver/config/charging-cache-config.xml** file.
2. For the RatedEventPublisher module, change the cache-store configuration entry by replacing the following:

```
<init-param>
  <param-name>cache-store</param-name>

<param-value>oracle.communication.brm.charging.ratedevent.publisher.internal.coherence.RatedEventPublisher
                    </param-value>
```

   with this:

```
<init-param>
  <param-name>cache-store</param-name>

<param-value>oracle.communication.brm.charging.util.coherence.internal.NoPersistenceCacheStore
                    </param-value>
```

   The publishing of rated events to the Oracle NoSQL database is disabled.
3. Save the file.

Verifying That Customer Balances Are Updated in BRM

To verify that customer balances are being updated in Oracle Communications Billing and Revenue Management (BRM), you can call BRM balance opcodes.

Verifying That Rated Event CDR Files Are Created

With ECE and Rated Event Formatter running, event data record (EDR) files are created for the usage requests that ECE processes. When the BrmCdrPluginDirect Plug-in is configured, the EDR files are formatted in BRM call details record (CDR) format. To verify that they are created, check the directory to which you publish the CDRs; this is the input directory of the Rated Event (RE) Loader Batch Controller.

Verifying That ECE Notifications Are Published to the JMS Topic

To verify that ECE external asynchronous notifications are being published to the JMS topic, you can use the following sample SDK notification programs:

- **sample_jms_client.sh**
- **sample_jms_server.sh**

Use these sample programs to check the correctness of the JMS topic.

You can also use the **sample_jms_client.sh** sample program to check the messages produced from the ECE side to the JMS topic.

Disabling the Publishing of ECE Notifications to the JMS Topic

Some types of testing may not require publishing ECE external notifications to the JMS topic.

To disable external notifications:

1. Open the *ECE_home*/**oceceserver/config/charging-cache-config.xml** file.

2. For the ServiceContext module, change the cache-store configuration entry by replacing the following:

```
<init-param>
  <param-name>cache-store</param-name>

<param-value>oracle.communication.brm.charging.notification.internal.coherence.
AsynchronousNotificationPublisher
                    </param-value>
```

   with this:

```
<init-param>
  <param-name>cache-store</param-name>

<param-value>oracle.communication.brm.charging.util.coherence.internal.NoPersis
tenceCacheStore
                    </param-value>
```

   ECE external notifications are disabled.

3. Save the file.

Verifying an ECE Integrated System

Verifying an ECE integrated system involves verifying that ECE can process usage requests (rate events) using the ECE product and event definitions defined in the system's event definitions (request specification data published to ECE from Pricing Design Center (PDC)). To verify a custom implementation, you must create custom request specification files for all the products you offer your customers.

To verify an integrated system, you must set up only one product offering in PDC and one BRM customer account. After verifying the ECE integrated system, you must repeat certain tasks for all your product offerings to complete the integration.

Verifying That Usage Requests Are Processed in an Integrated System

You use the ECE simulator to send requests to ECE for processing. The simulator emulates network traffic coming from a network mediation system. You use the ECE query tool to verify that the usage has impacted the customer balance.

> **Note:** You can use Diameter Gateway or a third-party network mediation software program to send usage requests to ECE. This section describes how to use the simulator only.

The simulator enables you to control the types of usage requests sent and the number and type of subscribers sending the usage requests. See the discussion about using the simulator in *BRM Elastic Charging Engine Implementation Guide* for more information about using the simulator.

To verify that usage requests can be processed, see the following topics:

- Starting ECE Nodes in the Cluster
- Running the Simulator to Send Usage Requests
- Verifying That Balances Are Impacted in ECE

Starting ECE Nodes in the Cluster

To start all ECE charging server nodes in the cluster:

1. Log on to the driver machine.
2. Change directory to the *ECE_home*/**ocecesever/bin** directory.
3. Start the Elastic Charging Controller:

   ```
   ./ecc
   ```

4. Start the ECE nodes by running the following command:

   ```
   start
   ```

To verify that the ECE nodes are running:

1. Access the ECE MBeans:
   a. Log on to the driver machine.
   b. Start the ECE charging servers (if they are not started).
   c. Start a JMX editor, such as JConsole, that enables you to edit MBean attributes.
   d. Connect to the ECE charging server node set to **start CohMgt = true** in the *ECE_home*/**ocecesever/config/eceTopology.conf** file.

      The **eceTopology.conf** file also contains the host name and port number for the node.

   e. In the editor's MBean hierarchy, expand the **ECE State Machine** node
2. Expand **StateManager**.
3. Expand **Attributes**.
4. Verify that the **stateName** attribute is set to **UsageProcessing**.

   This means the ECE nodes are running.

Running the Simulator to Send Usage Requests

To run the simulator and send usage requests:

1. Start the ECE simulator:

   ```
   start simulator
   ```

2. Initialize the simulator:

   ```
   init simulator
   ```

3. Run the sample workload:

   ```
   simulate simulator
   ```

   The simulator will take a few seconds to complete processing the workload.

4. Open the **invocation.log** file located in *ECE_home*/**ocecesever**. You should see statistics for the sample workload.

Verifying That Balances Are Impacted in ECE

To verify that the usage requests impacted customer's balances, use the ECE query tool.

Query for Customer Balances in the Customer Cache

Here are two examples of how to query the customer cache to return the customer balances:

```
$ ./query.sh
Coherence Command Line Tool
CohQl> "select value() from Balance where ownerId='cccc'"

$ ./query.sh
Coherence Command Line Tool
CohQl> "select value() from Balance where balanceId='xxxx'"
```

In the results of the query that are returned, locate the following string:

```
{currentBalance=UnitValue{quantity=amount, unit=Money{cur=USD}
```

where *amount* is the quantity amount of the balance impact.

Verifying That Friends and Family Calls Are Processed

To verify that friends and family calls are processed:

1. Ensure the appropriate provisioning tag is available in BRM by doing the following:
   a. Ensure you define a provisioning tag that includes the **Friends&Family** extended rating attribute (ERA).

      The BRM installation comes with a sample provisioning tag. For information about creating a friends and family provisioning tag in BRM, see the discussion of working with extended rating attributes in *BRM Setting Up Pricing and Rating*.

   b. Ensure the provisioning tag in BRM contains the same profile specification labels provided in PDC.

      The profile specification labels that come in the PDC installation are **MYFRIENDS** and **MYFAMILY**. Specify these labels in the provisioning tag when using the profile specification labels in PDC.

   c. (Optional) If you create a new provisioning tag in BRM, rather than using the sample provisioning tag, run the **SyncPDC** utility to synchronize the provisioning tag name with PDC.
2. If not already loaded, load the sample profile attribute specification for friends and family into PDC.

   The sample XML file is available at *PDC_home***/apps/Samples/Examples/OOB_ProfileSpecifications.xml**, where *PDC_home* is the directory in which PDC is installed.

   Use the PDC **ImportExportPricing** utility to load the XML file into the PDC database.
3. If not already loaded, load the sample custom analyzer rule for friends and family into PDC.

   The sample XML file is available at *PDC_home***/apps/Samples/Examples/OOB_CRs.xml**.

   OOB_CRs.xml contains three custom rules: **Friends&Family** and **ClosedUserGroup** and **SpecialDay**. These custom rules are designed to be used specifically with generic selectors.

Use the PDC **ImportExportPricing** utility to load the XML file into the PDC database.

4. In PDC, configure the charge offer for friends and family calls by doing the following:
   a. Create a generic selector with the **Friends&Family** custom analyzer rule.
   b. Create the charge offer for the friends and family calling service.
   c. For the charge offer, select the provisioning tag that specifies **Friends&Family**.
   d. Create the charge for the charge offer.
   e. For the charge, include the generic selector with the **Friends&Family** rule.

      Tip: You associate the friends and family rule in the generic selector with a *result*: a string value that maps to the rule, such as **Friends&Family**. At runtime, ECE uses this result in the charge to apply different rates for calls to friends and family.

5. Verify that Pricing Updater is started.
6. Publish the PDC pricing data to the ECE rating engine.

   Pricing Updater synchronizes the pricing data to ECE.
7. Verify that ECE External Manager (EM) Gateway is started.
8. In BRM, create the customer account, purchase the charge offer, and configure the friends and family phone numbers.

   The BRM customer data updates are incorporated into the ECE cache in real time through EM Gateway. For more information, see the discussion about synchronizing BRM and ECE customer data in *BRM Elastic Charging Engine Concepts*.
9. Generate usage for a friends and family call for the customer.

   Use the ECE SDK sample programs to generate usage.
10. Verify that balances are impacted as expected.

After you verify that friends and family calls are processed as expected using the friends and family sample data in PDC and BRM, create your own friends and family configurations.

Verifying That Closed User Group Calls Are Processed

To verify that closed user group calls are processed:

1. Ensure the appropriate provisioning tag is available in BRM by doing the following:
   a. Ensure you define a provisioning tag that includes the **ClosedUserGroup** extended rating attribute (ERA).

      The BRM installation comes with a sample provisioning tag. For information about creating a closed user group provisioning tag in BRM, see the discussion of working with extended rating attributes in *BRM Setting Up Pricing and Rating*.
   b. Ensure the provisioning tag in BRM contains the same profile specification labels that are provided in PDC.

The profile specification label that comes in the PDC installation is **CLOSEDUSERGROUP**. Specify this label in the provisioning tag when using the profile specification labels in PDC.

c. (Optional) If you create a new provisioning tag in BRM, rather than using the sample provisioning tag, run the **SyncPDC** utility to synchronize the provisioning tag name to PDC.

2. If not already loaded, load the sample profile attribute specification for closed user group into PDC.

   The sample XML file is available at the following:

   - For service-based closed user group samples:

     PDC_home/**apps/Samples/Examples/Sample_ServiceCUG_ProfileSpecification.xml**

   - For customer-based closed user groups that work with sample data:

     PDC_home/**apps/Samples/Examples/OOB_ProfileSpecifications.xml**

   Use the PDC **ImportExportPricing** utility to load the XML file into the PDC database.

   Tip: Closed user group profiles are rating profiles (known as extended rating attributes in BRM) that have a closed-user-group affiliation. The closed-user-group affiliation is enabled by setting the **useDynamicIdentifier** field to **true** in the PDC profile attribute specification.

3. If not already loaded, load the sample custom analyzer rule for closed user group into PDC.

   The sample XML file is available at the following:

   - For service-based closed user group samples:

     PDC_home/**apps/Samples/Examples/Sample_ServiceCUG_CR.xml**

   - For customer-based closed user groups that work with sample data:

     PDC_home/**apps/Samples/Examples/OOB_CRs.xml**

     OOB_CRs.xml contains three custom rules: **Friends&Family** and **ClosedUserGroup** and **SpecialDay**. These custom rules are designed to be used specifically with generic selectors.

   Use the PDC **ImportExportPricing** utility to load the XML file into the PDC database.

4. In PDC, configure the charge offer for closed user group calls by doing the following:

   a. Create a generic selector with the **ClosedUserGroup** custom analyzer rule.

   b. Create the charge offer for the closed user group calling service.

   c. For the charge offer, select the provisioning tag that specifies **ClosedUserGroup**.

   d. Create the charge for the charge offer.

   e. For the charge, include the generic selector with the **ClosedUserGroup** rule.

Tip: You associate the closed user group rule in the generic selector with a *result*: a string value that maps to the rule, such as **ClosedUserGroup**. At runtime, ECE uses this result in the charge to apply different rates for calls to the closed user group.

5. Verify that Pricing Updater is started.
6. Publish the PDC pricing data to the ECE rating engine.

   Pricing Updater synchronizes the pricing data to ECE.
7. Verify that EM Gateway is started.
8. In BRM, for both the *calling customer* and the *called customer*, create the customer account, purchase the charge offer, and configure the required closed-user-group-profile information for the customer.

   For example, if the closed user group profile is at the customer level, specify the closed user group phone number. If the closed user group profile is at the service level, specify the closed user group name.

   The BRM customer data updates are incorporated into the ECE cache in real time through EM Gateway. For more information, see the discussion about synchronizing BRM and ECE customer data in *BRM Elastic Charging Engine Concepts*.
9. Generate usage for a closed user group call for the calling customer.

   Use the ECE SDK sample programs to generate usage.
10. Verify that balances are impacted as expected.

Once you verify that closed user group calls are processed as expected using the closed user group sample data in PDC and BRM, create your own closed user group configurations.

Verifying That Balance Impacts Are Assigned to Bill Items

To verify that balance impacts are assigned to bill items according to your business rules:

Important: Before loading item type selectors into PDC, make a backup copy of the customized **config_item_tags.xml** and **config_item_types.xml** files in BRM.

1. Ensure that a storable class for each bill item type is available in BRM.

   If you are verifying that ECE can apply balance impacts to a custom bill item, ensure the custom storable class is available in BRM. For example, **/item/custom**.
2. If not already loaded, load the item type selector into PDC.

   The item type selector contains item specifications and item type selector rules.

   You associate item type selector rules with an *item tag*: a string value that maps to the item type. At runtime, ECE evaluates your item-type-selector rule. The result of the rule evaluation is a unique *item type*. ECE assigns balance impacts to the bill item associated to the item type.

   Item-type-selector XML files are available at *PDC_home***/apps/Samples/Examples**.

See *PDC User's Guide* for information about using the XML files.

See *PDC User's Guide* for instructions on setting item-specification attributes and creating item-type-selector-rule expressions.

Use the PDC **ImportExportPricing** utility to load the item-type-selector XML file into the PDC database.

3. Verify that Pricing Updater is started.
4. Verify that EM Gateway is started.
5. In BRM, create the customer account and purchase the charge offer for the service associated with the bill items for which you are verifying bill-item assignment.

   The BRM customer data updates are incorporated into the ECE cache in real time through EM Gateway. For more information, see the discussion about synchronizing BRM and ECE customer data in *BRM Elastic Charging Engine Concepts*.
6. Generate usage for the customer that impacts the bill items for which you are verifying bill-item assignment.

   Use the ECE SDK sample programs to generate usage.
7. Run billing.
8. Verify that balance impacts are assigned to bill items as expected.

Verifying That Payloads in Request Specifications Are Correctly Formed

To debug rating errors, you may need to verify that payloads in request specifications are correctly formed. You can view request specifications in the RequestSpecification cache by using the following CohQL command and piping the contents to a file:

```
select toSpecFormat() from RequestSpecification
```

The RequestSpecification cache contains read-only information.

If you identify an issue with a payload in a request specification, correct the issue in PDC as follows:

1. Export the event object.
2. Update the XML with the corrections.
3. Re-import the event object.

   After the event object is re-imported, PDC re-publishes the event object, and Pricing Updater updates the request specification in ECE in the RequestSpecification cache accordingly.

Part VI

Implementation Utilities

Part VI describes Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE) utilities that are used when implementing ECE. Part VI contains the following chapter:

- ECE Implementation Utilities

21

ECE Implementation Utilities

This chapter provides reference information for Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE) utilities that are used for implementing ECE in a BRM charging system.

configLoader

Use the **configLoader** utility to load the mediation specification data required for Diameter Gateway into ECE. The mediation specification file enables Diameter Gateway to associate each Diameter request with its respective usage-request builder. Diameter Gateway uses the mediation specification file to determine which product and event type combination applies to an incoming Diameter request.

For more information, see "Network Integration for Gy Interface Requests".

The **configLoader** utility loads the mediation specification file into configuration objects in the ECE cluster-replicated cache.

For information about editing the mediation specification file, see "Editing the Mediation Specification File".

The **configLoader** utility supports incremental loading of configuration data held in mediation specification files. If you are updating the mediation specification file, running the **configLoader** utility loads only the new data.

The **configLoader** utility also deploys the RADIUS mediation specification file into the ECE cluster.

Any previous RADIUS mediation specification that was in the ECE cluster is overwritten. See "Configuring the ECE System" in *ECE System Administrator's Guide* for more information.

Location

ECE_home/**oceceserver/bin**

Syntax

```
{start|stop} configLoader [-help]
```

Parameters

start
Starts **configLoader**.

stop
Stops **configLoader**.

-help
Displays the syntax for **configLoader**.

Results

Look in the *ECE_home*/**oceceserver/logs/configLoader.log** file for errors.

To verify that the mediation specification configuration objects were loaded into the ECE cluster-replicated cache, use the query tool to form CohQL queries on the cache name that stores the respective object. For example, for mediation specification files, you would look in the MediationSpecification cache.

For more information, see "Testing Scenarios in an Integrated System".

Configuration

To configure the **configLoader** utility:

1. Verify the configuration in the mediation specification file.
2. Open the *ECE_home*/**oceceserver/config/management/migration-configuration.xml** file.
3. Set the **configObjectsDataDirectory** parameter to the path of the directory that contains the ECE mediation specification file.

   The **configLoader** utility loads the mediation specification file from this directory into configuration objects in the ECE cluster-replicated cache.

   By default, this parameter is set to *ECE_home*/**oceceserver/sample_data/config_data**.
4. Save and close the file.

customerLoader

Use the customerLoader utility to load customer data into ECE from XML files.

In a test environment, use the **customerLoader** utility to load sample customer data into the ECE customer cache from XML files.

In a production environment, use the **customerLoader** utility with the **-incremental** parameter to load customer data incrementally.

> **Caution:** Do *not* run the **customerLoader** utility without the **-incremental** parameter in a production environment.

ECE customer XML data files are used by the **customerLoader** utility to load customer data into ECE.

ECE customer XML data files contain customer data in ECE XML data file format.

Use the customer file generator to create ECE customer XML data files.

For more information, see the following topics:

- About Customer Data
- About Customer File Generator

Location

ECE_home/**oceceserver/bin**

Syntax

```
{start|stop} customerLoader [-incremental][customer_updater_schema_name] [-help]
```

Parameters

start
Starts **customerLoader**.

stop
Stops **customerLoader**.

-incremental *customer_updater_schema_name*
Incrementally loads customer from *customer_updater_schema_name*, the schema name specified for Customer Updater in the **connectionconfiguration** section of the *ECE_home*/**config/management/charging-settings.xml** file.

-help
Displays the syntax for **customerLoader**.

Results

Look in the *ECE_home*/**oceceserver/logs/customerLoader.log** file for errors.

To verify that the customer objects were loaded in the ECE cluster-distributed cache, use the query tool to form CohQL queries on the cache name that stores the respective object. For example, you would look in the Customer cache.

For more information, see "Testing Scenarios in an Integrated System".

Configuration

To configure the **customerLoader** utility:

1. Open the *ECE_home*/**oceceserver/config/management/migration-configuration.xml** file.
2. Set the **configObjectsDataDirectory** parameter to the path of the directory that contains the credit profile and offer profile XML data files.

   The **customerLoader** utility loads the sample profile data from the XML data files in this directory into the ECE customer cache.

   By default, this parameter is set to *ECE_home*/**oceceserver/sample_data/config_data**.
3. Set the **productOfferingCrossRefFilePath** parameter to the path of the directory for the product-offering cross-reference XML data file.

   The **customerLoader** utility loads the product-offering cross-reference data from the XML data files in this directory into the ECE customer cache.

   By default, this parameter is set to *ECE_home*/**oceceserver/config/management/sample_data/customer_data**.
4. Set the **customerDataDirectory** parameter to one of the following:
   - To load the sample customer data files in ECE, specify the path of the directory for the ECE customer XML data file.

     The **customerLoader** utility loads the customer data in the XML data files published to this directory into the ECE customer cache.

     By default, this parameter is set to *ECE_home*/**oceceserver/config/management/sample_data/customer_data**.
   - To incrementally load the customer data from BRM into ECE, specify the schema from which the data has to be loaded into ECE. The schema and file name associations are found in the **connectionconfiguration** section of the *ECE_home*/**config/management/charging-settings.xml** file.
5. Set the **customerXmlPattern** parameter to the pattern of the customer XML data file *name* that you want **customerLoader** to load. The files with a name that starts with this pattern are loaded.

   By default, this parameter is set to **customer**.
6. Set the **remoteWmThreads** parameter to the number of parallel work manager threads to be used for the **customerLoader** utility.

   By default, this parameter is set to **1**.
7. Set the **batchSize** parameter to the number of customers to put into the repository in one put operation.

   Use **batchSize** to optimize performance of put operations into the repository. The more customers you insert into the repository in one put operation (rather than inserting each one individually), the better the performance.

   By default, this parameter is set to **5000**.
8. Save and close the file.

pricingLoader

Use the **pricingLoader** utility to load sample pricing data into the ECE pricing cache from XML files.

> **Important:** When you use Pricing Design Center (PDC), you run the ECE *Pricing Updater* for loading pricing data into ECE. Pricing Updater loads pricing data into ECE from the Java Message Service (JMS) queue to which PDC publishes the data.
>
> Do *not* run both the **pricingLoader** utility and Pricing Updater on the same ECE system.
>
> Use the **pricingLoader** utility only on a standalone installation when PDC is not used.

Location

ECE_home/**oceceserver/bin**

Syntax

```
[start|stop] pricingLoader [-help]
```

Parameters

-help
Displays the syntax for **pricingLoader.**

Configuration

To configure the **pricingLoader** utility:

1. Open the *ECE_home*/**oceceserver/config/management/migration-configuration.xml** file.
2. Set the **pricingDataDirectory** parameter to the path of the directory that contains the ECE pricing component XML data file.

   The **pricingLoader** utility loads the pricing data in the XML data files published to this directory into ECE caches.

   By default, this parameter is set for loading sample pricing component data from the *ECE_home*/**oceceserver/config/management/sample_data/pricing_data** directory.
3. Save and close the file.

query.sh

The query tool provides access to ECE cache content, enabling you to execute queries on ECE Coherence caches. For information about using the query tool, see "Using the Query Tool to Test ECE".

Location

ECE_home/**ocecesever/bin**

Syntax

```
query.sh [help]
```

Parameters

-help
Displays the syntax for **query.sh**.

Results

Look in the *ECE_home*/**ocecesever/logs/query_out.log** file for errors.

Part VII

Appendices for Implementation Guide

Part VII includes appendices for this Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE) guide. It contains the following appendices:

- ECE API Reference

A

ECE API Reference

This appendix summarizes the Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE) application programming interfaces (APIs).

For detailed information about each API, see *BRM Elastic Charging Engine Java API Reference*.

> **Important:** The ECE data model within the Coherence cache is subject to change. Oracle does not recommend that client applications directly use the Coherence API or the query tool for accessing ECE cache data. For querying ECE cache data, write your client applications to use the ECE APIs such as the balance query and authentication query APIs.

About ECE APIs

Client applications such as online mediation system applications must obtain data relevant to the subscriber from ECE in order to process requests from the network. ECE offers services that expose its client APIs so that applications can obtain information from ECE for further processing.

See the following topics for information about services that expose the ECE APIs:

> **Note:** For detailed information about these services, such as roles, input and output parameters, and errors or exceptions, see the *BRM Elastic Charging Engine Java API Reference*.

- Authentication API
- Balance API
- Charging API
- Notifications API
- Policy Management API
- Custom Plug-in API
- Top-Up API

Authentication API

Use the authentication API to query the login and password of subscribers.

Use the login and password information for:

- Implementing authentication methods outside of the ECE charging server
- Allowing subscribers to validate their login and password credentials against a charge offer to which they are subscribed

For detailed information about the authentication API, see the documentation for **oracle.communication.brm.charging.messages.query** in *BRM Elastic Charging Engine Java API Reference*.

Balance API

Use the balance API to query subscriber balances.

Use the subscriber balance for:

- Making policy decisions
- Sending the balance information to subscribers so they can monitor their network-usage expenses, validate their credit limit, or monitor their active reservation

When building a balance query request, you have the option to use the following balance query modes to restrict the contents of the balance query response that ECE returns:

- SUMMARY

  When using the SUMMARY balance query mode, the balance query response contains only the total balance at the balance element level.

- DETAILED

  When using the DETAILED balance query mode, the balance query response contains the detailed balance for each of the balance elements; this includes balance element specification information (such as credit profile and threshold information) and reservation information (such as the active and consumed reservation).

The balance query response mode you use may impact the overall performance of your system.

For detailed information about the balance API, see the documentation for **oracle.communication.brm.charging.messages.query** in *BRM Elastic Charging Engine Java API Reference*.

Charging API

The ECE charging API supports the following operation types:

- Initiate
- Update
- Terminate
- Cancel
- Debit_Unit

- Debit_Amount
- Refund_Unit
- Refund_Amount
- PriceEnquiry
- StartAccounting
- UpdateAccounting
- AccountingOn
- AccountingOff

For detailed information about the charging API, see the documentation for the following packages in *BRM Elastic Charging Engine Java API Reference*:

- **oracle.communication.brm.charging.brs**
- **oracle.communication.brm.charging.messages**

For information about how the charging API is called, see "About Charging Operation Types".

Notifications API

The notification framework publishes external notifications, the data in which can be used by external applications for further processing. For example, ECE can publish an external notification when a balance breaches a credit threshold value. A client application could use this information for sending an email to the customer informing the customer that a credit threshold has been breached.

See the discussion about notifications in *BRM Elastic Charging Engine Concepts* for information about the external notifications published by the notification framework.

Policy Management API

Use the policy management API to support policy-driven charging.

ECE offers a policy management API for supporting Sp and Sy reference-point policy and control operations. For more information, see "Integrating Policy Clients with ECE".

For detailed information about the ECE policy management API, see the documentation for the following packages in *BRM Elastic Charging Engine Java API Reference*:

- **oracle.communication.brm.charging.brs**
- **oracle.communication.brm.charging.messages.policy**

Custom Plug-in API

Use the custom plug-in API to implement a custom plug-in for formatting rated events into the format required by an external system.

A sample custom plug-in is available in the ECE SDK package. The sample is called **SampleRatedEventFormatterCustomPlugin.java**.

For detailed information about the custom plug-in API, see the documentation for **oracle.communication.brm.charging.ratedevent.custom.CustomPlugin** in *BRM Elastic Charging Engine Java API Reference*.

Top-Up API

Use the top-up API to send top-up amounts to ECE for updating the customer's balance.

See "Integrating Top-Up Clients with ECE" for more information.

For detailed information about the top-up API, see the documentation for **oracle.communication.brm.charging.brs** in *BRM Elastic Charging Engine Java API Reference*.