Oracle® Communications Billing and Revenue Management

ECE Implementing Charging

Release 15.0
F86247-03
September 2024

Oracle Communications Billing and Revenue Management ECE Implementing Charging, Release 15.0

F86247-03

Contents

Part III Managing ECE Notifications

10 Configuring Notifications in ECE

11 Configuring Subscriber Preferences

Part IV Managing Charging Sessions

12 About Linear and Non-Linear Rating

13 Managing Midsession-Rated Events

14 Managing Online Charging Sessions

15 Managing Session Start and End Times

16 Managing Reservations for Online Sessions

20 Connecting ECE to a Diameter Client

21 Connecting ECE to a RADIUS Client

Preface

This guide describes how to implement charging in Oracle Communications Billing and Revenue Management (BRM) Elastic Charging Engine (ECE).

This guide has been updated to include changes and new feature content added for release 15.0.1.

Audience

This guide is intended for application administrators and charging experts who customize and administer ECE.

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.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1

About Configuring Charging in Elastic Charging Controller

You can implement charging in Oracle Communications Elastic Charging Engine (ECE).

Topics in this document:

- About Usage Charging in ECE
- About Integrating Applications with ECE
- About Customizing ECE
- About Balance Management in a Prepaid Session

For an overview of ECE, see "About Billing and Revenue Management" and "ECE System Architecture" in *BRM Concepts*.

For information about administering ECE, see "ECE System Administration" in *BRM System Administrator's Guide*.

About Usage Charging in ECE

You can use ECE to perform the following types of usage charging:

- *Online charging* rates events in real-time, such as during a prepaid call.
- *Offline charging* is used for batch rating of events, typically from post-paid telephone usage.

ECE receives events as usage requests for both online and offline charging. *Usage requests* contain the event data that ECE needs for rating. For example, to rate a phone call, ECE needs the number that made the call and the start and end times.

When ECE receives a usage request, it uses the data in the usage request, typically the phone number, to identify the customer. which in turn identifies the charge offer they own that is used to rate the event. In addition, the usage request includes the data needed for rating, such as the start and end times of the event.

Each type of service and event needs to be rated differently. For example, some events are rated by measuring duration, and some by measuring volume. When you configure events and services, you create event definitions that specify the data needed for charging the event. The event definitions are sent to ECE, and are stored in an ECE cache. For each incoming event, ECE uses the event definition to choose a usage request builder that creates the usage request.

A default set of event definition data is installed with ECE. If you create custom services and events, you can enrich event definitions in Pricing Design Center (PDC) and create customized event definitions.

Usage requests are created when network mediation clients submit data to the ECE Client:

- For online charging, a real-time online event, such as a prepaid call, is routed from the network to the Diameter Gateway, which uses the ECE Client to create a usage request. ECE processes the usage request, authorizes the call, and sends a usage response back

to the Diameter Gateway. As the call progresses, ECE also manages the interaction with the network to handle update requests, reauthorizations, and top-ups. After the call has ended, ECE rates the call.

In addition to prepaid calls, online charging can be used for any service that the subscriber connects to and uses in real time, such as broadband access, digital content, streaming radio, and cable television.

- For offline charging, call detail records (CDRs) are processed by Offline Mediation Controller, which handles mediation tasks and normalization. Offline Mediation Controller acts as an ECE client application to create a usage request, which ECE uses to rate the events.

  Offline charging is used for batch rating of events, typically from post-paid telephone usage. Offline Mediation Controller performs mediation and normalization tasks, such as checking for duplicate calls and assembling calls that arrive in multiple records.

After an event is rated, ECE sends the rated event data to the BRM database, and the customer's balance is updated in both ECE and in the BRM database. The same process is used for loading online charging events and offline charging events.

> **Note:**
>
> The complete available balance cannot be consumed to satisfy the beat when there is a beat set for the charging. There will be leftover balance which is not granted/ consumed.

About Integrating Applications with ECE

You can integrate external applications and network with ECE using the following:

- **Diameter Gateway**: Allows you to integrate ECE with a Diameter client. See "Connecting ECE to a Diameter Client".
- **HTTP Gateway**: Allows you to integrate ECE with a 5G network. See "Connecting ECE to a 5G Client".

  After the HTTP Gateway is set up, your 5G client can send usage requests to ECE through the ECE REST API for online charging, offline charging, or both. See REST API for Elastic Charging Engine.
- **RADIUS Gateway**: Allows you to integrate ECE with a RADIUS client. This allows ECE to authenticate access requests and process accounting requests for online charging when your customers use your terminal server or NAS to connect to ECE. See "Connecting ECE to a RADIUS Client".
- **ECE Java API**: to integrate client applications with ECE (for example, implement a top-up client). For offline charging, ECE is preintegrated with Oracle Communications Offline Mediation Controller. For online charging, ECE uses Diameter Gateway for network integration (for creating ECE requests for all supported request types). See "About the ECE API".

About Customizing ECE

You can customize ECE in the following ways:

- Use the ECE SDK sample programs. For example, it shows how third-party systems can perform direct top ups in ECE. See "ECE Sample Programs".
- Use ECE extensions to customize ECE and its gateways. For example, implement pre-rating and post-rating customizations. See "Customizing Rating".

About Balance Management in a Prepaid Session

In prepaid sessions, ECE and Diameter Gateway work together with the network system to manage an event as it occurs. When a subscriber starts a prepaid call, the network collects information about the customer and sends authentication and authorization requests to ECE. ECE processes the requests and returns the results immediately so the network can connect the call. While the session is in progress, ECE tracks the subscriber's balance to ensure that it is sufficient to pay for the call.

ECE performs the following functions:

1. Authenticates customers by comparing the customer's ID with those stored in ECE.
   - For telco services, the ID is typically the MSID.
   - For broadband services, the ID is typically a login name and password.
2. Authorizes customers to use the service. ECE can perform these checks:
   - **Credit limit checking**. Determines whether the customer's account balance exceeds the temporary or permanent credit limit.
   - **Service status checking**. Confirms that the requested service is active in the customer's account.
   - **Duplicate session checking**. Checks for duplicate sessions.
3. Reserves a balance amount for the session. For example, customers can be authorized to download 100 bytes of data or to make a 30-minute telephone call.

   To reserve an amount, ECE does the following:

   - Receives the requested amount from the network and determines whether the user has a sufficient amount in his balance, based on the charge offers and discount offers that he owns and any amounts already reserved.
   - If the balance amount is insufficient, ECE calculates the maximum authorization based on the customer's credit limits. The effects of discounts, discount sharing, and charge sharing are included in the calculation of the maximum amount to authorize.

     ECE sends the validity time for the active reservation or *reservation validity* to the network mediation client. Reservation validity specifies how long a session can continue before the client must ask for a reauthorization.

     ECE sends a *reservation expiration* to the network mediation client. Reservation expiration specifies how long a session can continue before the client must report the consumed usage to ECE.

   When a prepaid session is authorized, BRM reserves a portion of the customer's balance for the event. This prevents customers from using that balance amount for other services while the session is in progress.

   BRM authorizes a customer to use a service for the following:

   - A specified duration or volume.
   - (Volume-based authorizations only) A specified validity period.

4. When the session ends, ECE sends information about the rated event to the BRM database and updates the subscriber's balance in the BRM database. ECE then returns any unused reserved balance amounts to the customer's balance.

   Active session and balance reservations are checked for expiration and are removed if the object has expired. Usually, only the terminated or canceled charge offer is cleaned up. However, other charge offers owned by the same customer and that share the balance object with the original charge offer are also cleaned up.

During a session, ECE does the following:

- Reauthorizes customers for extended usage if necessary. Reauthorization for prepaid services extends the following:
  - Authorized duration or volume
  - Validity period
- Alerts the network that a change that might require reauthorization occurs in a customer's account. For example, the customer is granted a balance amount that might apply to the current session. This is called *server-initiation reauthorization*.
- Cancels authorization for failed connections. After a session is authorized, the external network can sometimes not connect to the service. This can occur for the following reasons:
  - The call's destination was unavailable.
  - The validity period expired before the service was connected.
  - The customer terminated the session before the service was connected.

  In this situation, ECE can cancel the authorization and return any reserved balances to the customer's account.
- Updates balances in the customer's account.
- If your system is configured to receive in-session notifications from BRM (that is, when the **piggyback** business parameter is enabled), it appends specific in-session notifications to the responses it provides for authorization and reauthorization requests sent by a supported network connectivity application. You can configure customer preferences for sending notifications (for example, by SMS in a specific language). See "Configuring Subscriber Preferences".
- If the session uses policy-driven charging, it tracks the balance thresholds that trigger credit-limit notifications to the policy controller. Both in-session and out-of-session notifications are supported.

Part I
Using the ECE Java API

This part provides information about using the Oracle Communications Elastic Charging Engine (ECE) Java API. It contains the following chapters:

2

About the ECE API

You use the Oracle Communications Elastic Charging Engine (ECE) API to integrate ECE with third-party clients, such as top-up systems.

Topics in this document:

About the ECE API

ECE is preintegrated with Oracle Communications Offline Mediation Controller. To integrate other clients, such as top-up systems, you use the ECE SDK and ECE API. See Elastic Charging Engine Java API Reference for information.

The ECE SDK includes:

- Client libraries that enable your applications to connect to ECE and build usage requests.
- Sample programs that demonstrate how to use the ECE APIs.

ECE SDK includes a set of sample programs. You use these sample programs in the following ways:

- Use the sample programs as code samples for writing custom applications.
- Run sample programs to send requests to ECE and receive responses.
- 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 "ECE Sample Programs".

The ECE SDK is installed in *ECE_home*/**ocecesdk**.

About the 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

To send usage requests to ECE, client applications can call the ECE charging APIs according to the usage request builder defined by the ECE event definition.

For details about the charging API, see **oracle.communication.brm.charging.brs** and **oracle.communication.brm.charging.messages** in Elastic Charging Engine Java API Reference.

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 operation types shown in Table 2-1.

Table 2-1 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.

About the Authentication API

Use the authentication API to query the login and password of subscribers.

Use the login and password information to do the following:

- Implement authentication methods outside of the ECE charging server
- Enable subscribers to validate their login and password credentials against a charge offer to which they are subscribed

For details about the authentication API, see **oracle.communication.brm.charging.messages.query** in Elastic Charging Engine Java API Reference.

About the Custom Plug-in APIs

You can use the custom plug-in APIs to:

- Format rated events into the format required by an external system. To do so, use the **SampleRatedEventFormatterCustomPlugin.java** sample custom plug-in in the ECE SDK package.
- Write rated events in JSON format to the Kafka Server. To do so, use the **SampleRatedEventFormatterKafkaCustomPlugin.java** sample plug-in in the ECE SDK package.

For more information, see **oracle.communication.brm.charging.ratedevent.custom.CustomPlugin** in Elastic Charging Engine Java API Reference.

3

Configuring Multiple Services Credit Control

You can configure Multiple Services Credit Control in Oracle Communications Elastic Charging Engine (ECE).

Topics in this document:

About Multiple Services Credit Control

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 subrequests targeted to the same customer that share the same operation type and session ID but that individually apply to different charge offers.

When ECE receives MSCC requests, it assigns a different session ID to each of its subrequests. Doing this enables ECE to distinguish one subrequest from another when looking up the active session associated with each subrequest. An MSCC request results in an MSCC response containing a subresponse for each subrequest. Each subresponse contains a status indicating whether the subrequest succeeded or failed.

If you configured ECE to save the rated event information for MSCC requests in the Oracle NoSQL database, note the following:

- Rated event information is saved for each subrequest.
- The NoSQL key for the rated event is based on the session ID that ECE assigned (not 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 service, and the user identity, separated by underscore characters. For example:

  Original MSCC request ID: **1313b2ab-d51e-4545-8bba-25c731daf10b**

  Usage request's service: **VOICE**

  Usage request's user ID: **650123555**

  ECE session ID: **1313b2ab-d51e-4545-8bba-25c731daf10b_VOICE_650123555**

MSCC support applies to usage requests and query requests.

MSCC support does not include support for credit pools (G-S-U-Pool-Reference AVP where units of the service are pooled in a credit pool) and credit control (as described in section 5.1.2 of IETF RFC 4006).

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 a CCR is sent without MSCC AVPs, Diameter Gateway validates only the subscriber ID in the CCR and authenticates the subscriber.

See the **SampleMultipleServicesLauncher** sample program in the ECE SDK for an example of how to send MSCC requests to ECE. For more information, see "ECE Sample Programs".

4

Advice of Charge and Advice of Promotion

You can configure Advice of Charge (AOC) and Advice of Promotion (AOP) in Oracle Communications Elastic Charging Engine (ECE).

Topics in this document:

About Advice of Charge

ECE supports the 3GPP Advice of Charge (AoC) supplementary service by which customers can be informed about the cost for a requested service either in monetary format or nonmonetary format. AoC may be provided at the beginning of a session, during a session or at the end of a session.

To support AoC, ECE calculates the cost of using a service and relays that information to the network mediation software program, which can then pass the message to the customer.

About Advice of Promotion

ECE enables you to provide Advice of Promotion (AoP) information to customers to notify them that a better price can be obtained for a service they are about to use. For example, a network operator can send the AoP information in an IVR pre-call announcement for a Voice service.

To support AoP, ECE determines whether better pricing for a service is available near the time the customer's usage request is received. ECE sends that information to the network mediation software program, which sends a message 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 balance, or to generate a price estimation for using a balance.
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).
4. If the recalculated charge is less expensive for the customer, ECE sends the information about potential savings to the network mediation software program 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.

Note the following details about AoP:

- AoP is not configurable in PDC.
- AoP is a systemwide configuration (it is not configured on a per charge offer basis).
- The ready-to-use configuration of AoP gives advice based on time.

- When applying AoP, ECE uses the charge offers and discount offers eligible when the request is received to recompute the charge for the offset time period. If a different charge offer or a different discount offer applies to the future offset time period, AoP may advise a promotion when none exists or may not advise a promotion when a promotion is available.

When using AoP, ensure that your charge offers have tiered consumption configured accurately to prevent a credit breach of noncurrency balances.

To configure Advice of Promotion:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.server**.
4. Expand **Attributes**.
5. 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 ten 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.

5

Configuring Top-Ups

You can configure top-ups in Oracle Communications Elastic Charging Engine (ECE).

Topics in this document:

Integrating Top-Up Clients

ECE interfaces directly with top-up systems to manage balances. The top-up systems send the top-up amount to ECE, and then ECE updates the customer balance and sends the update to the BRM server.

ECE does not manage top-up authentication, authorization, or accounting operations.

For information about running sample programs that demonstrate how to use the ECE top-up API, see "ECE Sample Programs".

To configure top-up notifications, see "About ECE Notifications".

Detecting Duplicate Top-Up Requests

Duplicate top-up requests might occur within ECE as a result of node failures (for example, if an ECE server unexpectedly shuts down, ECE might internally resubmit a top-up request when restarted). To detect and eliminate these internal duplicate top-up requests, ECE maintains a top-up history cache.

> **Note:**
>
> Third-party top-up systems, such as voucher management systems, are expected to eliminate duplicate top-up requests coming from the network.

The ECE top-up history cache maintains a specified number of top-up message IDs for each customer. If the message ID of an incoming top-up request is already in the history, 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 retain a sufficient number of top-up request message IDs in your top-up history cache, 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 as though it were a new request and sends the top-up balance to the BRM server.
- The BRM server, which stores all top-up 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 adds the **Error from BRM: ERR_DUPLICATE** error to the BRM Gateway log file.

  You must manually track such errors in the BRM Gateway log file and correct the customer balance in ECE because the ECE customer balance is no longer synchronized with the customer balance in the BRM database.

To configure the number of top-up request message IDs in your top-up history cache:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.externalTopUpConfig**.
4. Expand **Attributes**.
5. 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.

Using the Top-Up API

To validate top-up requests, clients call the submitUpdate API with ExternalTopupUpdateRequest.

Consider the following points for how the ECE top-up API validates top-up requests:

- For a currency balance, 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 noncurrency balances (given that currency balances do not have a validity).
- Whenever validity must be set for a balance, both validity start and validity end must 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 response that includes the reason code for the failure. 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.

For details about the top-up API, see the documentation for **oracle.communication.brm.charging.brs** and **oracle.communication.brm.charging.messages.update** in Elastic Charging Engine Java API Reference.

6

Configuring Balance Queries

You can configure third-party clients to query Oracle Communications Elastic Charging Engine (ECE) for balance information.

Topics in this document:

Integrating Balance Query Clients

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.

ECE returns the balance element ID of each balance in the ECE balance query response. ECE returns the balance element ID of balances for SUMMARY, DETAILED, ALL, and TURBO 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.

To use the query APIs:

- 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**

For information about running sample programs that demonstrate how to use the ECE query APIs, see "ECE Sample Programs".

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
- Enabling 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 balances for one or more subscribers.

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

ECE returns the balance element numeric ID of each balance in the ECE balance query response. 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.

Configuring 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 charge offer. 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 charge offer. By default, debit request information is stored for ten debit request operations (for each charge offer) at any given time. For example, if the debit map contains ten entries 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.

To configure the debit request history:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.server**.
4. Expand **Attributes**.
5. 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.

About Balance Query Requests

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: The balance query response contains the balance element ID and the total balance at the balance element level.
- DETAILED: The balance query response contains the balance element ID and the detailed balance for each balance element. This includes balance element specification information (such as credit limits), grantor information (such as grantor ID and grantor type), reservation information (such as the active and consumed reservation), and rollover balance information (such as the rollover balance and the number of times the balance has been rolled over).
- ALL: The balance query mode specifies to return all the sub-balances after a particular time stamp. It is in the EM Gateway response.

- TURBO: The balance query mode specifies to return information about the balance element level, the current principal balance, and the loan balance. It is in the Diameter Gateway response.

The balance query response mode you use may impact the overall performance of your system.

For details about the balance API, see the documentation for **oracle.communication.brm.charging.messages.query** in Elastic Charging Engine Java API Reference.

Part II
Working with BRM

This part provides information about how Oracle Communications Billing and Revenue Management (BRM) and Oracle Communications Elastic Charging Engine (ECE) work together. It contains the following chapters:

- Synchronizing Data Between ECE and the BRM Database
- Loading ECE Rated Events into BRM
- Generating POIDs for Rated Events

7

Synchronizing Data Between ECE and the BRM Database

You synchronize data with the Oracle Communications Billing and Revenue Management (BRM) database to ensure that Oracle Communications Elastic Charging Engine (ECE) is rating service usage events with the most current pricing data.

Topics in this document:

About Synchronizing Data Between BRM and ECE

When customer data is updated in the BRM database, the updates must be applied synchronously (in real time) to ECE. For example, when a CSR adds, cancels, or modifies an account or when an adjustment such as a cycle fee is applied to an account balance, that information must be updated in ECE so that ECE can properly rate service usage events.

The Oracle Data Manager (DM) synchronizes customer data between BRM and ECE. The Oracle DM sends customer data in a business event to an Advanced Queuing (AQ) database queue, where ECE Customer Updater retrieves the business event and updates the information in ECE cache, allowing ECE to rate events using the latest customer information. For more information, see "Synchronizing Account Data Between BRM and ECE" in *BRM System Administrator's Guide*.

You can view or modify the list of BRM business events that are sent to ECE by using the *ECE_home*/**brm_config/payloadconfig_ece_sync.xml** file.

Setting Up Synchronization Between BRM and ECE

To set up your system to synchronize customer data between BRM and ECE:

1. Create and configure your AQ database queues. See "Configuring Your AQ Database Queues" in *BRM System Administrator's Guide*.
2. Configure BRM to send account data updates to ECE in real time. See "Enabling Real-Time Synchronization of BRM and ECE Customer Data Updates".
3. Configure BRM to retrieve real-time balances for a service from ECE. See "Configuring the Connection Manager to Get Real-Time Balances for a Service from ECE".

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).

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 0.0.0.0
-cm em_group ece PCM_OP_ECE_PUBLISH_EVENT
-cm em_pointer ece ip emGateway_host emGateway_port
```

   where:

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

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

3. If the publisher database number of EM Gateway in your system is not 0.0.0.0, replace it with the correct publisher database number in the following entry:

```
-cm ece_real_time_sync_db_no 0.0.9.8
```

4. Save and close the file.
5. If you changed the EM Gateway publisher database number in your CM **pin.conf** file, do the following:
   a. Open the *BRM_home*/**sys/eai_js/payloadconfig_ifw_sync.xml** file in a text editor (or the merged file if you merged payload configuration files).
   b. Locate the **PublisherDefs** section.
   c. In the **Publisher DB="***DB_number***"** entry, replace *DB_number* with the publisher database number of your EM Gateway.
6. Save and close the file.
7. If you changed the EM Gateway publisher database number, restart the Payload Generator External Module (also called the Enterprise Application Integration (EAI) Java Server or **eai_js**).
8. Restart the CM.

Configuring the Connection Manager 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
```

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

How ECE Gets Historical Data from the BRM Database

Because there is a gap of time between when a call occurs and when it is rated, information about the customer can change during that time. For example, a customer might change the phone number before a call is rated. ECE must look up account data based on the old number.

To retrieve historical information, ECE gets data from audited objects. By default, auditing in BRM is turned off for most objects. After you install the account synchronization components, you must run the **object_auditing.pl** script to turn on auditing for the objects and fields that ECE needs data about. See "object_auditing.pl" in *BRM System Administrator's Guide*.

8

Loading ECE Rated Events into BRM

You load rated events from Oracle Communications Elastic Charging Engine (ECE) into the Oracle Communications Billing and Revenue Management (BRM) database by using Rated Event Loader.

Topics in this document:

About Sending Rated Events to the BRM Database

After usage events are rated, ECE sends the rated event data to the BRM database by using Rated Event Loader and updates the customer's balance in both ECE and in the BRM database. The same process is used for loading online charging events and offline charging events.

For more information, see "About Loading Rated Events into the BRM Database" in *BRM Loading Rated Events*.

Adding a Rated Event Publisher Instance

If you are using Oracle NoSQL Database to store rated events, you must configure Rated Event Publisher. Rated Event Publisher publishes ECE-generated rated events to the Oracle NoSQL database data store.

To add a Rated Event Publisher instance:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.ratedEventPublishers**.
4. Expand **Operations**.
5. Select **addRatedEventPublisherConfiguration**.
6. Enter a value for the instance **name** parameter.
7. Click **addRatedEventPublisherConfiguration**.
8. Use Elastic Charging Controller (ECC) to start the RE Publisher instance.

Configuring Rated Event Publisher

To configure Rated Event Publisher:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.connectionConfiguratons.***instance_name*.
4. Specify values for the fields in Table 8-1.

Table 8-1 Fields for Configuring Rated Event Publisher Connection

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 preconfigured 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 preconfigured data store in an Oracle NoSQL database system. The default is **kvstore** for using a standalone Oracle NoSQL database system (KV-Lite).
name	noSQLConnection1	This parameter should match the **noSQLConnectionName** parameter in the **charging.ratedEventPublishers.***instance_name* configuration.

5. Expand the **charging.ratedEventPublishers.***instance_name* node.
6. Expand **Attributes**.
7. Specify values for the fields in Table 8-2.

Table 8-2 Fields for Configuring Rated Event Publisher

Name	Default	Description and Guideline
noSQLConnectionName	noSQLConnection1	This parameter should match the **name** parameter in the **charging.ratedEventPublishers.***instance_name* configuration for the correct connection configuration to the Oracle NoSQL database.

Table 8-2 (Cont.) Fields for Configuring Rated Event Publisher

Name	Default	Description and Guideline
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.

8. Use ECC to stop and restart Rated Event Publisher.

Configuring Item Assignment for Rated Events

You configure item assignments in ECE so that customer balance impacts can be tracked appropriately. Typically, the default configuration is sufficient. If you have custom item assignments, you might need to change the configuration for item assignments.

The item-type field maps to BRM items; this mapping is required for loading rated events from ECE to the BRM database. Each rated event record has an **item_type** field derived from the mapping specified in the **itemType** MBean attribute. The **itemType** MBean attribute lists the ECE service/event combinations used in event definitions.

For example, usage events are typically applied to the **/item/misc** object, also known as the **misc** item type. To map voice and data events to the **misc** item, ECE maps itemType="misc" to itemTag="VOICE_DATA_misc". The XML file that stores this configuration shows how the mapping works.

```
<itemTypeDetail itemType="misc" itemTag="VOICE_DATA_misc">
   <itemTagDetail
      productType="VOICE"
      eventType="USAGE">
   </itemTagDetail>
   <itemTagDetail
      productType="DATA"
      eventType="DATA_USAGE">
   </itemTagDetail>
</itemTypeDetail>
```

In this example, the VOICE_DATA_misc item tag includes two ECE service/event mappings: VOICE/USAGE and DATA/DATA_USAGE. When a usage request is created, the item mapping specifies that the **misc** item type will be assigned to the events.

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 for rated events:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".

2. Expand the **ECE Configuration** node.
3. Expand **charging.itemAssignmentConfig**.
4. Expand **Attributes**.
5. 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 reserved at ECE startup for each schema appears.
   - **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 delayed billing interval is set in the **ConfigBillingDelay** business parameter. See "Configuring Delayed Billing" in *BRM Configuring and Running Billing*.
6. 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 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 lifecycle state feature. If the subscriber lifecycle 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 the custom subscriber life cycle, which has the following states: Preactive, Active, Recharge Only, Credit Expired, Fraud Investigated, Dormant, Suspended, and Closed. See "Creating Custom Service Life Cycles" in *BRM Managing Customers* for more information.

You must configure the lifecycle states in ECE, so they stay synchronized with lifecycle states you add in BRM.

To configure life cycle states in ECE for BRM during runtime:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.lifecycleConfiguration**.
4. Expand **Operations**.
5. Select **addLifecycleDetails**.
6. Enter values for these parameters:

- **ProductType**: (String value) The product type for which you want to configure the rule and transition, such as VOICE, DATA, or SMS.
- **Rule**: (String value) The rule for the specified product type to allow or disallow usage. The characters need to be escaped while provided the same input field.
- **Transition**: (String value) The state transitions for the specified product type.
- **IsUsage**: (Boolean value) True specifies that the rule and transition is for USAGE. False specifies that the rule and transition is for EXTERNAL_TOP_UP.

7. Click the **addLifecycleDetails** button.

Including or Excluding a Customer's Remaining Balance in Rated Events

ECE can send the current balance and loan balance information with rated events. This allows your custom client application to display balance data with rated events. These balances can be either currency or non-currency, depending on the type of resource the transaction impacts.

By default, this value is set to **true** to send the balances with events. You can change the value to **false** if you want to retain the older event format.

To configure whether ECE sends balance data on rated events:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.server**.
4. Set the **populateCurrentLoanAmountsOnRef** attribute to either **true** or **false** as desired.

Accessing ECE Configuration MBeans

For all configurations, start by accessing the ECE configuration MBeans:

1. Log on to the driver machine.
2. Start the ECE charging servers (if they are not started).
3. Connect to the ECE charging server node enabled for JMX management.
   This is the charging server node set to **start CohMgt = true** in the *ECE_home***/config/eceTopology.conf** file, where *ECE_home* is the directory in which ECE is installed.
4. Start a JMX editor that enables you to edit MBean attributes, such as JConsole.
5. In the editor's MBean hierarchy, find the ECE configuration MBeans.

9

Generating POIDs for Rated Events

You can configure Oracle Communications Elastic Charging Engine (ECE) to generate POIDs for events that are created in ECE.

Topics in this document:

About Generating POIDs in ECE

You use portal object IDs (POIDs) to track rated events and bill items. For tracking events created in ECE, POIDs are generated as follows:

- For delayed events, ECE generates the POIDs by default.
- For prepaid events, BRM generates the POIDs and sends them to ECE by default. You can configure ECE to generate the POIDs. See "Configuring ECE to Generate POIDs for Prepaid Events".
- For non-usage events, such as subscription events, BRM generates the POIDs and sends them to ECE.

ECE uses Rated Event Formatter to generate the POIDs and persists the last allocated POID ID in the database. This ensures that the POIDs are generated without any duplication even if the ECE system is restarted.

The POID generated in ECE contains the following information:

```
event_type date cluster_id BRM_schema_id unique_id
```

See Table 9-1 for the description of each entry in the POID.

Table 9-1 POID Entries in ECE

Entry	Description
event_type	A unique 4-bit number assigned to each event type. For example, 0 is assigned to subscription events, 1 is assigned to postpaid events (USAGE_POSTPAID), and 2 to 7 is assigned to prepaid events (USAGE_PREPAID) depending on the **prepaidParttitionSet** value specified in BRM. The default value for *event_type* is 0.
date	The 16-bit date on which the POID is generated. The date is determined based on ECE **virtualTime** if it is enabled. For more information on **virtualTime**, see "Changing Time and Date to Test ECE".

Table 9-1 (Cont.) POID Entries in ECE

Entry	Description
cluster_id	A unique 4-bit number assigned to the Coherence cluster to identify ECE in the cluster. The *cluster_id* is limited to 0 to 15 and the maximum number of ECE clusters allowed in a deployment is 16. The default value for *cluster_id* is 0. If ECE is configured for disaster recovery, you must specify the cluster ID for each cluster used in the Active-hot standby or Active-cold standby systems.
BRM_schema_id	A unique 6-bit number assigned to the BRM schema. The *BRM_schema_id* is limited to 0 to 31.
unique_id	A unique 34-bit number assigned to each POID.

You can configure multiple instances of Rated Event Formatter for uninterrupted POID allocation. If the primary Rated Event Formatter instance fails, the secondary Rated Event Formatter instance ensures that the POIDs are generated and allocated without any interruption. In a disaster recovery deployment, if the Rated Event Formatter instance in the *primary* site fails, the Rated Event Formatter instance in the *backup* site continues the POID allocation for the events.

For tracking the bill items and non-usage events created in ECE, ECE uses the POIDs received from BRM. ECE persists the POID pool received from BRM in the database. This ensures that the reserved POID pool is retained in ECE even after the ECE restart. It allows ECE to continue the POID allocation using the existing POID pool.

Configuring ECE to Generate POIDs for Prepaid Events

To configure ECE to generate POIDs for prepaid events, you must perform the following:

1. Enable prepaid-event partitions in BRM. See "Enabling Prepaid Event Partitions in BRM".
2. Ensure that the cluster ID is configured for ECE clusters. The cluster ID must be specified if you have ECE configured for disaster recovery. See "Configuring Cluster ID".
3. Ensure that the name of the primary Rated Event Formatter instance is specified in each Rated Event Formatter instance. See the **primaryInstanceName** MBean attribute in "Configuring RE Formatter" in *Loading Rated Events*.

   The primary Rated Event Formatter instance must be specified if you have ECE configured for disaster recovery.
4. Enable POID generation for prepaid events in ECE. See "Enabling POID Generation for Prepaid Events in ECE".

Enabling Prepaid Event Partitions in BRM

To enable prepaid-event partitioning in BRM:

> **Note:**
>
> In multischema systems, perform this task first on the primary BRM installation machine and then on the secondary BRM installation machines.

1. Open the *BRM_home*/**sys/dm_oracle/pin.conf** file in a text editor.
2. Set the **prepaid_partition_set** entry to a numerical value only between **2** and **7**. For example:

   ```
   - dm prepaid_partition_set 2
   ```

3. Set the **prepaid_partition_transition_mode** entry to **1**:

   > **Note:**
   >
   > Setting this entry to **1** enables Data Manager to retrieve the partitions for the existing events. After retrieving all the partitions for the existing events (for example, after 90 days), set this entry to **0** to disable this mode.

   ```
   - dm prepaid_partition_transition_mode 1
   ```

4. Save and close the file.
5. Create an editable XML file from the **system** instance of the **/config/business_params** object:

   ```
   pin_bus_params -r BusParamsSystem bus_params_system.xml
   ```

6. Set the **prepaidPartitionSet** parameter to the value you specified in step 2. For example:

   ```
   <prepaidPartitionSet>2</prepaidPartitionSet>
   ```

7. Save the file as **bus_params_system.xml**.
8. Load the XML file into the BRM database:

   ```
   pin_bus_params bus_params_system.xml
   ```

9. Stop and restart the CM.
10. Go to the *BRM_home*/**apps/partition_utils** directory.
11. Enable prepaid-event partitions by running the following command:

    ```
    partition_utils -o enable -t prepaid
    ```

12. Add prepaid-event partitions by running the following command:

    ```
    partition_utils -o add -t prepaid -s start_date -u month|week|day -q quantity
    ```

    where:

    - *start_date* specifies the starting date for the new partitions. The format is MMDDYYYY.
    - *quantity* specifies the number of partitions to add. Enter an integer greater than 0.

    For more information on enabling and adding partitions, see "Partitioning and Managing BRM Database Tables" in *BRM System Administrator's Guide*.

Configuring Cluster ID

To configure the cluster ID for ECE clusters:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.

3. Expand **charging.clusters.***Cluster_Name*, where *Cluster_Name* is the name of the ECE cluster that you are configuring.
4. Expand **Attributes**.
5. Set the **id** attribute to a unique number that indicates the ID of the cluster in the POID generated in ECE.

   Rated Event Formatter uses the cluster ID in the POID to identify the ECE clusters. The cluster ID must be unique for each cluster.

Enabling POID Generation for Prepaid Events in ECE

To enable POID generation for prepaid events in ECE:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.brmCdrPlugins.***Instance_Name*, where *Instance_Name* is the name of the BrmCdrPluginDirect Plug-in instance you are configuring.
4. Expand **Attributes**.
5. Set the **prepaidPartitionSet** attribute to the value that you specified in the **prepaid_partition_set** entry in the *BRM_home***/sys/dm_oracle/pin.conf** file.

> **Note:**
>
> To enable POID generation in ECE, you must set this attribute to a number between **2** and **7**. If this attribute is set to **0**, ECE uses the POIDs received from BRM for tracking events.

Part III
Managing ECE Notifications

This part provides information about generating notifications for your customers or external systems from Oracle Communications Elastic Charging Engine (ECE). It contains the following chapters:

- Configuring Notifications in ECE
- Configuring Subscriber Preferences

10

Configuring Notifications in ECE

You can configure Oracle Communications Elastic Charging Engine (ECE) to publish in-session notifications to subscribers and system notifications to external applications.

Topics in this document:

About ECE Notifications

ECE supports these types of notifications:

- **In-session notifications for your subscribers**: These notifications are sent to customers during online charging. For example, notifying customers about their reserved balance amounts.

  You can configure in-session notifications for individual subscribers by using subscriber preferences. See "Configuring Subscriber Preferences".

- **Notifications for external applications**: These notifications contain information that external applications need. For example:
  - The network mediation system can use data from the external notification in conjunction with customer policy data to implement network policy control.

    ECE sends notifications for external applications to the ECE Notification queue. You must configure your external application to retrieve and process notifications from the ECE Notification queue.
  - BRM can use data in the external notification to run billing for a specific customer.

    ECE sends BRM notifications to the ECE Notification queue. You can configure BRM to retrieve notifications from the queue and send them to the BRM Server for processing.

All notifications are disabled by default. You can configure ECE to do the following:

- Publish in-session notifications to your subscribers
- Publish notifications to external applications, such as BRM
- Publish specific notification types to a dedicated Kafka topic
- Specify the type of notifications that are supported

- Send notifications to everyone in a sharing group when one of its members triggers a notification
- Configure BRM Gateway to retrieve notifications from the ECE Notification queue

Enabling ECE to Publish Notifications to External Applications

To enable ECE to publish notifications to external applications such as BRM, do the following:

1. Open the *ECE_home*/**config/charging-cache-config.xml** file.
2. Under the ServiceContext module, set **cache-store** to **oracle.communication.brm.charging.notification.internal.coherence.AsynchronousNotificationPublisher**:

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

3. Save the file.

Enabling Specific Notification Types

You can enable ECE to generate notifications for specific event types, such as exceeding a credit limit.

You specify whether a specific notification type is enabled and whether it can be sent to an external application, a subscriber, or both by setting its attribute to one of the values in Table 10-1.

Table 10-1 Notification Attribute Values

Attribute Value	Description
NONE	This notification type is disabled.
ASYNCHRONOUS	Send asynchronous notifications to external applications.
PIGGYBACK	Send in-session notifications to subscribers through the usage response message.
ASYNC_PIGGYBACK	Send both asynchronous notifications to external applications, and in-session notifications to subscribers through the usage response message.

To enable ECE to generate specific types of notifications:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.notification**.
4. Expand **Attributes**.
5. Specify which type of notifications can be sent to your customers, external applications, or both by setting the attributes in Table 10-2 to **NONE**, **ASYNCHRONOUS**, **PIGGYBACK**, or **ASYNC_PIGGYBACK**.

> **Note:**
>
> An asterisk (*) appears next to the default attribute value.

Table 10-2 Notification Types

Attribute Name	Description	Supported Values
creditCeilingBreachNotificationMode	Sends notifications when a customer's balance has breached a credit ceiling.	**NONE*** **ASYNCHRONOUS** **PIGGYBACK** **ASYNC_PIGGYBACK**
creditFloorBreachNotificationMode	Sends notifications when a customer's balance has breached a credit floor.	**NONE*** **ASYNCHRONOUS** **PIGGYBACK** **ASYNC_PIGGYBACK**
thresholdBreachNotificationMode	Sends notifications when a customer's balance has breached a credit threshold.	**NONE*** **ASYNCHRONOUS** **PIGGYBACK** **ASYNC_PIGGYBACK**
topUpNotificationMode	Sends notifications when a customer's balance is topped up.	**NONE*** **ASYNCHRONOUS** **PIGGYBACK** **ASYNC_PIGGYBACK**
rarNotificationMode	Sends notifications when an ongoing session requires a reauthorization request. See "Enabling Server-Initiated Reauthorization Requests".	**NONE*** **ASYNCHRONOUS**
externalTopUpNotificationMode	Sends notifications when a customer's balance is topped up through an external application.	**NONE*** **ASYNCHRONOUS**
billingNotificationMode	Sends notifications when a subscriber starts a charging session near the time billing is set to run in BRM. This ensures that billing generates new recurring grants and charges on time.	**NONE*** **ASYNCHRONOUS**
adviceOfChargeNotificationMode	Sends notifications to support the Advice of Charge (AoC) supplementary service.	**NONE*** **ASYNCHRONOUS** **PIGGYBACK** **ASYNC_PIGGYBACK**
replenishPoidIdNotificationMode	Sends notifications when ECE needs to obtain POID IDs from BRM.	**NONE** **ASYNCHRONOUS***
lifeCycleTransitionNotificationMode	Sends notifications when a customer's life-cycle state has changed.	**NONE*** **ASYNCHRONOUS**
firstUsageValidityInitNotificationMode	Sends notifications to synchronize the validity of first usage balance elements from ECE to external applications.	**NONE*** **ASYNCHRONOUS**
offeringUsageValidityInitNotificationMode	Sends notifications to synchronize the validity of offer usage balance elements from ECE to external applications.	**NONE** **ASYNCHRONOUS***

Table 10-2 (Cont.) Notification Types

Attribute Name	Description	Supported Values
spendingLimitNotificationMode	Sends notifications when a threshold for a policy-driven charging rule has been reached.	**NONE*** **ASYNCHRONOUS**
aggregatedSpendingLimitNotificationMode	Sends aggregated notifications when a threshold for a policy-driven charging rule has been reached.	**NONE*** **ASYNCHRONOUS**
subscriberPreferenceUpdateNotificationMode	Sends notifications when a subscriber's notifications are updated.	**NONE*** **ASYNCHRONOUS**
rerateJobCreateNotificationMode	Sends notifications when a rerate job is created.	**NONE** **ASYNCHRONOUS***
customEventNotificationMode	Sends notifications when a custom event occurs.	**NONE*** **ASYNCHRONOUS**
enrichBalanceQueryResponseMode	Enriches the ECE balance query response with the subscriber's preferences.	**NONE*** **PIGGYBACK**
loanGrantNotificationMode	Sends notifications when a customer is granted a loan.	**NONE*** **ASYNCHRONOUS** **PIGGYBACK** **ASYNC_PIGGYBACK**
automaticTopUpTriggerNotificationMode	Sends notifications when a customer's balance requires an automatic top up.	**NONE*** **ASYNCHRONOUS** **PIGGYBACK** **ASYNC_PIGGYBACK**
customBrmOpCodeEventNotificationMode	Sends notifications when a custom event requires a call to a BRM opcode.	**NONE*** **ASYNCHRONOUS**

Publishing Asynchronous Notifications to Multiple Kafka Partitions

You can configure ECE to replicate one or more asynchronous notification types and publish them to multiple partitions in the ECE Notification topic. This allows you to send the same notification to numerous external applications, with each application consuming notifications from a dedicated partition. For example, ECE could replicate a threshold breach notification and then:

- Publish the original notification to Partition 1, which is connected to BRM
- Publish the replicated notification to Partition 2, which is connected to Oracle Communications Convergent Charging Controller

To configure ECE to publish a notification to multiple partitions in a Kafka topic:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.notification**.
4. Expand **Attributes**.

5. In the **eventListForInternalExternalPublish** attribute, enter the list of notification types to replicate and publish to multiple partitions. Separate multiple notification types with a comma. The default is OFFERING_VALIDITY_INITIALIZATION_EVENT and EXTERNAL_TOP_UP_NOTIFICATION_EVENT.

   Table 10-3 lists the valid notification types.

Table 10-3 Notification Types

Notification Type	Description
SPENDING_LIMIT_NOTIFICATION_EVENT	Generated when a threshold for a policy-driven charging rule has been reached.
SUBSCRIBER_PROFILE_UPDATE_EVENT	Generated when a subscriber preference has been created, modified, and deleted.
CREDIT_FLOOR_BREACH_EVENT	Generated when a customer's balance has breached a credit floor.
CREDIT_CEILING_BREACH_EVENT	Generated when a customer's balance has breached a credit ceiling.
ADVICE_OF_CHARGE_EVENT	Generated when an Advice of Charge (AoC) event occurs.
THRESHOLD_BREACH_EVENT	Generated when a customer's balance has breached a credit threshold.
AGGREGATED_SPENDING_LIMIT_NOTIFICATION_EVENT	Aggregated notifications are generated when a threshold for a policy-driven charging rule has been reached.
RAR_NOTIFICATION_EVENT	Generated when an ongoing session requires a reauthorization request.
CUSTOM_EVENT_TYPE	Generated when a custom event occurs.
BILLING_NOTIFICATION_EVENT	Generated when a subscriber starts a charging session near the time billing is set to run in BRM. This ensures that billing generates new recurring grants and charges on time.
REPLENISH_POID_ID_EVENT	Generated when ECE needs to obtain POID IDs from BRM.
EXTERNAL_TOP_UP_NOTIFICATION_EVENT	Generated when a customer's balance is topped up through an external application.
LIFECYCLE_TRANSISTION_EVENT	Generated when a service's life cycle transitions to a new state.
FIRST_USAGE_VALIDITY_INIT_EVENT	Generated to synchronize validity of first usage balance elements from ECE to external applications.
RERATE_CREATE_JOB_EVENT	Generated when a rerate job is created.
OFFERING_VALIDITY_INITIALIZATION_EVENT	Generated to synchronize the validity of offer usage balance elements from ECE to external applications.
AUTOMATIC_TOP_UP_TRIGGER_NOTIFICATION_EVENT	Generated when a customer's balance requires an automatic top-up.

Publishing Asynchronous Notifications to a Separate Kafka Topic

By default, ECE publishes asynchronous notifications for external applications to the ECE Notification topic.

You can configure ECE to publish one or more asynchronous notifications to multiple partitions in a separate Kafka topic (called the ECE External Notification topic). ECE routes notifications

using the customer ID as a partition key, meaning each notification has a dedicated partition for a given customer.

Having separate partitions allows you to manage notifications more effectively and replicate the external partitions across sites to increase resiliency.

You configure an ECE on-premises installation to publish asynchronous notifications to a separate Kafka topic using the following entries in your **charging-settings.xml** file. For an ECE cloud native deployment, you set these entries in your **override-values.yaml** file for **oc-cn-ece-helm-chart**:

- **externalTopicEnabled**: Set this to **true**. The default is **false**.
- **externalTopicName**: Set this to the name of the ECE External Notification topic. The default is **EceExtNotifications**.
- **externalPartitions**: Set this to the total number of Kafka partitions in the external topic. The default is **15**.
- **eventListForECEExternalTopic**: Set this to one or more notification types listed in Table 10-3, separated by a comma.

During ECE runtime, you configure these entries as follows:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.kafkaConfigurations.BRM**.
4. Expand **Attributes**.
5. Enable the ECE External Notification topic and specify its configuration values:
   - **externalTopicEnabled**: Set this to **true**. This specifies that the ECE External Notification topic is created and in use.
   - **externalTopicName**: Set this to the name of the ECE External Notification topic.
   - **externalPartitions**: Set this to the total number of Kafka partitions in the ECE External Notification topic.
6. Under the **ECE Configuration** node, expand **charging.notification**.
7. Expand **Attributes**.
8. In the **eventListForECEExternalTopic** attribute, enter the list of notification types to publish to the ECE External Notification topic. Separate multiple notification types with a comma. See Table 10-3 for the list of valid notification types.
9. Perform a rolling upgrade of the ecs server.

Enabling In-Session Group Notifications in ECE

You can configure ECE to send notifications to everyone in a sharing group when one of its members triggers a notification. For example, when a member breaches a credit limit or credit threshold, a notification is sent to all members in the sharing group.

When group notifications are disabled, ECE sends notifications only to the member that triggered the notification.

To enable group notifications:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.server**.
4. Expand **Attributes**.
5. Set the **groupNotificationEnabled** attribute to **True**.
6. (For **firstUsageValidityInitNotificationMode** notification types only) Set **replicateFirstUsageValidityInitEvent** to **true**.
7. Configure the **NotificationEnabledAgreements** subscriber preference:
   a. Add a string for **NotificationEnabledAgreements** in the *BRM_home***/sys/msgs/active_mediation/active_mediation.en_US** file. For example:

```
STR
  ID = string_id;
  Version = 1;
  STRING = "NotificationEnabledAgreements";
END
```

   where *string_id* is a unique numerical ID for the string. See "Creating New Strings and Customizing Existing Strings" in *BRM Developer's Guide* for information about adding strings.

   b. Add **NotificationEnabledAgreements** to the *BRM_home***/sys/data/config/config_subscriber_preferences_map.xml** file. For example:

```
<SUBSCRIBER_PREFERENCES elem="preference_id">
       <NAME>NotificationEnabledAgreements</NAME>
       <SUBSCRIBER_PREFERENCE_ID>preference_id</SUBSCRIBER_PREFERENCE_ID>
       <STRING_ID>string_id</STRING_ID>
       <STR_VERSION>1</STR_VERSION>
       <DEFAULT></DEFAULT>
       <TYPE>1</TYPE>
 </SUBSCRIBER_PREFERENCES>
```

   where:

   - *preference_id* is a unique numerical ID for the preference.
   - *string_id* is the same value you used for **ID** in the **active_mediation.en_US** file.

> **Note:**
>
> Leave the **<DEFAULT>** element blank to ensure that only subscribers who opt in will receive notifications.

When group notifications are enabled, you set subscriber preferences at the profile level to enable notifications for the individual subscribers who want to receive them. See "Configuring Group Notifications" for more information.

Including Rollover Balances in Notifications

Rollover data is included in the balance data synchronized from BRM to ECE using the PIN_FLD_ROLLOVER_DATA field. You can customize notifications sent by ECE to include this

data using the ECE SDK. To do so, extend ECE at the post-charging extension point to call the **getRolloverData()** method.

Enriching Notifications Using ECE Extensions

Using the ECE SDK, you can customize the information included in notifications sent by ECE. Enriching the notification payload can help you provide subscribers with relevant data like their initial granted balance, current balance, session ID, and notification type.

To enrich notifications, extend ECE at the post-charging extension point and use the **SamplePostChargingExtension** program. See "Post-Charging Extension - Enriching Notifications" for more information.

About Configuring BRM Gateway to Process ECE Notifications

ECE publishes notifications intended for external applications to the ECE notification queue. If the notifications are targeted for BRM, BRM Gateway retrieves the notifications from the queue and calls the appropriate BRM opcode. Table 10-4 lists the BRM opcode called for each ECE notification type.

Table 10-4 ECE Notification to BRM Opcode Mapping

ECE Notification Type	BRM Opcode Called	Description
AUTOMATIC_TOP_UP_TRIGGER_NOTIFICATION_EVENT	PCM_OP_PYMT_TOPUP	Generated for top ups made directly via ECE.
BILLING_NOTIFICATION_EVENT	PCM_OP_BILL_MAKE_BILL	Generated when an event occurs after a customer's accounting cycle has terminated. Billing is triggered. Relevant charges and changes in state are synchronized to ECE through the EM Gateway when the opcode completes.
CUSTOM_BRM_OP_CODE_EVENT_NOTIFICATION_EVENT	The opcode called is based on the opcode XML tag.	Permits custom opcodes to be called through the BRM Gateway.
EXTERNAL_TOP_UP_NOTIFICATION_EVENT	PCM_OP_BILL_DEBIT	Synchronizes external top ups, received directly in ECE, for persistence within BRM.
FIRST_USAGE_VALIDITY_INITIALIZATION_EVENT	PCM_OP_BAL_CHANGE_VALIDITY	Initializes BRM-side balance validities based on first-usage alignment configuration.
LIFECYCLE_TRANSITION_NOTIFICATION_EVENT	PCM_OP_CUST_UPDATE_SERVICES	Life cycle transitions are invoked (for things like pre-active to active SIMs), which trigger service status changes on BRM. These changes may indirectly trigger other configured activity.
LOAN_GRANT_EVENT	PCM_OP_LOAN_NOTIFY_THRESHOLD	For triggering loan assessment in BRM (usually a low-balance automatic loan that may permit a session to continue rather than exhaust a customer's credit).
OFFER_VALIDITY_INITIALIZATION_EVENT	PCM_OP_SUBSCRIPTION_SET_VALIDITY	Initializes BRM-side subscription validity based on first-usage alignment configuration.
REPLENISH_POID_ID_NOTIFICATION_EVENT	PCM_OP_GET_POID_IDS	POIDs (used for **/item** assignment in ECE) are preemptively replenished via this opcode call when about 20% of the initial POID cache is remaining.
RERATE_CREATE_JOB_EVENT	PCM_OP_RERATE_INSERT_RERATE_REQUEST	Triggers rerating requests in BRM.

Table 10-4 (Cont.) ECE Notification to BRM Opcode Mapping

ECE Notification Type	BRM Opcode Called	Description
SUBSCRIPTION_CYCLE_FORWARD_EVENT	PCM_OP_SUBSCRIPTION_CYCLE_FORWARD	Permits triggering of cycle-forward events in BRM whenever a balance (that is considered recurring) is expiring as part of an ongoing ECE session.

Configuring BRM Gateway to Process ECE Notifications

To configure BRM Gateway to process ECE notifications, you must connect BRM Gateway to both BRM and the ECE notification queue:

1. When you install ECE, do this:
   - Add details for connecting the BRM Gateway to the BRM Connection Manager (CM).
   - If you are using Oracle WebLogic for notification handling, specify to create WebLogic queues and enter the details for your ECE Notification queue and Suspense queue.
   - If you are using Apache Kafka for notification handling, specify to create Kafka topics and enter the details for your ECE Notification topic and Suspense topic.

   > **Note:**
   >
   > Systems that support 5G networks must use Apache Kafka for notification handling.

   For more information, see "Installing an ECE Integrated System" in *Elastic Charging Engine Installation Guide*.
2. During the ECE postinstallation process, do this:
   - If you are using Oracle WebLogic for notification handling, run the **post_Install.pl** script to create your ECE Notification queue, Suspense queue, and Acknowledgment queue. See "Creating WebLogic JMS Queues for BRM" in *Elastic Charging Engine Installation Guide*.
   - If you are using Apache Kafka for notification handling, run the **kafka_post_install.sh** script to create your ECE Notification topic and Suspense topic. Then, run the **post_Install.pl** script and choose to create only the Acknowledgment queue. See "Creating Kafka Topics for ECE" and "Creating WebLogic JMS Queues for BRM" in *Elastic Charging Engine Installation Guide*.
3. Configure your BRM Gateway instances:
   - If you want to configure a single BRM Gateway instance, see "Configuring the BRM Gateway".
   - If you want to configure multiple BRM Gateway instances, see "Configuring Multiple BRM Gateway for Multi-Schema Deployments".
4. Configure the BRM Gateway queues or topics:
   - If you are using WebLogic queues, see "Configuring WebLogic Queues for BRM Gateway".

- If you are using Kafka topics, see "Connecting BRM Gateway to Kafka Topics and BRM".
- If you are using queues from an external application, see "Considerations for Using a Non-WebLogic Server JMS Provider".

Configuring the BRM Gateway

You must configure a BRM Gateway instance for each schema in your BRM database. To configure a single BRM Gateway instance:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.brmGatewayConfiguration**.
4. Expand **Attributes**.
5. Use the following attributes to configure BRM Gateway:
   - **name**: The name of this BRM Gateway instance.
   - **schemaNumber**: The number of the database schema BRM Gateway connects to in the BRM database.
   - **emptyQueueThreadSleepInterval**: Specifies the interval, in milliseconds, in which the gateway thread pings the ECE Notification queue to check if a connection is available when the queue is empty.
   - **jmsReceiveSleepInterval**: Specifies the interval, in milliseconds, where the BRM Gateway will wait and continue if there are no messages from the Kafka or WebLogic queue.
   - **brmResponseTimeOutInterval**: Specifies the interval, in milliseconds, where BRM opcode service will wait for all responses from CM.
   - **gatewaySleepInterval**: Specifies the number of milliseconds between connection retry attempts for the Notification queue.
   - **jmsBatchSize**: Specifies the maximum number of failed update requests BRM Gateway can retrieve from the Suspense queue in a batch.
   - **jmsReceiveTimeout**: Specifies the timeout in milliseconds for Kafka/JMS to receive messages from the queue.
   - **brmWorkerThreads**: Specifies the number of gateway threads, which, as part of the charging settings configuration, is 10 but should not be less than 1.
   - **brmSchedulerThreadInitialDelay**: Specifies the initial delay in milliseconds before the BRM opcode service task is executed.
   - **brmSchedulerThreadDelayPeriod**: Specify the interval, in milliseconds, at which the BRM opcode service starts executing continuously within the given period.
   - **brmSuspenseQueuePeriod**: Specifies the maximum amount of time, in milliseconds, BRM Gateway has to retrieve a batch from the Suspense queue.
   - **connectionRetryCount**: Specifies the number of times BRM Gateway can retry a connection to the BRM CM after it fails.
   - **connectionRetryInterval**: The amount of time, in milliseconds, between reconnection attempts to the CM.

Configuring Multiple BRM Gateway for Multi-Schema Deployments

You must configure a BRM Gateway instance for each schema in your BRM database, i.e., you cannot have a single BRM Gateway in multi-schema. For example, in a system with three schemas, you must have BRM Gateway 1 connected to Schema 1, BRM Gateway 2 connected to Schema 2**,** and BRM Gateway 3 connected to Schema 3.

In on-premises systems, you can optionally define more than one BRM Gateway instance for each schema. In this case, ECE automatically makes one of the BRM Gateways active while the other instances are inactive. One of the inactive instances comes online only if the active instance goes down. For example, assume BRM Gateway 1 and 2 are connected to Schema 1. When BRM Gateway 1 is active, it remains active and continues processing ECE notifications until the instance goes down. Then, BRM Gateway 2 becomes active and starts processing notifications.

To configure multiple BRM Gateway instances:

1. Open the *ECE_home***/config/management/charging-settings.xml** file.
2. Delete the existing **charging.brmGatewayConfiguration** section:

```
<brmGatewayConfiguration
    config-
class="oracle.communication.brm.charging.appconfiguration.beans.connection.BRMGateway
Configuration"
    name="brmGatewayConfiguration"
    emptyQueueThreadSleepInterval="50"
    jmsReceiveSleepInterval="100"
    brmResponseTimeOutInterval="600000"
    gatewaySleepInterval="2000"
    jmsBatchSize="10"
    jmsReceiveTimeout="2000"
    brmWorkerThreads="10"
    brmSchedulerThreadInitialDelay="10"
    brmSchedulerThreadDelayPeriod="3"
    brmSuspenseQueuePeriod="1800000"
    connectionRetryCount="10"
    connectionRetryInterval="10000">
</brmGatewayConfiguration>
```

3. Copy the following **charging.brmGatewayConfigurations** section into the file:

```
<brmGatewayConfigurations
    config-
class="oracle.communication.brm.charging.appconfiguration.beans.connection.BRMGateway
Configurations">
    <brmGatewayConfigurationList config-class="java.util.ArrayList">
        <brmGatewayConfiguration
            config-
class="oracle.communication.brm.charging.appconfiguration.beans.connection.BRMGateway
Configuration"
            name="brmGatewayN"
            clusterName="@CLUSTER_NAME@"
            emptyQueueThreadSleepInterval="50"
            jmsReceiveSleepInterval="100"
            brmResponseTimeOutInterval="600000"
            gatewaySleepInterval="2000"
            jmsBatchSize="10"
            jmsReceiveTimeout="2000"
            brmWorkerThreads="10"
            brmSchedulerThreadInitialDelay="10"
```

```
            brmSchedulerThreadDelayPeriod="3"
            brmSuspenseQueuePeriod="1800000"
            connectionRetryCount="10"
            connectionRetryInterval="10000"
            schemaNumber="N"/>
</brmGatewayConfigurationList>
```

4. Under **brmGatewayConfigurationList**, add this section for each additional BRM Gateway instance that you want to create:

```
<brmGatewayConfiguration
    config-
class="oracle.communication.brm.charging.appconfiguration.beans.connection.BRMGateway
Configuration"
    name="brmGateway"
    clusterName="@CLUSTER_NAME@"
    emptyQueueThreadSleepInterval="50"
    jmsReceiveSleepInterval="100"
    brmResponseTimeOutInterval="600000"
    gatewaySleepInterval="2000"
    jmsBatchSize="10"
    jmsReceiveTimeout="2000"
    brmWorkerThreads="10"
    brmSchedulerThreadInitialDelay="10"
    brmSchedulerThreadDelayPeriod="3"
    brmSuspenseQueuePeriod="1800000"
    connectionRetryCount="10"
    connectionRetryInterval="10000"
    schemaNumber="1"/>
</brmGatewayConfigurationList>
```

5. Edit these properties for each BRM Gateway instance:
   - **name**: Set this to the name of your BRM Gateway instance in this format: **brmGateway***N*. For example, name the first instance brmGateway1, the second instance brmGateway2, and so on.
   - **schemaNumber**: Set this to the schema number that the BRM Gateway instance connects to in the BRM database. Enter **1** to connect to the first schema, **2** to enter the second schema, and so on.
6. Save and close the **charging-settings.xml** file.
7. Stop BRM Gateway:

```
stop brmGateway
```

8. Open the *ECE_home*/**config/eceTopology.conf** file.
9. Add a row to the file for each BRM Gateway instance.

   For example, to configure three BRM Gateway instances, add three rows in which the node name value is brmGateway1, brmGateway2, and brmGateway3, and the role value for all three rows is brmGateway:

```
#node-name          |role                |host name (no spaces!) |host ip |JMX port
|start CohMgt |JVM Tuning File
brmGateway1         |brmGateway          |localhost              |        |9994
|false        |defaultTuningProfile
brmGateway2         |brmGateway          |localhost              |        |9994
|false        |defaultTuningProfile
brmGateway3         |brmGateway          |localhost              |        |9994
|false        |defaultTuningProfile
```

10. Save and close the **eceTopology.conf** file.

11. From the *ECE_home*/**bin** directory, start Elastic Charging Controller (ECC):

    ```
    ./ecc
    ```

12. Start all BRM Gateway instances:

    ```
    start brmGateway
    ```

Connecting BRM Gateway to Kafka Topics and BRM

To connect BRM Gateway to the ECE Notification Kafka topic and BRM:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.server**.
4. Expand **Attributes**.
5. Set the **kafkaEnabledForNotifications** property to **true**.
6. Expand **charging.kafkaConfigurations**.
7. Expand **Attributes**.
8. Specify the Kafka configuration values for the attributes in Table 10-5.

   Verify that the name, hostname, and topic names that you provide match the settings entered when installing ECE.

Table 10-5 Settings for Connecting BRM Gateway to Kafka Topics

Property Name	Description
name	The name of your ECE cluster.
hostname	The host name and port number of the machine in which Apache Kafka is up and running. If it contains multiple Kafka brokers, create a comma-separated list.
topicName	The name of the Kafka topic in which BRM Gateway retrieves notifications from ECE.
partitions	The total number of Kafka partitions in your topics. The recommended number to create is calculated as follows: [(Max Diameter Gateways * Max Peers Per Gateway) + (1 for BRM Gateway) + Internal Notifications]
kafkaBRMReconnectionInterval	The amount of time, in milliseconds, BRM Gateway waits before attempting to reconnect to the Kafka topic.
kafkaBRMReconnectionMax	The maximum amount of time, in milliseconds, BRM Gateway waits before attempting to reconnect to a broker that has repeatedly failed to connect. The **kafkaBRMReconnectionInterval** will increase exponentially for each consecutive connection failure up to this maximum.

9. Expand **charging.connectionConfigurations.brmConnection**.

10. Expand **Attributes**.
11. Specify the configuration values for connecting BRM Gateway to the BRM Connection Manager (CM):
    - **hostName**: Enter the host name of the server in which the CM is running. For example: abc01.example.com.
    - **loginName**: Enter the user name for logging in to BRM. The default is **root.0.0.0.1**.
    - **cmPort**: Enter the port number for the BRM CM.

About BRM Gateway Error Handling

When BRM Gateway calls an opcode, it checks whether the opcode ran successfully. If it did not, the exception from the opcode is checked and, if the exception is one of the following, it retries the opcode call.

- ERR_TIMEOUT
- ERR_DEADLOCK
- ERR_STREAM_EOF
- ERR_IM_CONNECT_FAILED
- ERR_EM_CONNECT_FAILED
- ERR_NAP_CONNECT_FAILED

If the opcode continues to fail after all of the configured retry attempts, the notification is moved to the Suspense topic.

If the exception is due to a duplicate message (ERR_DUPLICATE), the opcode call is not retried and the message is not moved to the Suspense topic.

Configuring WebLogic Queues for BRM Gateway

To configure WebLogic queues for BRM Gateway:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.connectionConfigurations.brmConnection**.
4. Expand **Attributes**.
5. Use the following attributes to specify connection values to the BRM Connection Manager (CM):
    - **loginName**: Enter the user name for logging in to BRM. The default is **root.0.0.0.1**.
    - **hostName**: Enter the IP address or the host name of the computer on which BRM is configured.
    - **cmPort**: Enter the port number for the BRM CM.

WebLogic Server Configuration Settings for the connectionFactory

For the connectionFactory created within the WebLogic server for creating connections to the JMS topic, do the following:

1. Log on to the WebLogic Server on which the JMS topic resides.

2. In the WebLogic Server Administration Console, from the JMS modules list, select the ConnectionFactory that applies to the JMS topic used for ECE notifications.
3. Do the following:
   a. In the Client tab, set the **Reconnect Policy** to **All**.
   b. In the Transactions tab, set the **Transaction Timeout** to **2147483647**.

Refer to the Oracle WebLogic Server documentation for information about setting up a JMS topic on WebLogic Server.

Considerations for Using a Non-WebLogic Server JMS Provider

If you choose to use a JMS provider other than WebLogic Server for publishing ECE notification events, you must do the following on the driver machine:

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

Modifying JMS Credentials for Publishing External Notifications

When you install ECE, you can specify to publish notifications to a WebLogic JMS queue (named ECE Notification queue) as well as how to connect to the queue.

If you need to change the connection information after ECE is installed, you can edit the parameters in the **JMSConfiguration.xml** file.

To modify the JMS credentials in ECE, do the following:

1. Open the *ECE_home***/config/JMSConfiguration.xml** file.
2. Locate the **<MessagesConfigurations>** section.
3. Specify the values for the parameters in the **NotificationQueue** section:

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

   a. For the **HostName** parameter, specify the host name of the WebLogic server on which the JMS topic resides.
   b. For the **Port** parameter, specify the port number on which the WebLogic server resides.
   c. For the **UserName** parameter, specify the user for logging in to the WebLogic server.

      This user must have write privileges on the JMS topic created.
   d. For the **Password** parameter, specify the password for logging in 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 first run a utility to encrypt the new password before entering it here.

e. For the **ConnectionFactory** parameter, specify the connectionFactory created within the WebLogic server for creating connections to it.

   You must also configure settings in the WebLogic server for the connectionFactory. See "WebLogic Server Configuration Settings for the connectionFactory" for information.

f. For the **QueueName** parameter, specify the JMS topic.

   This is the JMS topic which holds the published external notification messages.

g. For the **Protocol** parameter, specify the wire protocol used. For the WebLogic server, set this to **t3://**.

4. Save and close the file.

11

Configuring Subscriber Preferences

You can configure subscriber preferences such as how they want to receive notifications from the network in Oracle Communications Elastic Charging Engine (ECE).

Topics in this document:

About Subscriber Preferences

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 via SMS text messages.

By default, BRM enables you to manage the following subscriber preferences:

- Preferred channel of communication: IVR, SMS, e-mail, and so on
- Preferred language of communication: English, French, and so on
- Number of days before which the customer wishes to receive the notification
- Interval between two successive notifications
- Timestamp of the last notification sent to the subscriber

BRM stores information about each subscriber's preferences in a subscriber profile repository. BRM stores the types of preferences that you track and their default values in the **/config/subscriber_preferences_map** object. BRM stores each subscriber's preferences at the account level and the service level in individual **/profile/subscriber_preferences** objects.

For more information on the **/config/subscriber_preferences_map** and **/profile/subscriber_preferences** objects, see *BRM Storable Class Reference*.

Setting Up Notifications to Include Subscriber Preferences

To set up ECE to include subscriber preference information in notifications:

1. Enable in-session notifications. See "Enabling Specific Notification Types".
2. Specify the subscriber preferences that can be collected along with their default values. See "Configuring the Subscriber Preferences to Collect".
3. (Optional) Configure ECE to enrich external notifications with subscriber preference information. See "Configuring ECE to Enrich External Notifications with Subscriber Preference Information".
4. (Optional) Configure ECE to send credit limit and threshold breach notifications to multiple members of a sharing group. See "Configuring Group Notifications".

Afterward, you can start collecting your subscribers' preferences through Billing Care, Collections Center, or your custom client application. See "Collecting Your Subscribers' Preferences Using Client Tools".

Configuring ECE to Enrich 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 set for customers in BRM are also stored in ECE.

You can configure ECE to enrich the following types of ECE external notifications with all or a subset of subscriber preferences:

- 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

If the same subscriber preference is defined as a customer preference and as a service preference, ECE uses the service preference. If a subscriber preference is not specified for the service but is specified for the customer, ECE uses the customer subscriber preference.

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

1. If you do not have it, obtain the list of subscriber preference names you have 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 that you previously set in your BRM system.
2. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
3. Expand the **ECE Configuration** node.
4. Expand **charging.notification**.
5. Expand **Attributes**.
6. Set the **subscriberPreferenceUpdateNotificationMode** attribute to **ASYNCHRONOUS**.
7. Select a notification type for which notification messages are to be enriched with subscriber preference information.
8. Specify values for the following attributes:
   - **enrichName**: Enter **subscriberPreferences**.
   - **enrichValue**: 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.

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

The following is an example of the **SubscriberPreferences** block for a threshold breach notification 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>
```

Configuring the Subscriber Preferences to Collect

You can configure which subscriber preferences are displayed and collected in your client applications. To do so, you configure the **config_subscriber_preferences_map.xml** file and load its contents into the database using the **load_config** utility. The default **config_subscriber_preferences_map.xml** file includes sample subscriber preferences, but you can add, modify, or remove them.

To configure the subscriber preferences to collect through your client applications:

1. (Optional) Customize the subscriber preferences by doing the following:
   a. Open the *BRM_home*/**sys/data/config/config_subscriber_preferences_map.xml** file.
   b. Modify or delete the existing subscriber preferences.
   c. For each custom subscriber preference you want to include, add a **<SUBSCRIBER_PREFERENCES>** array element and define the fields in Table 11-1.

Table 11-1 Elements That Store Subscriber Preferences

Element	Description
<NAME>	The name of the preference.
<SUBSCRIBER_PREFERENCE_ID>	The ID associated with the preference.
<STRING_ID>	Used for localization. Billing Care and Customer Center use this information to display the localized preference name.
<STR_VERSION>	The string field version.
<DEFAULT>	The default value for the preference field.
<TYPE>	The type of value that the preference can be assigned from one of the following types: • **1**: STR (alphanumeric) • **2**: INT (integer) • **3**: ENUM (indicating that the preference is one of an ordered list of possible values. An array of values must be provided for this selection.) See the **<VALUES>** element in this table. • **4**: DECIMAL • **5**: TSTAMP (timestamp) • **6**: TIME (time in 24-hour format *HH:mm*) • **7**: MULTI SELECT ENUM (a fixed list that is displayed as a string with check boxes or combo boxes. The list is stored as a string in CSV format.) • **262**: TIME RANGE (time range in 24-hour format: *HH:mm-HH:mm*. For example: 16:00-20:00) For example, to provide a set of possible values, you set **<TYPE>** to **3** and enter an array of values for this preference in **<VALUES>**.
<VALUES>	An array list of possible values for the field, which is used only for **ENUM** types.

For example, the following entries define a new subscriber preference named **Include Current Date** with the possible values of **True** or **False** (default).

```
<ConfigObject>
   <DESCR>Subscriber Prefs Map</DESCR>
   <NAME>Subscriber Prefs Map</NAME>
   <SUBSCRIBER_PREFERENCES elem="x">
      <NAME>Include Current Date</NAME>
      <SUBSCRIBER_PREFERENCE_ID>10</SUBSCRIBER_PREFERENCE_ID>
      <STRING_ID>10</STRING_ID>
      <STR_VERSION>1</STR_VERSION>
      <DEFAULT>909</DEFAULT>
      <TYPE>3</TYPE>
         <VALUES elem="0">
            <VALUE>True</VALUE>
            <STRING_ID>908</STRING_ID>
            <STR_VERSION>998</STR_VERSION>
         </VALUES>
         <VALUES elem="1">
            <VALUE>False</VALUE>
            <STRING_ID>909</STRING_ID>
            <STR_VERSION>999</STR_VERSION>
         </VALUES>
   </SUBSCRIBER_PREFERENCES>
</ConfigObject>
```

d. Save the **config_subscriber_preferences_map.xml** file.

2. Open the *BRM_home*/**apps/load_config/pin.conf** file in a text editor.
3. Add the following as the last entry:

```
- load_config validation_module libLoadValidTCFAAA LoadValidTelcoAAA_init
```

4. Save the **pin.conf** file.
5. Load the **config_subscriber_preferences_map.xml** file into the database by running the **load_config** utility:

```
load_config config_subscriber_preferences_map.xml
```

> **Note:**
>
> - The **load_config** utility requires a configuration (**pin.conf**) file.
> - If you do not run the utility from the directory in which the configuration file is located, include the complete path to the file.
>
> For more information, see "load_config" in *BRM Developer's Guide*.

6. Stop and restart the Connection Manager (CM).

To verify that the updated preference configurations were loaded, you can display the **/config/subscriber_preferences_map** object by using the Object Browser, or use the **robj** command with the **testnap** utility.

For more information on the **/config/subscriber_preferences_map** object, see *BRM Storable Class Reference*.

Configuring Group Notifications

You can configure ECE to send credit limit and threshold breach notifications to multiple members of a sharing group. By default, only the member who triggered the breach is notified.

To configure group notifications:

1. Confirm that group notifications are enabled as described in "Enabling In-Session Group Notifications in ECE".
2. For group owners, set the **ResourcesForSendingNotification** subscriber preference at the account level to a comma-separated list of resource balance element IDs that the owner wants to send notifications for.
3. For group members:
   a. Set **ResourcesForReceivingNotification** at the account level to a comma-separated list of resource balance element IDs that the member wants to receive notifications for. Values included in this list must also be listed in **ResourcesForSendingNotification** for the group owner.
   b. Set **NotificationEnabledAgreements** at the account or service level as a comma-separated list of sharing group names that the subscriber wants notifications for. For example:

```
SharingAgreement12, Charge_Sharing14, DataSharing_143
```

These sharing groups must contain the resources specified in **ResourcesForReceivingNotification**.

c. Set **OfflineNotificationEnabled** at the account or service level to **true** if the group member wants to receive notifications when they are offline for breaches caused by other group members. The setting at the service level overrides the setting at the account level.

> **Note:**
>
> Because group owners can also be group members, an individual subscriber might have all of these preferences set in their subscriber profile.

Collecting Your Subscribers' Preferences Using Client Tools

You can add subscriber preferences to your customers' accounts by using one of the following:

- Billing Care. See "Viewing and Adding Subscriber Preferences" in *Billing Care Online Help*.
- A custom client application calling the Profiles REST endpoints in the Billing Care REST API. See REST API References for Billing Care.
- A custom client application calling the subscriber profile opcodes. See "Managing and Customizing Profiles" in *BRM Opcode Guide*.
- Customer Center. See the Customer Center Help.

These client tools use the configurations in the ***/config/subscriber_preferences_map*** object to list the preferences that a subscriber can configure dynamically. The client tool does not display the subscriber preferences pages and dialog boxes if this object is missing.

By default, CSRs cannot access subscriber preferences in Billing Care and Customer Center. You must grant them permission to view and update subscriber preferences by doing the following:

- Billing Care: Granting CSRs the **ProfileResource** authorization resource. See "About Billing Care Authorization Resources" in *BRM Security Guide*.
- Customer Center: Assigning CSRs the ***/customercenter/subscriber_preference*** role in Customer Center or Permissioning Center. See "Setting Up Permissions in BRM Applications" in *BRM System Administrator's Guide*.

Part IV
Managing Charging Sessions

This part provides information about managing online and offline charging sessions in Oracle Communications Elastic Charging Engine (ECE). It contains the following chapters:

12

About Linear and Non-Linear Rating

Learn about linear and non-linear rating in Oracle Communications Elastic Charging Engine (ECE).

Topics in this document:

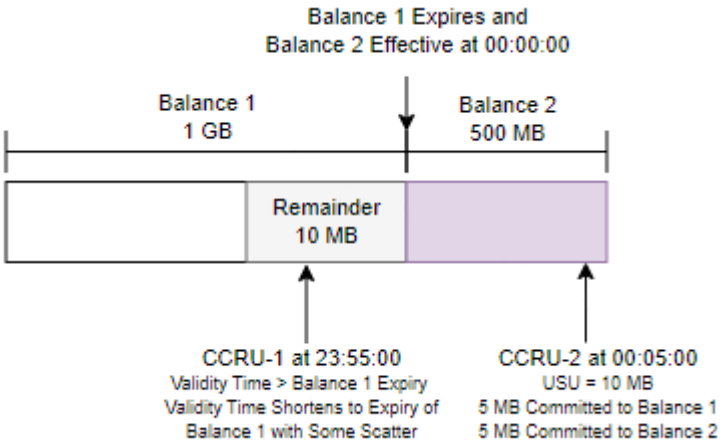
About Linear Rating

Linear rating, a concept widely used in the telecom industry, assumes that all resource consumption occurs at a constant rate. For example, a rate of 1 MB per minute for data usage or 1 free minute per minute of voice usage. If a customer consumes 100 MB in 10 minutes, linear rating assumes a usage rate of 10 MB per minute.

However, this approach can have consequences when boundary conditions arise if consumption is not linearly consumed over time. This is because linear rating evenly distributes the resource consumption across the usage period. For this reason, linear rating is best suited for voice and event usage, which have constant consumption rates.

Figure 12-1 shows a linear rating example in which usage is granted at 23:55:00 with CCRU-1 and is reported as 10 MB in the subsequent CCRU-2 at 00:05:00.

Figure 12-1 Linear Rating Example



In this example, the total usage is 10 MB in 10 minutes, so linear rating calculates a consumption rate of 1 MB for each minute. Since Balance 1 expires at 00:00:00, linear rating distributes the usage as follows:

- 5 MB consumed between 23:55:00 and 23:59:59, which is charged to Balance 1
- 5 MB consumed between 00:00:00 and 00:05:00, which is charged to Balance 2

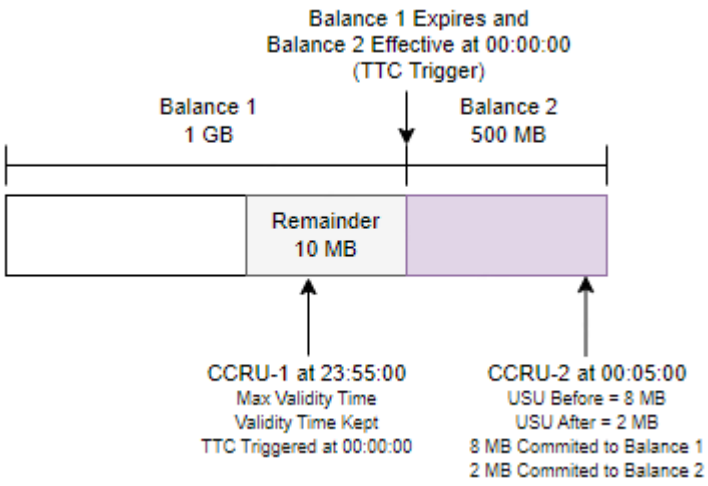
Balance 1 reduces from 10 MB to 5 MB, and Balance 2 reduces from 500 MB to 495 MB.

About Non-Linear Rating

Non-linear rating makes no assumption about the resource consumption velocity. For instance, it can accurately monitor a customer's data consumption of 10 MB in the first five minutes and an additional 20 MB in the subsequent five minutes. Non-linear rating calculates resource consumption based on the granted usage amount and uses Tariff Time Change (TTC) to manage boundary change conditions. This method ensures precise usage reporting both before and after boundary conditions change.

Figure 12-2 shows a non-linear rating example in which CCRU-1 is granted 10 MB from 23:55:00 until 23:59:59 because a TTC occurs at midnight. CCRU-2 is granted a separate usage amount from 00:00:00 until the next TTC or credit limit.

Figure 12-2 Non-Linear Rating Example



In this example, CCRU-2 reports that 8 MB were consumed before the TTC and 2 MB after the TTC. Non-linear rating distributes the usage as follows:

- 8 MB consumed between 23:55:00 and 23:59:59, which is charged to Balance 1
- 2 MB consumed between 00:00:00 and 00:05:00, which is charged to Balance 2

Balance 1 reduces from 10 MB to 2 MB, and Balance 2 reduces from 500 MB to 498 MB.

ECE applies non-linear rating to all usage by default. Non-linear rating enables ECE to rate long network sessions incrementally based on the exact data consumed between tariff changes. It also lets operators show subscribers the running balance based on the data consumption after each tariff change.

When using non-linear rating, ECE determines if there is a tariff change during the authorization and reservation process for a session request from the network. ECE creates a reservation for resources before and after TTC and sends a Tariff-Time-Change AVP in the response to the network to record the exact data consumed before and after the tariff change. Additionally, ECE conducts reverse rating to calculate the amount of usage the subscriber can afford and reserves the balance for the RSUs based on the worst-case charging condition (the

maximum charge that can be applied for the RSUs). This process ensures that the overall usage does not exceed the customer's credit limit and that there is no revenue leakage, regardless of whether the balance is consumed before or after the tariff change.

Configuring ECE to Use Linear Rating for Specific Products and Events

By default, ECE uses non-linear rating for all usage events, which is ideal for data usage. However, you might want to use linear rating for some events and products, such as those for voice usage.

To configure ECE to use linear rating for specific products or event-and-product combinations:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.reservationConfig**.
4. Expand **Operations**.
5. For each product that you want to use linear rating, select **enabledOrDisableNonLinear** and then set these attributes:
   - **productType**: Specify the name of the product defined in the ECE request specification data, such as TelcoVoice.
   - **enableOrDisable**: Set this to **false**.
6. For each product and event combination to use linear rating, select **enabledOrDisableNonLinearBasedOnProductTypeAndEventType** and set these attributes:
   - **enableOrDisableNonLinear**: Set this to **false**.
   - **productType**: Enter the name of the product defined in the ECE request specification data. For example: TelcoVoice.
   - **eventType**: Enter the name of one or more events separated by a comma. For example: EventVoice.
7. Expand the **ECE Configuration** node.
8. Expand **charging.server**.
9. Expand **Attributes**.
10. Select the **tariffTimeChangeSupported** attribute and set the value to **false**.

13

Managing Midsession-Rated Events

You can configure Oracle Communications Elastic Charging Engine (ECE) to generate a rated event during the middle of a network session based on the trigger criteria that you specify.

Topics in this document:

About Midsession-Rated Events

By default, ECE generates a rated event for a network session when a session ends. You can also configure ECE to generate rated events when an update operation occurs during the session. Such events are called *midsession-rated events*.

To generate midsession-rated events, you enable the feature and then define conditions, called *triggers*, that initiate the generation of these 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)
- Custom criteria, such as network condition changes or spans from one offer to another due to balance exhaustion

> **Note:**
>
> To trigger a midsession-rated event based on custom criteria, you must extend ECE at the pre-rating or post-rating extension points. See "Pre-Rating Extension – Generating Midsession-Rated Event" and "Post-Rating Extension – Generating Midsession-Rated Events" for more information.

Each trigger is associated with a service-event pair. If an ongoing session meets the trigger conditions when an update operation occurs, a midsession-rated event for the specified service is generated.

> **Note:**
>
> ECE checks for trigger conditions only during update operations. For example, 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.

Configuring ECE to Generate Midsession-Rated Events

To configure ECE to generate midsession-rated events:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.midSessionCdrConfiguration**.
4. Expand **Attributes**.
5. Set the **MidSessionCdrEnabled** attribute to **true**.
6. Define trigger conditions for one or more service-event pairs:
   a. Expand **Operations**.
   b. Click **addOrUpdateMidSessionCdrTriggerDetails**.
   c. Specify values for the fields listed in Table 13-1.

Table 13-1 Fields for Defining Midsession-Rated Event Triggers

Field	Description
productType	The name of the service for which you are creating the trigger (for example, **"DATA"**).
eventType	The name of the event for which you are creating the trigger (for example, **"DATA_USAGE"**).
triggerName	The name of the trigger you are defining.
qtyFields	The 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	The unit of measure for conditions based on quantity (for example, **"MEGABYTES"**).
qtyValue	The total quantity of the unit that triggers event generation (for example, **"70"**).
durationUnit	The unit of measure for conditions based on duration (for example, **"HOURS"**).
durationValue	The amount of the unit that triggers event generation (for example, **"70"**).
timeOfDay	The 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.
midSessionForNonCounterGrants	Whether to generate an event based on the consumed unit.

Table 13-1 (Cont.) Fields for Defining Midsession-Rated Event Triggers

Field	Description
offerChangeTrigger	Whether to create midsession-rated events for charge offer changes. Can have values of **true** or **false**. **Note:** If this value is set to **true**, you must also enable the **postRatingMidSessionExtension** extension. For more information, see "Generating Midsession-Rated Events for Charge Offer Changes".

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.

> **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).

Generating Midsession-Rated Events for Charge Offer Changes

You can configure ECE to generate midsession-rated events whenever it starts rating a customer's usage with a new charge offer. For example, assume a session that consists of the following:

- Talk time reported: 30 minutes
- First 12 minutes are rated using Charge Offer 1
- Next 15 minutes are rated using Charge Offer 2
- Last 3 minutes are rated using Charge Offer 3

For this session, ECE generates two midsession-rated events:

- CDR 1 for the first 12 minutes
- CDR 2 for the next 15 minutes

The remaining 3 minutes stay in the active session.

If the session is split into multiple CDRs, the start and end times of each CDR can be estimated.

To configure ECE to generate midsession-rated events for charge offer changes:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.midSessionCdrConfiguration**.
4. Expand **Attributes**.
5. Set the **MidSessionCdrEnabled** attribute to **true**.
6. To define a new trigger, follow the steps in "Configuring ECE to Generate Midsession-Rated Events".
7. To update an existing trigger:
   a. Expand **Operations**.
   b. Click **addOrUpdateMidSessionCdrTriggerDetails**.
   c. In the **offerChangeTrigger** field, enter **true**.
8. Under the **ECE Configuration** node, expand **charging.extensions**.
9. Expand **Attributes**.
10. Set the **postRatingMidSessionExtension** attribute to **oracle.communication.brm.charging.sdk.extensions.Sample postRatingMidSessionExtension** or to the name of your custom extension.

> **Note:**
>
> ECE does not generate midsession-rated events when:
>
> - Only one charge offer is applied to the rating result.
> - The **postRatingMidSessionExtension** extension hook class is not defined.
> - The charge offer has a rate plan with multiple tiers.

Table 13-2 describes how ECE behaves based on the settings of the **postRatingMidSessionExtension** attribute, the **offerChangeTrigger** attribute, and whether the rate plan has multiple tiers.

Table 13-2 Attribute and Rate Plan Settings

postRatingMidSession Extension is defined	offerChangeTrigger is enabled	Rate plan is multi-tier	Rated Event
True	True	False	A midsession-rated event is created for each charge offer, except for the last one.
True	True	True	No midsession-rated events are created for charge offer changes.
True	False	True or False	No midsession-rated events are created for charge offer changes.

Table 13-2 (Cont.) Attribute and Rate Plan Settings

postRatingMidSession Extension is defined	offerChangeTrigger is enabled	Rate plan is multi-tier	Rated Event
False	True	True or False	No midsession-rated events are created for charge offer changes.
False	False	True or False	No midsession-rated events are created for charge offer changes.

Generating Midsession-Rated Events When the USU is Block Missing

When the network sends a Final Unit Indicator (FUI) followed by a top up, ECE generates a reauthorization request (RAR). The network then sends an update request, sometimes not including the Used Service Units (USU) block for the ongoing session. When this occurs, ECE generates a midsession-rated event. This ensures that any part of the reservation consumed by the account is reported, preventing revenue loss.

Customizing the Worst-Case Charging Reservation

When resources are requested around tariff-time changes, ECE performs reverse rating to calculate the usage the subscriber can afford and reserves the balance for the requested service units based on the worst-case charging condition.

You can customize how ECE performs the worst-case charging reservation by using the ECE SDK. To do so, extend ECE at the post-rating or post-charging extension point to do the following in **ExtensionRatingPeriod**:

- Invoke the **isAdjustedForWorstCost** method, which determines whether worst-case charging was applied when calculating the reservation amount. The method returns a Boolean value.
- Add customizations to perform if **isAdjustedForWorstCose** is **true**.

For example:

```
Collection<ExtensionRatingPeriod> ratingPeriodCollection =
extensionContext.getRatingPeriods();

for (ExtensionRatingPeriod ratingPeriod : ratingPeriodCollection) {
        if(ratingPeriod.isAdjustedForWorstCost()){
                //your customizations
        }
}
```

Viewing the Reason for a Midsession-Rated Event

When ECE generates a midsession-rated event, it automatically records why the event was split in the event's **midSessionCDRSplitReason** field. ECE sets the field to one of the preconfigured reason codes in Table 13-3.

Table 13-3 Reasons for Creating Midsession-Rated Event

Reason Code	Description	Enum Value
CONFIGURED_VOLUME_REACHED	The event exceeded a configured volume, such as 100 MB.	1
CONFIGURED_DURATION_REACHED	The event exceeded a configured amount of time, such as 2 hours.	2
RATING_CONDITION_CHANGE	The trigger was caused by a change in tariffs.	3
CONFIGURED_TIME_OF_THE_DAY_CROSSED	The event crossed a configured time of day, such as midnight.	4
INTERNAL_TRIGGER	The trigger was caused by a context change.	5
EXTERNAL_TRIGGER	The midsession trigger condition was met.	6
MULTIPLE_USU	The event contained multiple Used Service Units (USUs). ECE generates a midsession-rated event for each USU.	7
MULTIPLE_USU_AND_EXTERNAL_TRIGGER_OR_INTERNAL_TRIGGER	The event was caused by any of the above triggers.	8
PRE_MIDSESSION_CONDITION	The event reached a custom trigger at the prerating extension point.	9
POST_MIDSESSION_CONDITION	The event reached a custom trigger at the post-rating extension point.	10
ZONE_CHANGE	The trigger was caused by a change in zone.	11
BALANCE_EXHAUST	The event was triggered after the complete exhaustion of an existing balance.	12

The following shows a sample snippet of a midsession-rated event with the **reasonForMidSessionCDRSplit** field set:

```
balanceOutputMap = {BALANCE=BalanceOutputImpl{ customerId=39384585, balanceId='BALANCE',
status='SUCCESS', firstUsageValidityMap{},
impacts{840=[OutImpactImpl{ balanceItemId=1, impact=UnitValue{quantity=60.00,
unit=Money{curr=USD}}, validityRule=null,
firstUsageCreatedTime=null, type=0}]}, grantValidityMap={840={1=[Pair{first=null},
{second=null}]}}, recurringImpactMap=null,
firstUsageValidityRuleMap={}, firstUsageCreatedTimeMap={},
currentAndLoanAmounts={840=[Pair{first=60.00}, {second=0}]}}}}'}}
{reasonForMidSessionCDRSplit=[CONFIGURED_DURATION_REACHED]}]
```

You can also extend ECE to add these preconfigured reason codes to midsession-rated events generated by custom criteria. To do so, you extend ECE at the pre-rating and post-rating midsession extension points.

For more information, see these ECE SDK sample programs:

- **SamplePreRatingMidSessionExtension**. See "Pre-Rating Extension – Generating Midsession-Rated Event".
- **SamplePostRatingMidSessionExtension**. See "Post-Rating Extension – Generating Midsession-Rated Events".

14

Managing Online Charging Sessions

You can learn how to configure online charging sessions in Oracle Communications Elastic Charging Engine (ECE).

Topics in this document:

Configuring ECE to Support Prepaid Usage Overage

You can configure ECE to capture any overage amounts by prepaid customers during an active session, which can help you prevent revenue leakage. If the network reports that the number of used units during a session is greater than a customer's available allowance and credit limit, ECE charges the customer up to the available allowance. It then creates an overage record with information about the overage amount and sends it to the ECE Overage topic. You can create a custom solution for reprocessing the overage amount later on.

> **Note:**
>
> Prepaid usage overage is supported on Kafka Server only. It is not supported on WebLogic Server.

For example, assume a customer has a prepaid balance of 100 minutes, but uses 130 minutes during a session. ECE would charge the customer for 100 minutes, create an overage record for the remaining 30 minutes of usage, and write the overage record to the ECE Overage topic.

To configure ECE to support prepaid usage overage, do the following:

1. Ensure that you created an ECE Overage topic and connected ECE to your Kafka Server. See "Connecting ECE to Kafka Topics".
2. Enable ECE to check for and capture any usage overage:
   a. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
   b. Expand the **ECE Configuration** node.
   c. Expand **charging.server**.
   d. Expand **Attributes**.
   e. Set the **checkReservationOverImpact** attribute to **true**. (The default is false.)

You can also customize ECE to include any existing unrated quantity in a CDR or notification using the ECE SDK post charging extension (**postCharagingExtension**). This can be done using the **OverageDetailsImpl** class. The unrated quantity can be then captured by the **getOverageDetailsMap()** method that is available in the extension.

Managing Dynamic Charging Overrides for Online Sessions

Dynamic charging allows you to override an offer's default rate on a per customer basis based on the date. For example, you could create a dynamic override that charges new customers $0.04 per MB for the first six months, $0.05 per MB for the next four months, and then the default rate for all subsequent months.

You define dynamic charging overrides by creating a pricing tag and associating it with an event type, date ranges, and an amount. For example, you could create a pricing tag named **MyPrice** associated with a $0.05 per MB charge for **/event/session/telco/gsm** events from June through September. You could then add the pricing tag to your offers' balance impacts.

You configure dynamic charging overrides using PDC or the **ImportExportPricing** utility. See "Configuring Dynamic Pricing for Usage Events" in *PDC Creating Product Offerings*.

BRM stores information about dynamic charging overrides in the **/offering_override_values** object.

During rating, ECE determines whether a pricing tag in a balance impact matches a pricing tag and date range combination in the **/offering_override_values** object. If it finds a match, ECE charges the amount associated with the pricing tag. If it does not find a match, ECE charges the default amount.

Processing Granted Allowances Before Applying Usage Charges

ECE processes a customer's granted allowances and usage charges in the order in which they appear in the product offerings you define in PDC. For example, assume you configure a product offering in PDC that grants 5 GB of free data per month and then charges $10 per additional GB of data. If a customer purchases the product offering and uses 8 GB of data in a month, ECE would first consume the 5 GB of free data and then apply a $30 charge for the remaining 3 GB of data usage.

To configure charges in PDC to process granted allowances before applying usage charges, see "Configuring Pricing to Consume Granted Allowances Before Charging" in *PDC Creating Product Offerings*.

Enabling Server-Initiated Reauthorization Requests

ECE can perform server-initiated reauthorization requests (RAR) during an ongoing session. This enables 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 from the following conditions:

- 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 balance that expires mid-session, a balance 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 balances are added to a sharing group owner's account.

To enable server-initiated reauthorization requests:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.notification**.
4. Expand **Attributes**.
5. 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.
6. Under the **ECE Configuration** node, expand **charging.server**.

7. Expand **Attributes**.
8. 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.
   - 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. This is the default.

> **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.

Customizing Server-Initiated Reauthorization for Sharing Groups

By default, when ECE performs a server-initiated RAR during a customer's ongoing session, it generates RAR notifications for all of the customer's ongoing sessions.

You can customize ECE to support server-initiated RARs for sharing group members. In this case, ECE initiates RAR notifications for all ongoing sessions of the customer responsible for triggering the RAR along with all ongoing sessions of the sharing group members. For example, assume a sharing group includes accounts A, B, C, and D, and accounts A and C have ongoing sessions using the sharing group balance. If an RAR is triggered for account C, ECE sends RAR notifications for account A's and C's ongoing sessions.

To do so, extend ECE at the prerating, post-charging, and post-rating extension points and use these ECE SDK sample programs: **SampleRarPostChargingExtension**, **SampleRarPostRatingExtension**, and **SampleRarPreRatingExtension**. See "Rating/Charging Extension – Triggering RAR Notifications" for more information.

Configuring ECE to Return Remaining-Balance Information in Usage Responses

You can configure ECE to return a 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 balances impacted by the session (that is, the balances to which the session applied balance impacts).

For charge distribution scenarios (charge sharing), ECE returns the remaining-balance information for the balances impacted by the sharer's usage.

To configure ECE to return remaining-balance information in usage responses:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.server**.
4. Expand **Attributes**.
5. 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 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 valid sub-balances of (current
     balance + consumed reserved amount)}
     ```

Configuring Taxation in ECE

By default, you configure taxation in Pricing Design Center (PDC) and the information is published to BRM and ECE. ECE then applies taxes during rating using the following:

- Tax codes, which apply simple flat taxes.
- Tax selectors, which apply tax codes based on account, service, event, and profile attributes.
- Tax exemption selectors, which reduce or eliminate the amount of tax your customers pay based on account, service, event, and profile attributes.

See "About Calculating Taxes" in *BRM Calculating Taxes* for more information.

You can also configure simple, fixed-rate tax, such as GST or VAT, for both charges and alterations (discounts) directly in ECE.

To configure taxation in an ECE runtime environment:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.taxation**.

4. Expand **Operations**.
5. Click **addTaxDetails**.
6. Specify values for the following parameters:

> **Note:**
>
> 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.

7. Specify an additional **taxCode**, **taxRate**, and **taxGlId** value for each charge offer or discount offer to which a tax applies.

Managing Direct Debit Data in ECE Cache

By default, ECE stores information about all direct debit events in its cache so it can validate any refund requests against one of the events.

However, some event types may not be eligible for refunds, so storing the direct debit information in the cache is unnecessary. To save space and improve performance, you can configure ECE not to store direct debit information in its cache for specific event types.

To prevent ECE from storing direct debit information in its cache for specific event types:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.server**.
4. Expand **Attributes**.
5. In the **excludedEventsForDebitRefundSessions** attribute, list the event types separated by a comma. For example, enter **EventDelayedSessionTelcoGsm, EventDelayedSessionTelcoGprs**.

Configuring How ECE Manages Active Sessions When Network Elements Fail

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.

To configure how ECE manages active sessions when network elements fail:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.server**.
4. Expand **Attributes**.
5. 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.

Configuring ECE to Redirect Subscriber Sessions to a Service Portal

Service providers can redirect a subscriber session to a service portal, which is 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 website 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 the **http://myPrePaidHomeNetworkRedirect.com** URL address.

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.

When the credit-control client receives the Final-Unit-Indication in the answer from ECE, the credit-control client behavior depends on the value, TERMINATE or REDIRECT, indicated in the Final-Unit-Action AVP. If you do not configure redirection rules, ECE indicates a Final-Unit-Action of TERMINATE in the usage response.

To configure ECE to redirect subscriber sessions to a service portal:

1. Create your redirection rules in a text editor and save the file.

   If you have multiple redirection rules, 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 configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
3. Expand the **ECE Configuration** node.
4. Expand **charging.redirectionConfiguration**.
5. Expand **Attributes**.
6. 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.

> **Note:**
>
> Modifying a redirection-rule configuration in JConsole 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.

For example, the following redirection rule specifies that when a customer is roaming, the redirect address is **http://RedirectRoaming.com** and the redirect address type is **URL**:

```
"( (@fui AND @roamingRequest) => [redirect_type:"URL",redirect_address:
"http://RedirectRoaming.com"];
```

This redirection rule specifies that when a customer is both postpaid and roaming, the redirect address is **http://RedirectRoaming.com**, the redirect address type is **URL**, and the maximum redirection time is 900 seconds:

```
"( (@fui AND ({business_profile([name:"POSTPAID"])} == "true" )) AND
@roamingRequest) => [redirect_type:"URL",redirect_address:
"http://RedirectRoaming.com",redirect_validity:"900"]"
```

Table 14-1 shows redirection-rule conditions that you can use to create redirection rules.

Table 14-1 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")	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. For example: `{business_profile([name:"POSTPAID"])} == "true" )`
@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 definitions. 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")`

Table 14-1 (Cont.) ECE Redirection-Rule Conditions

ECE Redirection-Rule Conditions	Description

@productType	Retrieves the service.
	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 14-1 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 website 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 the **http://myPostPaidRoamingRedirect.com** URL address 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 the **http://myPostPaidHomeNetworkRedirect.com** URL address 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 the **http://myPrePaidRoamingRedirect.com** URL address.
- Given a subscriber with an account using a prepaid payment type who is *not* roaming, redirect the subscriber to the **http://myPrePaidHomeNetworkRedirect.com** URL address.

Enabling Match Factor in ECE

ECE supports the match factor in discounting.

To enable the match factor in ECE:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.server**.
4. Expand **Attributes**.
5. Set the **matchFactorEnabled** attribute to **true**.

Configuring Diameter Gateway to Bypass Rating During ECE Downtime

During a planned maintenance activity or in an unplanned downtime of an ECE node, you can configure Diameter Gateway to continue receiving the CCRs and responding to the service network without rating the CCRs in real-time.

When Diameter Gateway is configured to bypass rating, it persists the Diameter CCRs to the Oracle NoSQL database. Later, when ECE nodes are restored, you can replay the persisted CCRs to the ECE charging servers for rating and updating balance impacts. With this functionality, services can be delivered to your subscribers without any interruption.

To configure Diameter Gateway to bypass rating, perform the following procedures:

1. Enable the **ocsBypassExtension** charging extension.
   a. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
   b. Expand the **ECE Configuration** node.
   c. Expand **charging.extensions**.
   d. Expand **Attributes**.
   e. Specify the fully qualified class name of the extension for the **ocsBypassExtension** attribute.
2. Start the persistence of the diameter messages. See "Managing the Persistence of Usage Requests During ECE Downtime".
3. After the ECE nodes are restored, use Diameter Replayer to replay the CCRs to ECE for rating and updating balance impacts. See "Replaying Persisted Requests into ECE".

Managing the Persistence of Usage Requests During ECE Downtime

When Diameter Gateway is configured to bypass rating during a planned maintenance activity or in an unplanned downtime of ECE, Diameter Gateway receives CCRs and persists them to Oracle NoSQL Database. Later, when the ECE nodes are restarted, you stop the bypass rating extension and replay the persisted messages to the ECE charging servers for rating and updating balance impacts.

Managing persistence of diameter requests involves starting and stopping the persistence of the requests. Before persisting the incoming diameter requests, ensure that the **ocsBypassExtension** charging extension is enabled. See "Configuring Diameter Gateway to Bypass Rating During ECE Downtime" for information on configuring Diameter Gateway to bypass rating.

For planned maintenance activities, you start the persistence of usage requests before the ECE nodes become unavailable. During an outage, persistence of diameter requests starts only if the **ocsBypassExtension** charging extension is enabled. If only the extension is enabled and bypass rating is not started, the usage requests do not flow to the ECE Charging Server nodes. In such a scenario, Diameter Gateway returns Error 5012 for the requests.

> **Note:**
>
> The bypass rating functionality is supported for Gy diameter messages only.

To persist incoming diameter requests:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **DiameterGateway** node.
3. Expand **BFTTask**.
4. Expand **Operations**.
5. Click **startByPass**.

   The Diameter Gateway starts persisting the incoming requests.

In planned maintenance activities or in unplanned downtime, once the ECE nodes become available and start running, you must stop the persistence of the requests. Otherwise,

messages are persisted even after the ECE nodes are up and running, which results in a large volume of requests that need to be replayed.

To stop persisting the requests, click **stopByPass**. Before doing this, you can check if persistence of requests is running by clicking **BFTRunning**. If bypassing rating is enabled, this field shows **True**. Otherwise, it shows **False**.

Replaying Persisted Requests into ECE

When Diameter Gateway is configured to bypass rating during a planned maintenance or unplanned downtime of ECE, Diameter Gateway persists the incoming CCRs to the Oracle NoSQL Database. Later, during a non-peak period, you replay the persisted CCRs to the ECE charging server when the ECE server is restored and is ready to process real-time requests. When you replay the persisted CCRs, the requests are passed to the ECE charging server, which then rates the CCRs and updates the balance impacts. You can plan for when to start replaying the persisted messages, considering replaying the persisted messages can have performance impacts while real-time requests are also processed. Before replaying the persisted requests, ensure that bypass rating extension is disabled and the ECE nodes are up and running. To check if bypass rating is stopped, click **BFTRunning** in ECE MBeans.

To replay the persisted requests into ECE:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **DiameterGateway** node.
3. Expand **BFTTask**.
4. Expand **Operations**.
5. Click **persistedMessageCount** to view the number of requests that have been persisted and are ready for replay.
6. Click **startBFTReplayer**.

   The replayer starts replaying the persisted messages to the ECE charging servers. As messages are replayed to the servers, the number of **persistedMessageCount** keeps decreasing until it becomes **0**.

   The **replayedMessageCount** field shows the number of requests that are being replayed. As requests are replayed, the number of **replayedMessageCount** keeps increasing until it matches the initial count of **persistedMessageCount**.

You stop replaying the messages once all the persisted messages are replayed into the ECE charging servers.

To stop replaying the persisted messages:

1. Check the status of the replayer by clicking **BFTReplayerRunning**. Ensure that this field shows **Running**.
2. Check the status of the **replayedMessageCount** field. Ensure that this field shows **0**, which indicates that all the persisted messages are replayed into the ECE charging servers.
   - **replayedMessageCount** shows the number of Diameter messages that are replayed to the ECE charging server by the current instance of Diameter Replayer.
   - **persistedMessageCount** shows the number of Diameter messages that have been persisted to Oracle NoSQL database, but are yet to be replayed.
3. Click **stopBFTReplayer**.

Replaying of messages is stopped.

Accessing ECE Configuration MBeans

For all configurations, start by accessing the ECE configuration MBeans:

1. Log on to the driver machine.
2. Start the ECE charging servers (if they are not started).
3. Connect to the ECE charging server node enabled for JMX management.
   This is the charging server node set to **start CohMgt = true** in the *ECE_home***/config/** **eceTopology.conf** file, where *ECE_home* is the directory in which ECE is installed.
4. Start a JMX editor that enables you to edit MBean attributes, such as JConsole.
5. In the editor's MBean hierarchy, find the ECE configuration MBeans.

Customizing Consumption Order of Loan and Principal Balances

By default, ECE consumes the customers' loan balance before consuming their principal balance. However, you can configure ECE to consume the main balance first at the service type level as well as the event type level. This customization allows you to prevent fraud where the customer might use the loan balance and leave without settling the loan.

You customize ECE to consume the principal balance first by extending ECE at the **prerating** extension point. Add the following lines to your **prerating** extension Java class:

```
void setUseLoanBalanceItem(Boolean useLoanBalanceItem);
        }
```

For more information, see "Customizing Rating"

Location-Based Charging

Location-based charging allows you to alter the rate based on your customer's location. For example, you could charge $5 per GB for customers located in Silicon Valley.

You configure location-based charging in PDC by:

- Creating a value map that associates a value map name with a set of cell IDs. For example, you could create a value map named **Silicon Valley** associating with cell IDs **209**, **408**, **415**, **650**, and **510**.

  See "Configuring Value Maps" in *PDC Creating Product Offerings* for more information.
- Creating a charge selector or discount selector that maps a value map name to a specific charge or discount.

  For example, you could map the London value map to a 5 pound per GB charge and map the Manchester value map to a 6 pound per GB charge. See “Charge Selectors” in *PDC Online Help*.
- Adding the charge selector to a charge offer, or adding the discount selector to a discount offer.

When ECE rates an incoming usage request, it determines whether a cell ID in the request is included in a value map. If it is included, ECE checks whether:

- A charge selector associates the value map name to a charge amount.
- A discount selector associates the value map name to a discount amount.

If ECE finds a match, it applies the charge or discount associated with the value map name.

15

Managing Session Start and End Times

You can configure how Oracle Communications Elastic Charging Engine (ECE) sets the start and stop times for your customers' sessions.

Topics in this document:

Using Session Connect Time for Charging

ECE uses the session attempt time, which is the time a session is initiated, as the session start time for calculating usage charges. For example, when a customer initiates a call at 10:00:00 and the call actually gets connected at 10:00:30, 10:00:00 is considered as the call start time.

You can configure ECE to instead use the session connect time, which is the time the session actually begins, as the session start time for calculating usage charges. You can do this by setting the **connectionTimeEnabled** entry in the *ECE_home*/**config/management/charging-settings.xml** file to **true**.

To use the session connect time for calculating usage charges:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.reservationConfig**.
4. Expand **Operations**.
5. For each product that you offer, do the following:
    a. Select **enabledOrDisableConnectionTime**.
    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).
        - **enableOrDisable**: Enter **true** to use the session connect time.
    c. Click the **enabledOrDisableConnectionTime** button.

Optimizing Network Signaling

Many times, bundles expire at midnight on a particular day. If your customers are using bundles at that time, the renewal and request messages that are transmitted at midnight can cause an undue network load. To prevent this, you can configure ECE to randomize the validity times for a service so that renewal requests do not occur simultaneously.

To optimize network signaling by randomizing validity times:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.notification**.
4. Expand **Attributes**.
5. Set **subscriptionCycleForwardMode** to **true**.

Configuring ECE to Align Validity Start and End of Conditional Balance Impacts and Charge Offers

When you design your pricing components in Pricing Design Center, you can create charges for which conditional balance impacts are configured.

You can configure a runtime option in ECE that aligns the validity start of a conditional balance impact with the validity start of the associated purchased charge offer and aligns the validity end of a conditional balance impact with the validity end of the associated purchased charge offer. For example, if a customer activates a conditional balance impact valid for three days and the charge offer with which it was purchased is not valid after one day, this configuration specifies whether the conditional balance impact can still be used after the charge offer validity has ended. If ECE does not align the validity end of the conditional balance impact with the validity end of the charge offer the customer purchased, the balance can be used by another charge offer.

To configure ECE to align validity start and end of conditional balance impacts and charge offers:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.server**.
4. Expand **Attributes**.
5. 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.

16

Managing Reservations for Online Sessions

You can configure how Oracle Communications Elastic Charging Engine (ECE) reserves your customers' resources for online sessions.

Topics in this document:

Configuring Reservation Expiration and Validity

In an online session, the network sends the following to ECE:

- **Usage updates**: Keeps ECE informed about the balance impact of an event. In response, ECE tells the network if the balance is sufficient to continue the session, or if reauthorization is needed.
- **Reauthorization requests**: Requests an extension of the session. In response, ECE determines whether the customer's balance is sufficient and, if so, reauthorizes the call.

ECE determines how many resources to reserve for a usage session and when the usage session expires by using the following information passed in the requests:

- **Reservation duration**: This amount is used to calculate the amount of resources to reserve for the usage session. For example, if the duration is 20 minutes and the rate is $2 per minute, ECE reserves $40 for the usage session. At the end of the duration, the client must ask for a reauthorization to extend the usage session.

  You do not want the duration to be too low because it takes network activity to report usage. You also do not want the value to be too high because that increases the risk of revenue leakage if the customer uses up his balance before the reservation expires.
- **Validity time:** This specifies how long the reservation is valid. When the validity time ends, the client must send a usage report to ECE. If the network mediation client does not communicate the used service units (USU) within the validity time, ECE considers the reserved balances to be available for subsequent session requests. The available reserved balances are cleaned up by housekeeping processes when the session terminates.

  The validity time is set for each session and is reset whenever an interim request is received for the session. After the validity time for a session expires, any reserved balances are released and become available to other active sessions for the same charge offer.

ECE calculates when the reservation expires by adding together the reservation duration and validity time. For example, if the usage request specifies a reservation duration of 240 seconds and a validity time of 600 seconds, the reservation expires in 840 seconds.

When ECE receives a usage request that does not specify a validity time or reservation duration, ECE uses the default values specified.

To configure the default values for reservation expiration and validity:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.reservationConfig**.
4. Expand **Attributes**.
5. Specify the defaults for the following attributes:
   - **validityTime** (Long data type): Enter the amount of time, in seconds, that a reservation remains valid. The default is one hour.
   - **reservationDuration** (Long data type): Enter the amount of time, in seconds, that is used to calculate the amount of resources to reserve. The default is one hour.

   For detailed information, see **ReservationConfigMBean** in the **BizParamConfigMBean** package of Elastic Charging Engine Java API Reference.

Configuring a Minimum Quantity for Reservation

You can configure a minimum reservation quantity for charging events. If the customer does not have enough balance to reserve the minimum quantity of the charging event, ECE tells the network that there is not enough in the balance to fulfill the request.

> **Note:**
>
> 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.

To configure the minimum quantity for reservation:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.reservationConfig**.
4. Expand **Operations**.
5. Select **setMinAuthorizedQuota**.
6. 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.

> **Note:**
>
> 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.

7. Click the **setMinAuthorizedQuota** button.

Configuring Reservation Quota for Services

When ECE receives a usage request that does not specify a requested amount, ECE uses a default usage amount. You configure a systemwide initial quota and a systemwide incremental quota for each combination of service and RUM. When initiating a call, ECE applies the initial quota for the reservation. For update requests, ECE applies the incremental quota for the reservation.

To configure the reservation quota for services:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.reservationConfig**.
4. Expand **Operations**.
5. For *each service* 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.

      > **Note:**
      >
      > 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 service-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 service-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.

Managing Dynamic Quotas for Online Sessions

Dynamic quota allows you to allocate the quota dynamically for each parallel session of a subscriber based on the rules you configure in Pricing Design Center (PDC). For configuring dynamic quota selectors, see "Configuring Dynamic Quota" in *PDC Creating Product Offerings*.

If dynamic quota selector rules are configured for a service-event combination, ECE does one of the following depending on whether the network passes a requested service unit (RSU) in a usage request:

- **An RSU is passed in the usage request**: ECE evaluates and applies the dynamic quota selector rules to the usage request to derive the allocated quota. ECE returns the derived quota (in the GSU) and the following quota attributes in the usage response to the network:
  - **Quota holding time.** Specifies how long a granted quota can be idle before releasing the reservation.
  - **Volume quota threshold.** Specifies how much of the granted quota must be consumed before a subscriber can request additional quota. This attribute is configured per service, event, and number of granted units.
  - **Validity time.** Specifies whether validity time can be set to a fixed value per service-event combination at runtime. This attribute is independent of the number of units in the granted quota.
- **An RSU is not passed in the usage request**: ECE does not return a quota in its response to the network.

If dynamic quota selector rules are *not* configured for a service-event combination, ECE uses the default quota configuration for deriving the quota and quota attributes.

> **Note:**
>
> You can customize the dynamic quota allocation to suit your business requirements. For more information, see "Sample Extensions".

Triggering RAR Notifications for Ongoing Sessions

When you use dynamic quotas for long running sessions to reduce network signaling, you can trigger server-initiated reauthorization requests to get the exact reservation balance before performing other business operations.

To generate server-initiated reauthorization requests, you must generate RAR notifications. To generate these notifications, you can implement custom logic by using the following ECE extensions in the rating/charging flow:

- Pre-rating extension
- Post-rating extension
- Post-charging extension

For more information, see "Rating/Charging Extension – Triggering RAR Notifications".

17

Managing Rounding and Consumption Rules

You can configure how Oracle Communications Elastic Charging Engine (ECE) rounds balance impacts and the order in which it consumes sub-balances.

Topics in this document:

Configuring Rounding for a Resource

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. These rules can be different for each processing stage. For information on configuring the rounding rules, see "Adding Rounding Rules for Specific Events" in *PDC Online Help*.

However, you can configure system-wide rounding in ECE for currency and noncurrency resources to apply the rule across all processing stages.

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 USD.

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.

To configure rounding for a resource:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".

2. Expand the **ECE Configuration** node.
3. Expand **charging.server**.
4. Expand **Attributes**.
5. Specify values for the following currency and noncurrency resource attributes as appropriate:
   - **currencyScale** or **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**.
   - **currencyRoundingMode** or **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 website:

     `https://docs.oracle.com/javase/8/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 value 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 can configure a systemwide rounding rule to round up the fractional value of the authorized resource.

Rounding up the authorized resource quantity may result in customers exceeding their credit limits. Configure this only if your business requires that your customers must be able to use all of their balances.

To configure whether to round up the fractional value of the authorized resource quantity by authorizing an additional RUM unit:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.server**.
4. Expand **Attributes**.
5. 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 Systemwide Consumption Rules for Balances

When more than one validity-based sub-balance 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, see "Specifying the Order in Which Sub-Balances Are Consumed" in *PDC Creating Product Offerings*.
- Configuring consumption rules in BRM, see "Specifying the Order in Which Resource Sub-Balances Are Consumed" in *BRM Configuring Pipeline Rating and Discounting*.

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 configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.server**.
4. Expand **Attributes**.
5. 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**.

Configuring the Consumption Order for Offers with the Same Priority (Release 15.0.1 or later)

ECE consumes allowances from offers based on their configured priority, from the highest priority to the lowest priority. If multiple offers have the same priority, ECE determines which offer to consume allowances from first based on the value of the **offerSelectionModeOnEquiPriorityOffers** entry in your **charging-settings.xml** file or your **oc-cn-ece-helm-chart/values.yaml** file:

- **START_TIME**: ECE consumes the allowance from the offer with the earliest validity start date. This is the default.
- **END_TIME**: ECE consumes the allowance from the offer with the earliest validity end date.

For example, assume a customer has purchased the following offers, which have the same priority:

- Offer A grants 1 GB of data with a validity period from Jan 1 through Jan 30.
- Offer B grants 200 MB of data with a validity period from Jan 10 through Jan 20.

On Jan 15, the customer downloads 100 MB of data. If the entry is set to **START_TIME**, ECE first consumes the data allowance from Offer A. If the entry is set to **END_TIME**, ECE first consumes the data allowance from Offer B.

During ECE runtime, you configure the consumption order as follows:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.server**.
4. Expand **Attributes**.
5. Select the **offerSelectionModeOnEquiPriorityOffers** attribute and set it to **START_TIME** or **END_TIME**.

Configuring Balance Impact Rounding in ECE

You can configure rounding for a currency and noncurrency balance impact by specifying rules for how Oracle Communications Elastic Charging Engine (ECE) rounds the rating impact amounts it calculates. The rules you specify are applied at the system level so the rounding of the rating impact amounts apply across all charges, discounts, and chargeshares. The rounding is also applied to reverse rating calculations and taxation calculations.

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**.

See the Java SE API Reference documentation for information about using the Java math rounding enum.

For information about configuring rounding for currency and noncurrency balance impacts, see "Configuring Rounding for a Currency Balance Impact " and "Configuring Rounding for a Noncurrency Balance Impact".

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 USD.

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.

Configuring Rounding for a Currency Balance Impact

You can configure rounding for a currency balance impact by specifying rules for how ECE rounds the rating impact amounts it calculates.

The rules you specify are applied as a systemwide configuration for rounding the rating impact amounts of all charges, discounts and chargeshares. The rounding is also applied to all reverse rating calculations and taxation calculations.

See "Configuring Rounding for a Noncurrency Balance Impact" for information about configuring rounding for noncurrency balance impacts.

To configure rounding for a currency balance impact:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.server**.
4. Expand **Attributes**.
5. 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.

  See the Java SE API Reference documentation for information about using the Java math rounding enum.

  For example, enter **UP** to round up away from zero or **DOWN** to round down towards zero.

  The default is **HALF_UP**.

6. Save your changes.

Configuring Rounding for a Noncurrency Balance Impact

You can configure rounding for a noncurrency balance impact. The rules you specify are applied as a systemwide setting and apply for rounding the rating 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 balance impact:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.server**.
4. Expand **Attributes**.
5. 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.

     See the Java SE API Reference documentation for information about using the Java math rounding enum.

     For example, enter **UP** to round up away from zero or **DOWN** to round down towards zero.

     The default is **HALF_UP**.
6. Save your changes.

For information about configuring rounding for currency balance impacts, see "Configuring Rounding for a Currency Balance Impact ".

Configuring Rounding When Authorizing Multiple RUMs

When ECE receives an online charging request for a specified amount of usage, ECE uses the customers product offerings to determine if the customer's balance amount is sufficient for the requested amount. If the request applies to multiple RUMs; for example, occurrence and duration, fractional values of the authorized balance quantity might result. You can enable ECE to round up the authorized balance quantities to the nearest whole number.

If your business requires that your customers must be able to use *all* of their balances, configure ECE to round up the authorized balance quantity. However, rounding up the authorized balance quantity may result in customers exceeding their credit limits.

You configure whether to round up the fractional value of the impacted balance quantity as a systemwide setting. ECE authorizes an additional RUM unit when the quantity is a fraction.

To configure rounding when authorizing multiple RUMs:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole.
2. Expand the **ECE Configuration** node.
3. Expand **charging.server**.
4. Expand **Attributes**.
5. 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**.
6. Save your changes.

> **Note:**
>
> When discounting custom events, you may encounter the following ECE exception:
>
> ```
> RatingMessageBundle-6000: The rate plan failed to evaluate. The cause
> is probably due to a misconfiguration in the rate plan graph.
> ```
>
> This may occur because the discount offer contains two rating periods of different RUMs and RUM units, and the total quantity cannot be calculated across them. In this scenario, ensure that you have a configuration to filter out the RUM and pass in the rating periods of the same RUM into the trigger. To discount both RUMs, use the filter to create separate configurations for each RUM.

Part V

Integrating with External Systems

This part provides information about integrating external applications with Oracle Communications Elastic Charging Engine (ECE). It contains the following chapters:

- Connecting ECE to a 5G Client
- Generating CDRs for External Systems
- Connecting ECE to a Diameter Client
- Connecting ECE to a RADIUS Client
- Configuring Policy-Driven Charging

18

Connecting ECE to a 5G Client

You can set up 5G network integration for online and offline charging by using the Oracle Communications Elastic Charging Engine (ECE) HTTP Gateway.

> **Caution:**
>
> Deploying charging for 5G with HTTP Gateway (5G CHF) requires a cloud native deployment of ECE and BRM components. The HTTP Gateway can be used only on an ECE cloud native system.

Topics in this document:

About the HTTP Gateway

You integrate 5G clients with ECE by using the HTTP Gateway. The HTTP Gateway sends usage requests to ECE for online and offline charging and then sends responses to the 5G network.

The HTTP Gateway supports the 5G service-based architecture and does the following:

- Receives ECE REST API requests from 5G clients and then translates them into batch request server (BRS) requests.
- Determines whether a usage request requires online or offline charging. See "About Determining the Charging Type" for details.
- Submits BRS requests to the ECE server.
- Receives responses from the ECE server and then translates them into REST API responses.
- Responds to the 5G clients.
- Consumes notifications from the ECE notification topic and then notifies the 5G clients by making a REST call to the URL stored in the system.
- Publishes details about ECE REST API requests that failed to the ECE failure Kafka topic.

When configured to do so, the HTTP Gateway can also send ECE REST API requests from 5G clients to the CDR Gateway for generating CDRs. For more information, see "Generating CDRs for External Systems".

About Determining the Charging Type

HTTP Gateway determines whether a usage request requires online charging or offline charging as follows:

- For INITIATE requests, based on the **multipleUnitUsage** block. If the block is present, the request needs online charging. If the block is missing, the request needs offline charging.
- For UPDATE requests, based on the value of the **quotaManagementIndicator** field in the request. If the value is set to **ONLINE_CHARGING**, the request needs online charging. If the field is missing or the value is set to **OFFLINE_CHARGING**, the request needs offline charging.
- For TERMINATE requests, based on the value of the **quotaManagementIndicator** field in the request. If the value is set to **ONLINE_CHARGING**, the request needs online charging. If the field is missing or the value is set to **OFFLINE_CHARGING**, the request needs offline charging.

For more information about these properties, see the Nchf Converged Charging endpoints and Nchf Offline-Only Charging endpoints in REST API for Elastic Charging Engine.

About Sending Notifications to HTTP Gateway

You can configure ECE to send the following notifications to the ECE Notification topic, where they are retrieved by the HTTP Gateway:

- Spending Limit Notification (SNR): Specifies that a subscriber has reached a spending threshold or limit.
- Reauthorization Request (RAR): Specifies that a reauthorization request is needed.

Configure HTTP Gateway to consume SNR and RAR notifications from the ECE Notification topic. When configured to do so, HTTP Gateway listens on the ECE Notification topic and filters notifications based on the notification type in the header. Depending on the notification type, HTTP Gateway does the following:

- Retrieves SNR notifications from the Kafka topic and makes a REST API call to **notifUri** to post the notification.
- Retrieves RAR notifications from the Kafka topic and makes a REST API call to post the notification to **notifUri** in the usage request.

Integrating HTTP Gateway with 5G Networks

For information about conformance with industry standards, see *ECE 5G CHF Protocol Implementation Conformance Statement*.

To integrate HTTP Gateway with your 5G network:

1. When you install ECE, do this:
   - Specify to use Apache Kafka topics and enter the details for your ECE notification topic, Suspense topic, ECE failure topic, and ECE overage topic.
   - Enable Network Repository Function (NRF) registration in one of your HTTP Gateway servers.
   - Specify the details for connecting to the BRM Gateway.

   For more information, see "Installing Elastic Charging Engine" in *ECE Installation Guide*.

2. During the ECE postinstallation process, do this:
   - Run the **kafka_post_install.sh** script to create your ECE notification topic, Suspense topic, ECE failure topic, and ECE overage topic. See "Creating Kafka Topics for ECE" in *ECE Installation Guide*.
   - Run the **post_Install.sh** script and choose to create only the Acknowledgment queue. See "Creating WebLogic JMS Queues for BRM" in *ECE Installation Guide*.
3. Configure the NRF registration details for each HTTP Gateway instance in your system. See "Configuring Registration Details for the HTTP Gateway Server".
4. Configure one or more NF services. See "Configuring NF Services".
5. Configure the HTTP Gateway to send usage requests to ECE Server for convergent charging. See "Configuring HTTP Gateway for Convergent Charging".
6. Define any custom service-event mappings in Pricing Design Center. See "Enabling Charging for Custom Events" in *PDC Creating Product Offerings*.
7. Edit your mediation specification file and load it into ECE. The mediation specification enables the HTTP Gateway to associate each HTTP request with its respective usage-request builder. See "Editing the HTTP Gateway Mediation Specification File".
8. Connect ECE to your Kafka Server topics. See "Connecting ECE to Kafka Topics".
9. Configure ECE to send notifications to HTTP Gateway through the ECE notification topic. See "Configuring ECE to Send Notifications to HTTP Gateway".
10. (Optional) Configure ECE to send information about failed usage requests to the ECE failure topic. See "Recording Failed ECE Usage Requests".
11. (Optional) Configure ECE to generate CDRs for any prepaid usage overage and send them to the ECE overage topic. See "Configuring ECE to Support Prepaid Usage Overage".
12. (Optional) Configure your charging function (CHF) operations to route communication through a Services Communication Proxy (SCP). See "Configuring Communication through SCP".
13. Start your HTTP Gateway. See "Starting the HTTP Gateway".

After the HTTP Gateway is set up, your 5G clients can start:

- Submitting ECE REST API requests to HTTP Gateway for online or offline charging by ECE. See "Using the ECE REST API".
- Sending unrated 5G usage events to HTTP Gateway so they can be converted into CDRs and sent to roaming partners, data warehousing system, and legacy billing systems. See "Generating CDRs for External Systems".

Configuring Registration Details for the HTTP Gateway Server

You can register:

- One or more HTTP Gateway server instances with an NRF registration server.
- One HTTP Gateway server instance with multiple NRF registration servers.
- One HTTP Gateway server instance with both primary and secondary NRF registration servers. In this case, when a primary NRF server goes down, the HTTP Gateway server registers with the secondary NRF server. See "Configuring Multiple Primary and Secondary NRF Registration Servers" for more information.

You do so by configuring the registration details in JConsole and then starting the HTTP Gateway instance.

To build an NRF registration request for an HTTP Gateway server instance:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.nfProfileConfigurations**.
4. Expand **Attributes**.
5. Specify the registration values for the fields in Table 18-1.

Table 18-1 Fields for Configuring NRF Registration

Attribute Name	Mandatory	Description
name	Yes	The name of the HTTP Gateway instance.
clusterName	Yes	The name of the cluster that the HTTP Gateway belongs to.
allowedNfDomains	No	The network function (NF) domains that are allowed to access the HTTP Gateway server. Enter a regular expression for the domains according to the ECMA-262 dialect [8]. If not provided, any NF domain is allowed to access the HTTP Gateway server.
allowedNfTypes	No	The type of NFs that are allowed to access the HTTP Gateway server, such as AMF or SMF. If not provided, any NF type is allowed to access the HTTP Gateway server.
allowedNssaisSd	No	The S-NSSAI Slice Differentiator (SD) ID of the network slices that are allowed to access the HTTP Gateway server. If not provided, any slice is allowed to access the HTTP Gateway server.
allowedNssaisSst	No	The S-NSSAI Slice/Service Type (SST) ID of the network slices that are allowed to access the HTTP Gateway server. If not provided, any slice is allowed to access the HTTP Gateway server.
allowedPlmnsMcc	No	The Mobile Country Code (MCC) of the PLMNs that are allowed to access the HTTP Gateway server. If not provided, any PLMN is allowed to access the HTTP Gateway server.
allowedPlmnsMnc	No	The Mobile Network Code (MNC) of the PLMNs that are allowed to access the HTTP Gateway server. If not provided, any PLMN is allowed to access the HTTP Gateway server.
capacity	No	The static capacity information in the range of 0-65535, expressed as a weight relative to other NF instances of the same type. The default is **0**. If the capacity is also present in the **nfServiceList** parameters, those will have precedence over this value.
customInfo	No	The specific data for custom NFs.
defaultNotificationSubscriptions CallbackUri	No	The callback URI for the default notification type.
defaultNotificationSubscriptions N1MessageClass	No	The information element (IE) that is used to identify the class of the N1 message type.

Table 18-1 (Cont.) Fields for Configuring NRF Registration

Attribute Name	Mandatory	Description
defaultNotificationSubscriptions N2InformationClass	No	The information element (IE) that is used to identify the class of the N2 message type.
defaultNotificationSubscriptions NotificationType	No	The type of notification for the corresponding callback URI.
fqdn	Yes	The FQDN of the HTTP Gateway server. For AMF, the FQDN registered with the NRF is the AMF name. For example: **chf-demo.novalocal**.
gpsiRangeListEnd	No	The ending range for the list of GPSIs that can be served by the CHF instance. The default is **100008**. If not provided, the CHF can serve any GPSI.
gpsiRangeListPattern	No	The pattern for the list of GPSI ranges that can be served by the CHF instance. The default is **^extid-.+@oracle1.com$**. If not provided, the CHF can serve any GPSI.
gpsiRangeListStart	No	The starting range for the list of GPSIs that can be served by the CHF instance. The default is **10000**. If not provided, the CHF can serve any GPSI.
heartBeatTimer	No	The time, in seconds, between two consecutive heartbeat messages from an NF Instance to the NRF.
httpGatewayName	Yes	The name of the HTTP Gateway that this property configuration belongs to.
interPlmnFqdn	No	The FQDN that is used for inter-PLMN routing as specified in 3GPP 23.003 [12]. This is required if the HTTP Gateway needs to be discoverable by other NFs in a different PLMN.
ipv4Addresses	No	The IPv4 addresses of the HTTP Gateway.
ipv6Addresses	No	The IPv6 addresses of the HTTP Gateway.
load	No	The current load percentage of the HTTP Gateway, ranging from 0 to 100.
locality	No	Operator-defined information about the location of the HTTP Gateway instance, such as the geographic location.
nfProfileChangesInd	No	Whether the IE is absent in the request to the NRF and may be included by the NRF in the NFRegister or NFUpdate response: • **true**: The NF Profile contains changes. • **false**: The complete NF Profile. This is the default.
nfProfileChangesSupportInd	No	Whether the IE may be present in the NFRegister or NFUpdate request and will be absent in the response: • **true**: The NF Service Consumer supports receiving NF Profile Changes in the response. • **false**: The NF Service Consumer does not support receiving NF Profile Changes in the response. This is the default.
nfServicePersistence	No	If present, and set to **true**, it indicates that the different instances of an NF Service in this NF instance, supporting the same API version, can persist their resource state in shared storage. Thus, these resources are available after a new NF service instance supporting the same API version is selected by a NF Service Consumer (see 3GPP 23.527 [27]). Otherwise, it indicates that the NF Service Instances of the same NF Service cannot share a resource state inside the NF Instance. Possible values are **true** or **false**.

Table 18-1 (Cont.) Fields for Configuring NRF Registration

Attribute Name	Mandatory	Description
nfStatus	Yes	The status of the HTTP Gateway server: • **REGISTERED** • **SUSPENDED** • **UNDISCOVERABLE** The default is **REGISTERED**.
nfType	Yes	The type of NF. Set this to **CHF**. **Note:** ECE supports only **CHF**.
nrfHttp2Enable	No	Whether the endpoint URL of the NRF registration server (**nrfRestEndPointUrl**) uses the HTTP/2 protocol: • **true**: The HTTP/2 protocol is used. This is the default. • **false**: The HTTP/1 protocol is used.
nrfRestEndPointUrl	Yes	The endpoint URL of the NRF registration server. To register this HTTP Gateway instance with multiple NRF registration servers, list the endpoint URLs separated by a comma. For example: **http://localhost:8080,http://localhost:8081**. If this field is missing, NRF URL registration will not occur.
nrfSecondarySiteRestEndPointUrls	No	The endpoint URL of one or more secondary NRF servers. The list can include NRF servers on the same site and on different sites. See "Configuring Multiple Primary and Secondary NRF Registration Servers" for more information.
nsiList	No	The list of Network Slice Instances (NSIs) of the HTTP Gateway. If not provided, the HTTP Gateway can serve any NSI.
perPlmnSnssaiListPlmnIdMcc	No	The Mobile Country Code (MCC) value for **plmnSnssai**.
perPlmnSnssaiListPlmnIdMnc	No	The Mobile Network Code (MNC) value for **plmnSnssai**.
perPlmnSnssaiListSd	No	The Slice Differentiator (SD) value for **plmnSnssai**.
perPlmnSnssaiListSst	No	The Slice/Service Type (SST) value for **plmnSnssai**.
plmnListMcc	No	The MCC value for **plmnList**.
plmnListMnc	No	The MNC value for **plmnList**.
plmnRangeListEnd	No	The ending range for the list of PLMNs (including the PLMN IDs of the CHF instance) that can be served by the CHF instance. The default is **333333**. If not provided, the CHF can serve any PLMN.
plmnRangeListPattern	No	The pattern for the list of PLMNs (including the PLMN IDs of the CHF instance) that can be served by the CHF instance. If not provided, the CHF can serve any PLMN.
plmnRangeListStart	No	The starting range for the list of PLMNs that can be served by the CHF instance. The default is **100000**.
priority	No	The priority of this NF service relative to other services of the same type, where lower values indicate higher priority. The value must be in the range of 0-65535. The default is **1**. **Note**: The NRF may overwrite the received priority value when exposing an NFProfile.
recoveryTime	No	The timestamp for when the NF was started or restarted, in DateTime format.

Table 18-1 (Cont.) Fields for Configuring NRF Registration

Attribute Name	Mandatory	Description
snssaisSdl	No	The S-NSSAIs of the NF. When present, this IE represents the list of S-NSSAIs supported in all the PLMNs listed in the plmnList IE. If not provided, the NF can serve any S-NSSAI.
snssaisSst	No	The S-NSSAIs of the NF. When present, this IE represents the list of S-NSSAIs supported in all the PLMNs listed in the plmnList IE. If not provided, the CHF can serve any S-NSSAI.
supiRangeListEnd	No	The end of a list of GPSI ranges that can be served by the CHF instance, such as 1009. The default is **10008**. If not provided, the CHF can serve any GPSI.
supiRangeListPattern	No	The pattern for a list of GPSI ranges that can be served by the CHF instance. The default is **^nai-450081.+@.+org$**. If not provided, the CHF can serve any GPSI.
supiRangeListStart	No	The start of a list of GPSI ranges that can be served by the CHF instance. The default is **10000**. If not provided, the CHF can serve any GPSI.

Configuring Multiple Primary and Secondary NRF Registration Servers

You can configure an HTTP Gateway server instance to register with multiple primary and secondary NRF registration servers. In this case, when the heartbeat to a primary NRF server fails, HTTP Gateway retries the heartbeat for a configurable number of times. If all retries fail, HTTP Gateway initiates a registration request on the secondary NRF server. If all retries fail on the secondary NRF server, HTTP Gateway initiates a registration request on the next secondary NRF server.

If, later on, the primary NRF server comes back online, HTTP Gateway re-registers with the primary NRF server, starts the heartbeat in the primary NRF server, and then deregisters from the secondary NRF server.

To configure an HTTP Gateway instance to register with both primary and secondary NRF registration servers:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.nfProfileConfigurations**.
4. Expand **Attributes**.
5. Specify the value for the following attributes:
   - **nrfRestEndPointUrl**: Set this to the endpoint URL of the primary NRF server.

     To configure multiple primary NRF servers, list the endpoint URLs separated by a comma (**,**).
   - **nrfSecondarySiteRestEndPointUrls**: Set this to the endpoint URL of the secondary NRF server.
     - To configure multiple secondary NRF servers for one primary NRF server, list the endpoint URLs, in order, separated by a semicolon (**;**). HTTP Gateway uses the secondary NRF servers in the order listed.

  - If you have multiple primary NRF servers and want to map each to a different secondary NRF server, separate the secondary endpoint URLs by a comma (**,**).
  - If a secondary NRF server is not available for one of the primary NRF servers, enter **NA** or **None**.

Example: Configure One Primary with No Secondaries

This example specifies to configure a primary NRF server (**primaryNRF_1**) with no secondary NRF servers.

```
nrfRestEndPointUrl="www.primaryNRF_1.com"
secondaryNrfEndpointUrl=""
```

Example: Configure One Primary with Two Secondaries

This example specifies to configure a primary NRF server (**primaryNRF_1**) with two secondary NRF servers (**secondaryNRF_A** and **secondaryNRF_B**).

```
nrfRestEndPointUrl="www.primaryNRF_1.com"
secondaryNrfEndpointUrl="www.secondaryNRF_A.com;www.secondaryNRF_B.com"
```

Example: Configure One Primary with Two Secondaries, and Another Primary with One Secondary

This example specifies to configure:

- A primary NRF server (**primaryNRF_1**) with two secondary NRF servers (**secondaryNRF_A** and **secondaryNRF_B**)
- A primary NRF Server (**primaryNRF_2**) with one secondary NRF server (**secondaryNRF_C**)

```
nrfRestEndPointUrl="www.primaryNRF_1.com,www.primaryNRF_2.com"
secondaryNrfEndpointUrl="www.secondaryNRF_A.com;www.secondaryNRF_B.com,www.secondaryNRF_C
.com"
```

Example: Configure Three Primaries, Two with a Secondary and One with No Secondary

This example specifies to configure:

- A primary NRF server (**primaryNRF_1**) with one secondary NRF server (**secondaryNRF_A**)
- A primary NRF server (**primaryNRF_2**) with no secondary NRF server (**None**)
- A primary NRF Server (**primaryNRF_3**) with one secondary NRF server (**secondaryNRF_B**)

```
nrfRestEndPointUrl="www.primaryNRF_1.com,www.primaryNRF_2.com,www.primaryNRF_3.com"
secondaryNrfEndpointUrl="www.secondaryNRF_A.com,None,www.secondaryNRF_B.com"
```

Configuring NF Services

You must configure at least one NF service for HTTP Gateway. By default, HTTP Gateway includes one, but you can add more.

To add or remove an NF service configuration, do this:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole.
2. Expand the **ECE Configuration** node.
3. Expand **charging.nfServiceConfigurations**.

4. Expand **Operations**.
5. Do one of the following:
   - To add an NF service configuration, enter its name and then click **addNfServiceConfiguration**.
   - To remove an NF service configuration, enter its name and then click **removeNfServiceConfiguration**.

To configure an NF service, do this:

1. Access the ECE configuration MBeans.
2. Expand the **ECE Configuration** node.
3. Expand **charging.nfServiceConfigurations.***Instance_Name*.
4. Expand **Attributes**.
5. Specify the NF service configuration values for the attributes in Table 18-2.

Table 18-2 NF Service Configuration Attributes

Attribute Name	Mandatory	Description
allowedNfDomains	No	The network function (NF) domains that are allowed to access the service instance. Enter a regular expression for the domains according to the ECMA-262 dialect. If not provided, any NF domain is allowed to access the service instance.
allowedNfTypes	No	The type of NFs that are allowed to access the service instance. If not provided, any NF type can access the service instance.
allowedNssaisSd	No	The S-NSSAI Slice Differentiator (SD) ID of the network slices that are allowed to access the service instance. If not provided, any slice can access the service instance.
allowedNssaisSst	No	The S-NSSAI Slice/Service Type (SST) ID of the network slices that are allowed to access the service instance. If not provided, any slice can access the service instance.
allowedPlmnsMcc	No	The Mobile Country Code (MCC) of the PLMNs that are allowed to access the service instance. If not provided, any PLMN can access the service instance. When included, the **allowedPlmns** attribute does not need to include the PLMN IDs registered in the **plmnList** attribute of the NF Profile.
allowedPlmnsMnc	No	The Mobile Network Code (MNC) of the PLMNs that are allowed to access the service instance.
apiFullVersion	Yes	The full version number of the API as specified in 3GPP 29.501.
apiPrefix	No	The optional path segments used to construct the {apiRoot} variable of the different API URIs.
apiVersionInUri	Yes	The version of the service instance to be used in the URI for accessing the API.
capacity	No	The static capacity information in the range of 0-65535, expressed as a weight relative to other services of the same type. The default is **50**. The capacity and priority parameters, if present, are used for NF selection and load balancing.
defaultNotificationSubscriptionsCallbackUri	No	The callback URI for the default notification type.

Table 18-2 (Cont.) NF Service Configuration Attributes

Attribute Name	Mandatory	Description
defaultNotificationSubscriptions N1MessageClass	No	The information element (IE) that is used to identify the class of the N1 message type.
defaultNotificationSubscriptions N2InformationClass	No	The information element (IE) that is used to identify the class of the N2 message type.
defaultNotificationSubscriptions NotificationType	No	The type of notification for the corresponding callback URI.
expiry	No	The expiration date and time of the NF service. The default is **2020-12-01T18:55:08.871Z**.
fqdn	No	The FQDN of the NF service instance.
interPlmnFqdn	No	The FQDN that is used for inter-PLMN routing as specified in 3GPP 23.003. This is required if the service instance needs to be discoverable by other NFs in a different PLMN.
ipv4Address	No	The IPv4 address.
ipv6Address	No	The IPv6 address.
load	No	The current load percentage of the NF service, ranging from 1 through 100. The default is **5**.
name	Yes	The name of the NF service.
nfServiceStatus	Yes	The status of the NF service instance: • **REGISTERED** • **SUSPENDED** • **UNDISCOVERABLE** The default is **REGISTERED**.
port	No	The port number. The default is **0**.
primaryChfServiceInstance	No	The specific data for a CHF service instance. Include this IE if the CHF service instance serves as a secondary CHF instance of another primary CHF. When present, set it to the **serviceInstanceId** of the primary CHF service instance.
priority	No	The priority used for service selection (relative to other services of the same type), ranging from 0 through 65535. Lower values indicate a higher priority. The default is **1**. The NRF may overwrite the received priority value when exposing an NFProfile with the Nnrf_NFDiscovery service.
recoveryTime	No	The timestamp when the NF service was started or restarted. For example, 2019-08-03T18:55:08.871Z. The format should be of type DateTime.
scheme	Yes	The URI scheme, such as **http** or **https**. The default is **http**.
secondaryChfServiceInstance	No	Include this IE if the CHF service instance serves as a primary CHF instance of another secondary CHF. When present, set it to the **serviceInstanceId** of the secondary CHF service instance. Do not set this IE when **primaryChfServiceInstance** is present.
serviceInstanceId	Yes	The unique ID of the service instance within a given NF instance. The default is **chf1**.

Table 18-2 (Cont.) NF Service Configuration Attributes

Attribute Name	Mandatory	Description
serviceName	Yes	The name of the service instance: • **nnrf-nfm** • **nnrf-disc** • **nudm-sdm** • **nudm-uecm** • **nudm-ueau** • **nudm-ee** • **nudm-pp** • **namf-comm** • **namf-evts** • **namf-mt** • **namf-loc** • **nsmf-pdusession** • **nsmf-event-exposure** • **nausf-auth** • **nausf-sorprotection** • **nausf-upuprotection** • **nnef-pfdmanagement** • **npcf-am-policy-control** • **npcf-smpolicycontrol** • **npcf-policyauthorization** • **npcf-bdtpolicycontrol** • **npcf-eventexposure** • **npcf-ue-policy-control** • **nsmsf-sms** • **nnssf-nsselection** • **nnssf-nssaiavailability** • **nudr-dr** • **nlmf-loc** • **n5g-eir-eic** • **nbsf-management** • **nchf-spendinglimitcontrol** • **nchf-convergedcharging** • **nnwdaf-eventssubscription** • **nnwdaf-analyticsinfo** The default is **nchf-convergedcharging**.
supportedFeatures	No	The supported features of the NF Service instance.
transport	No	The transport protocol. The default is **TCP**.
httpGatewayName	Yes	The name of the HTTP Gateway that this property configuration belongs to.
clusterName	Yes	The name of the cluster that the HTTP Gateway belongs to.

Configuring HTTP Gateway for Convergent Charging

Configure the HTTP Gateway to send usage requests to ECE Server for convergent charging, and to consume SNR and RAR notifications from the ECE notification topic.

To configure the HTTP Gateway for convergent charging:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole.
2. Expand the **ECE Configuration** node.
3. Expand **charging.httpGatewayConfigurations**.
4. Expand **Attributes**.
5. Specify values for the fields in Table 18-3.

Table 18-3 Fields for Converged Charging

Name	Default	Description
serverSslKeyStore	"httpGatewayServer.jks"	The name of the KeyStore file for ECE server.
serverSslKeyStoreType	"@ECE_HTTPGATEWAY_KEYSTORE_TYPE@"	The SSL KeyStore type for ECE server, such as **JKS** or **pkcs12**.
serverSslKeyStoreAlias	"@ECE_HTTPGATEWAY_KEYSTORE_ALIAS@"	The alias name for the ECE server SSL KeyStore.
walletLocation	"@ECE_WALLET_LOCATION@"	The path to the ECE wallet.
snrHttp2Enable	"true"	Whether HTTP Gateway uses the HTTP/2 protocol: • **true**: The HTTP/2 protocol is used. • **false**: The HTTP/1 protocol is used.
retryIntervalInMillis	"5000"	The amount of time, in milliseconds, between reconnection attempts to ECE server.
notificationListenerConnectionPoolSize	"10"	The number of threads used by the HTTP Gateway instance for retrieving notifications from the ECE Notification topic.

6. Expand **charging.HttpGatewayConfigurations.***name*, where *name* is the name of the HTTP Gateway instance.
7. Expand **Attributes**.
8. Specify values for the fields in Table 18-4.

Table 18-4 Fields for an HTTP Gateway Instance

Name	Default	Description
name	"@HTTPGATEWAY_NAME@"	The name of the HTTP Gateway instance.
clusterName	"@CLUSTER_NAME@"	The name of the cluster that the HTTP Gateway belongs to.
serverHttp2Enabled	"@HTTPGATEWAY_HTTP2_ENABLED@"	Whether ECE Server uses the HTTP/2 protocol: • **true**: The HTTP/2 protocol is used. • **false**: The HTTP/1 protocol is used.
serverPort	"@HTTPGATEWAY_SERVER_PORT@"	The HTTPS port number of the server on which HTTP Gateway resides.
serverHttpPort	"@HTTPGATEWAY_SERVER_HTTP_PORT@"	The HTTP port number of the server on which HTTP Gateway resides.

Table 18-4 (Cont.) Fields for an HTTP Gateway Instance

Name	Default	Description
nrfHeartBeatRetryCount	"4"	The number of times the heartbeat retries if it fails during registration.
serverSslEnabled	"@HTTPGATEWAY_SERVER_SSL_ENABLED@"	Whether SSL communication is enabled between your HTTP client and the HTTP Server (**true**) or not (**false**).
processingThreadPoolSize	"200"	The number of threads used by the HTTP Gateway instance to process a set of incoming usage requests.
processingQueueSize	"32768"	The number of incoming usage requests that can be processed simultaneously by the HTTP Gateway.
kafkaBatchSize	"10"	The size of the Kafka batch.
externalTrafficInfo	""	The location of the header URL information.
nrfRetryIntervalInSecond	"60"	The amount of time to wait before retrying a connection to an NRF instance.
nrfRetryCount	"3"	The number of retry attempts for each NRF instance before attempting to register on a new NRF site.

Editing the HTTP Gateway Mediation Specification File

The mediation specification file enables HTTP Gateway to associate each ECE REST API request with its respective usage-request builder. HTTP Gateway uses the mediation specification to determine which service and event combination applies to an incoming ECE REST API request, enabling it to select the event definition that applies to the event to be rated.

You configure HTTP Gateway to base its selection of event definitions on any combination of the following in the request:

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

From the preceding values, HTTP Gateway derives the following fields, which uniquely identify the event definition to use for building the BRS request for ECE:

- ProductType (service)
- EventType
- Version

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***/sample_data/config_data/specifications/ece_end2end/http_mediation.spec**.

   It is recommended to create only one mediation specification file. 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:
   - **Service-Identifier**: The Service-Identifier is a placeholder.
   - **Rating-Group**: The Rating-Group value sent in the ECE REST API request.
   - **ProductType**: The service you have defined for the event.
   - **EventType**: The event definition you have defined for the event.
   - **Version**: The version number of the event definition object that you want to apply to the event.

   Define this information for each event definition object defined in the mediation table.

   For each received request, HTTP Gateway correlates the Rating-Group, and Event-Timestamp 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, service, and event).
3. (Optional) In the **ValidFrom** field of the table, set a future date and time when you want HTTP Gateway to recognize a newly deployed event definition object.

   For example, to have requests processed according to a new specification on December 16, 2020, you would enter:

   ```
   |   ValidFrom
    | "2020-12-16T12:01:01"
   ```

   You can also specify a time zone. For example,

   ```
   |   ValidFrom
    | "2020-12-16T12:01:01 PST"
   ```

   If a time zone is not sent, the **ValidFrom** field is set to UTC.
4. Save the **http_mediation.spec** file in the directory where you save your configuration data.
5. Verify that the directory specified in the *ECE_home***/config/management/migration-configuration.xml** file is the directory where you save your configuration data.
6. Run 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 into the repository, HTTP 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. Restart the HTTP Gateway.
8. Load the mediation specification file into the ECE server by using the **configLoader** utility.

Connecting ECE to Kafka Topics

You can connect ECE to the following ECE Kafka topics so that ECE can publish notifications, failed usage requests, and CDRs with usage overage information to them:

- **ECE notification topic**: Stores notifications from ECE.
- **Suspense topic**: Stores failed notifications from ECE.

- **ECE failure topic**: Stores details about failed ECE usage requests, such as the user ID and request payload. See "Recording Failed ECE Usage Requests" for more information.
- **ECE overage topic**: Stores overage records, which contain details about usage overage amounts for prepaid customers. See "Configuring ECE to Support Prepaid Usage Overage" for more information.

To connect ECE to your Kafka topics:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.kafkaConfigurations**.
4. Expand **Attributes**.
5. Specify the Kafka configuration values for the attributes in Table 18-5.

Table 18-5 kafkaConfiguration Properties

Property Name	Description
name	The name of your ECE cluster.
hostname	The host name and port number of the machine in which Apache Kafka is installed. If it contains multiple Kafka brokers, create a comma-separated list.
topicName	The name of the Kafka topic where ECE will publish notifications.
suspenseTopicName	The name of the Kafka topic where failed notifications are published.
failureTopicName	The name of the Kafka topic where ECE will publish details about failed usage requests.
overageTopicName	The name of the Kafka topic where ECE will publish overage records with information about your prepaid customer's usage overage during online sessions.
partitions	The number of Kafka partitions in your topics. The recommended number to create is calculated as follows: [(Max HTTP Gateway Nodes) + (Max Diameter Gateway Nodes * Max Diameter Clients) + (1 for BRM Gateway) + (1 for Internal Notifications)] For example, if you have 2 HTTP Gateway nodes, 4 Diameter Gateway nodes, 10 Diameter Gateway clients, and a BRM Gateway, you would need [(2 + (4 * 10) + 1 + 1) = 44 Kafka partitions. Arbitrarily, you can set this to a maximum value.
failurePartitions	The number of Kafka partitions in your ECE failure topic.
kafkaProducerReconnectionInterval	The amount of time, in milliseconds, the Notification Publisher waits before attempting to reconnect to the Kafka topic.

Table 18-5 (Cont.) kafkaConfiguration Properties

Property Name	Description
kafkaProducerReconnectionMax	The maximum amount of time, in milliseconds, the Notification Publisher waits before attempting to reconnect to a broker that has repeatedly failed to connect. The **kafkaProducerReconnectionInterval** will increase exponentially for each consecutive connection failure, up to this maximum.
kafkaDGWReconnectionInterval	The amount of time, in milliseconds, Diameter Gateway waits before attempting to reconnect to the Kafka topic.
kafkaDGWReconnectionMax	The maximum amount of time, in milliseconds, Diameter Gateway waits before attempting to reconnect to a broker that has repeatedly failed to connect. The **kafkaDGWReconnectionInterval** will increase exponentially for each consecutive connection failure, up to this maximum.
kafkaBRMReconnectionInterval	The amount of time, in milliseconds, BRM Gateway waits before attempting to reconnect to the Kafka topic.
kafkaBRMReconnectionMax	The maximum amount of time, in milliseconds, BRM Gateway waits before attempting to reconnect to a broker that has repeatedly failed to connect. The **kafkaBRMReconnectionInterval** will increase exponentially for each consecutive connection failure, up to this maximum.
kafkaHTTPReconnectionInterval	The amount of time, in milliseconds, HTTP Gateway waits before attempting to reconnect to the Kafka topic.
kafkaHTTPReconnectionMax	The maximum amount of time, in milliseconds, HTTP Gateway waits before attempting to reconnect to a broker that has repeatedly failed to connect. The **kafkaHTTPReconnectionInterval** will increase exponentially for each consecutive connection failure, up to this maximum.

Configuring ECE to Send Notifications to HTTP Gateway

You can enable ECE to send SNR and RAR notifications to the ECE Notification topic, where they will be retrieved by the HTTP Gateway. To do so:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.server**.
4. Expand **Attributes**.
5. Set the **kafkaEnabledForNotifications** property to **true**.

6. Expand **charging.notification**.
7. Expand **Attributes**.
8. Set the **rarNotificationMode** and **spendingLimitNotificationMode** properties to one of the following:
   - **NONE**: ECE does not send this notification type.
   - **ASYNCHRONOUS**: ECE sends an asynchronous notification to the ECE Notification topic.

     If configured to do so, the HTTP Gateway will consume the notification from the Kafka topic and dispatch it through a REST API call.

Recording Failed ECE Usage Requests

ECE may occasionally fail to process usage requests. For example, a data usage request could fail because a customer has insufficient funds. You can configure ECE to publish details about failed usage requests, such as the user ID and request payload, to the ECE failure topic in your Kafka server. Later on, you can reprocess the usage requests or view the failure details for analysis and reporting.

To enable the recording of failed ECE usage requests:

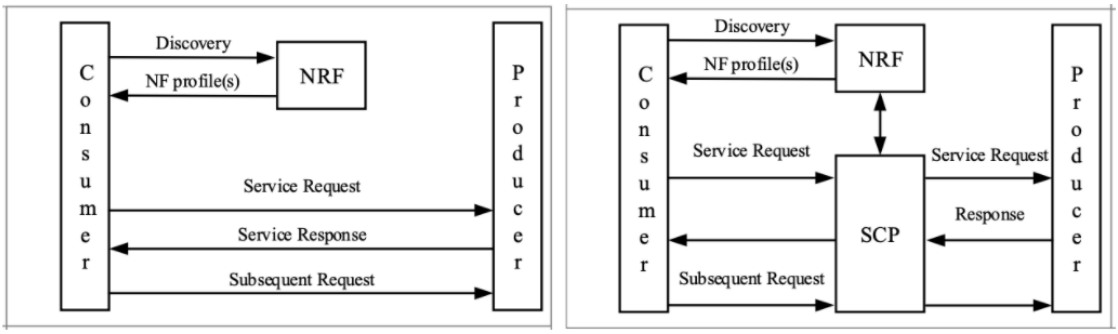
1. Access the ECE configuration MBeans in a JMX editor, such as JConsole.
2. Expand the **ECE Configuration** node.
3. Expand **charging.kafkaConfigurations**.
4. Expand **Attributes**.
5. Set the **persistFailedRequestsToKafkaTopic** property to **true**.

> **Note:**
>
> You can also examine the HTTP Gateway log files to determine why a usage request failed. See "Troubleshooting Failed Usage Requests" in *BRM System Administrator's Guide* for more information.

Configuring Communication through SCP

HTTP Gateway supports the communication models shown in Figure 18-1 for the charging function (CHF) operations.

Figure 18-1 Supported Communication Models for CHF Operations



If your system routes communication between the charging functions (CHF) and other network functions through an Oracle Services Communications Proxy (SCP), perform these steps:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole.
2. Expand the **ECE Configuration** node.
3. Expand **charging.HttpGatewayConfigurations**.
4. Expand **Attributes**.
5. Select **scpAuthorities** and enter the URL of the primary and secondary SCP authority, delimited by commas. For example: scpAuthorities="scp1.example.com,scp2.example.com".
6. Expand **charging.HttpGatewayConfigurations.***name*, where *name* is the name of the HTTP Gateway instance.
7. Expand **Attributes**.
8. Select **nrfHeartBeatRetryCount** and enter the maximum number of tries for a heartbeat call. If all retries have been exhausted, HTTP Gateway starts registering with a new NRF.
9. Select **nrfRetryCount** and enter the number of retries for NRF registration.

Starting the HTTP Gateway

When the HTTP Gateway starts, it automatically joins the Coherence cluster and gains access to ECE caches and invocation services that it uses to send requests to ECE. At start up, the HTTP Gateway checks for notifications in the ECE Notification topic.

To start the HTTP Gateway:

1. Ensure that the HTTP Gateway server and other components are started.
2. Start ECC:

   ```
   ./ecc
   ```

3. Start the HTTP Gateway:

   ```
   ecc:000> start httpGateway
   ```

Using the ECE REST API

After you set up the HTTP Gateway on your system, your 5G clients can submit requests to the ECE REST API.

The ECE REST API supports the following CHF operation types:

- Creating an initial quota reservation for a converged charging session. For example, initially reserving 500 MBytes for a data session.
- Updating the quota reservation for a converged charging session. For example, reserving an additional 100 MBytes for a data session that is in progress.
- Releasing the quota reservation when the converged charging session ends.
- Creating an initial request for an offline-only charging session.
- Updating a request for an offline-only charging session.
- Ending an offline-only charging session.
- Subscribing a customer to spending limit notifications.
- Updating a customer's spending limit subscription.
- Unsubscribing a customer from spending limit notifications.
- Creating a usage consumption resource.
- Deleting a usage consumption resource.
- Retrieving a subscriber's current usage consumption.
- Retrieving the current usage consumption for all subscribers.

For more information about these operation types, see REST API for Elastic Charging Engine.

You make calls to the ECE REST API through Swagger at this URL:

`https://`*`hostname`*`:`*`serverhttpsPort`*`/openapi-ui/index.html`**

where *hostname* is the host name of the machine on which HTTP Gateway is running, and *serverhttpsPort* is the port number on the HTTP Gateway server.

> **Note:**
>
> If your system's communication model includes an SCP, include the **Authority** header in all HTTP requests to the CHF operations. Set the **Authority** header to the host name and port number of the SCP Authority Server. For example:
>
> ```
> Authority: example.com:1534
> ```

19

Generating CDRs for External Systems

You can configure Oracle Communications Elastic Charging Engine (ECE) to generate call detail records (CDRs) for unrated 5G usage events. ECE does not use CDRs for convergent charging, so it does not generate them by default. You might want CDRs for roaming partners, data warehousing, and legacy billing systems.

> **Caution:**
>
> Generating CDRs for unrated 5G events requires a cloud native deployment of ECE and BRM components. The HTTP Gateway and CDR Gateway can be used only on an ECE cloud native system.

Topics in this document:

About Using the HTTP Gateway

By default, HTTP Gateway sends all 5G usage requests to ECE Server for online and offline charging.

You can configure HTTP Gateway to convert some of the usage requests into CDRs based on the charging type by enabling CDR generation. You can then send the CDRs to roaming partners, a data warehousing system, or legacy billing systems for rating.

When CDR generation is enabled, HTTP Gateway routes usage requests to one of the following, depending on your configurations and the charging type:

- ECE Server for charging. See "Using the ECE REST API".
- The CDR Gateway for generating CDRs. See "About Generating CDRs".

You specify where to route online charging requests and offline charging requests when you configure the HTTP Gateway. See "Configuring HTTP Gateway for CDR Generation".

About Generating CDRs

You can configure ECE to generate CDR files for unrated 5G usage events. You might want CDR files for roaming partners, data warehousing, and legacy billing systems. ECE can publish CDR files to your file system or a Kafka messaging service. In both cases, the files are

in JSON format. For details about the CDR format, see "CHF-CDR Format" in *ECE 5G CHF Protocol Implementation Conformance Statement*.

When configuring ECE to generate CDRs, you can specify whether to:

- Generate CDRs for online charging requests, offline charging requests, or both.
- Generate individual CDRs or aggregate multiple CDRs together according to the trigger criteria you specify.
- Purge completed CDRs from the CDR store that are older than a specified number of days.
- Purge orphaned CDRs from the store that are older than a specified number of seconds.

If you set up your ECE system for disaster recovery, you can also specify to:

- Mark partially processed CDRs in the CDR Store as incomplete.
- Mark when a duplicate usage update occurs in a CDR.
- Support a custom value for why a CDR session was closed.

For more information, see "About CDR Generator in an Active-Active System" in *BRM System Administrator's Guide*.

About Saving CDR Files to Disk

If you configure CDR Gateway to save JSON-formatted CDR files to disk, it stores the files to the directory you specify using the following file naming format:

```
ClusterName_StartTimeStamp_EndTimeStamp_SequenceNumber.Extension
```

For example: BRM_1654514133000_1654514134000_1.out

You set the file name extension, the maximum number of CDRs that can be written to a CDR output file, and the directory in which to store CDR files when you configure the CDR Formatter Plug-In. See "Configuring the CDR Formatter Plug-in".

About the CDR Generation Process

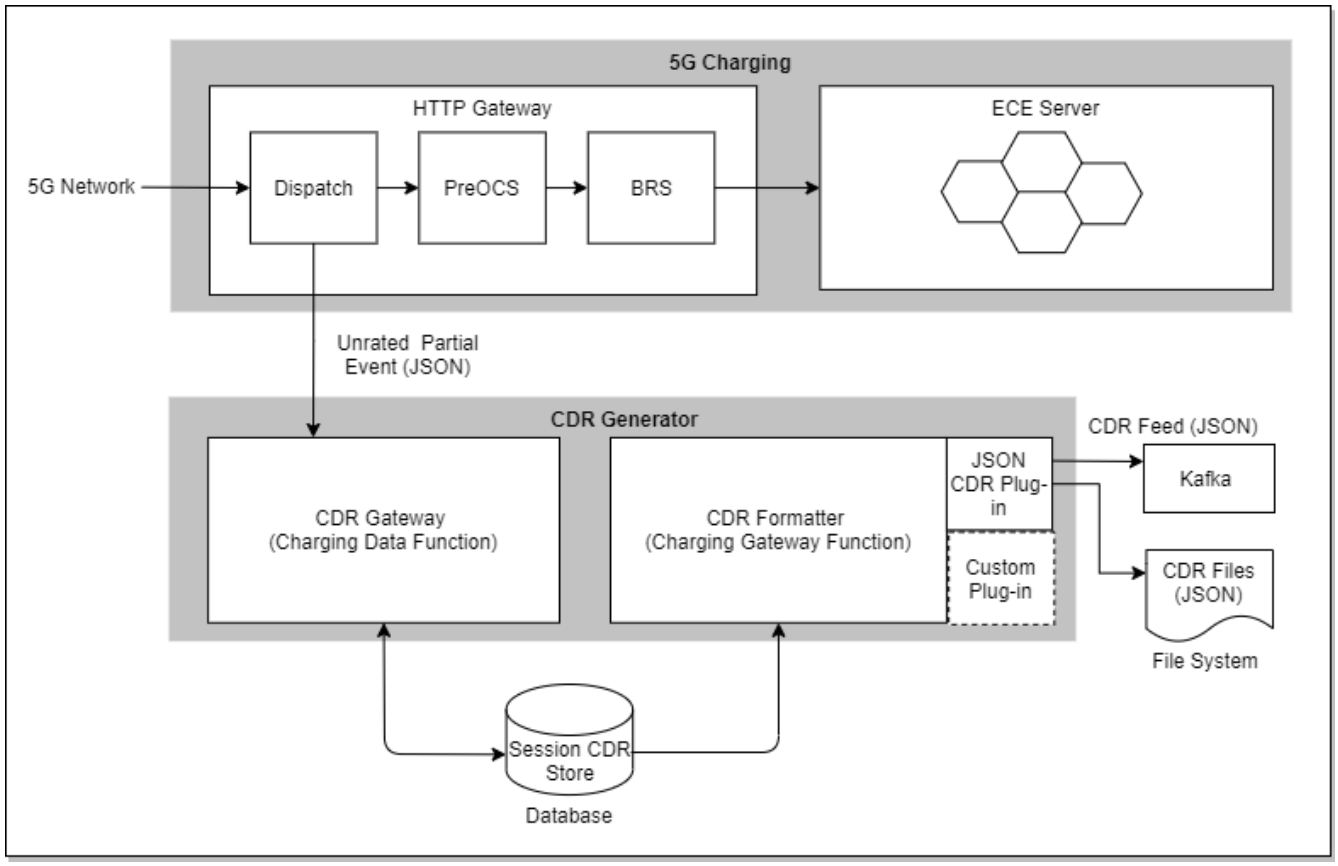
ECE generates CDR files in the Charging Function component (CHF) using this process:

1. The HTTP Gateway sends a request to the CDR Gateway. The requests can be for online charging, offline charging, or both.
2. The CDR Gateway does the following:
   a. Generates individual CDR records for each request or aggregates multiple requests into a CDR record according to the trigger criteria you specify.
   b. Stores CDR records in the CDR database. You can use either Oracle NoSQL Database or Oracle Database.
3. The CDR Formatter does the following:
   a. Extracts CDR records from the database and passes them to the CDR Formatter plug-in module for processing. You can use the default plug-in included with ECE or create a custom plug-in.
   b. Purges CDR records from the database after a specified amount of time. You can purge both processed and incomplete CDR records based on your configuration.

4. The CDR Formatter plug-in module generates CDR files. Depending on your configuration, it stores them on the disk or sends them to the Kafka messaging service.

Figure 19-1 shows the process flow for generating CDRs.

Figure 19-1 CDR Process Flow



Setting Up ECE to Generate CDRs

To set up ECE to generate CDRs:

1. Connect your 5G client to HTTP Gateway. See "Connecting ECE to a 5G Client".
2. Configure HTTP Gateway to route usage requests to the CDR Gateway. See "Configuring HTTP Gateway for CDR Generation".
3. Configure the CDR Gateway to generate CDRs and store them in the database. See "Configuring the CDR Gateway".
4. Configure the CDR Formatter to extract unrated CDRs from the database. See "Configuring the CDR Formatter".
5. Specify the plug-in to use for creating JSON formatted CDR files. See "Configuring the CDR Formatter Plug-in".

Accessing ECE Configuration MBeans

For all configurations, start by accessing the ECE configuration MBeans:

1. Log on to the driver machine.
2. Start the ECE charging servers (if they are not started).
3. Connect to the ECE charging server node enabled for JMX management.
   This is the charging server node set to **start CohMgt = true** in the *ECE_home*/**config/eceTopology.conf** file, where *ECE_home* is the directory in which ECE is installed.
4. Start a JMX editor that enables you to edit MBean attributes, such as JConsole.
5. In the editor's MBean hierarchy, find the ECE configuration MBeans.

Configuring HTTP Gateway for CDR Generation

You can configure HTTP Gateway to send usage requests to the CDR Gateway by enabling CDR generation.

You can also configure HTTP Gateway to route usage requests to ECE Server for rating or the CDR Gateway based on the type of charging request:

- **Offline Charging Requests**: To send offline charging requests to ECE Server, set the **rateOfflineCDRinRealtime** attribute to **true**. To send them to the CDR Gateway, set the attribute to **false**.
- **Online Charging Requests**: To send online charging requests to ECE Server, set the **generateCDRsForOnlineRequests** attribute to **false**. To send them to the CDR Gateway, set the attribute to **true**.

To configure the HTTP Gateway:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.httpGatewayConfigurations**.
4. Expand **Attributes**.
5. Specify values for the fields in Table 19-1.

Table 19-1 Fields for Configuring CDR Generation

Name	Default	Description
cdrGenerationEnabled	"false"	If set to **true**, ECE generates CDRs according to your configuration. If set to **false**, ECE uses convergent charging for all charging requests. If set to **false**, no other settings in this area are relevant.
cdrGenerationStandaloneMode	"false"	If set to **true**, HTTP Gateway doesn't send any requests to ECE, but it still generates CDRs.
cdrGatewayList	"localhost:8084"	Specifies one or more servers for the CDR Gateway. Use a comma-separated list for multiple servers.
cdrGatewayRetry	"3"	The number of attempts for sending requests to CDR Gateway before giving up.
rateOfflineCDRinRealtime	"false"	If set to **true**, HTTP Gateway sends offline charging requests to ECE for rating.
generateCDRsForOnlineRequests	"true"	If set to **true**, HTTP Gateway generates CDRs for all online charging requests.

Configuring the CDR Gateway

You configure the CDR Gateway to connect to your CDR storage database. You can also configure the CDR Gateway to generate either individual CDRs or aggregate multiple CDRs together according to trigger criteria that you specify.

To configure the CDR Gateway:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.cdrGatewayConfigurations**.
4. Expand **Attributes**.
5. Specify values for the fields in Table 19-2.

Table 19-2 Fields for Configuring CDR Gateway

Name	Default	Description
name	"cdrGateway1"	The name of this CDR Gateway instance. You can specify multiple CDR Gateways when configuring the HTTP Gateway.
primaryInstanceName	"cdrGateway1"	The name of the primary CDR Gateway instance.
schemaNumber	"1"	The number of the database schema used to store CDRs. Different CDR Gateways can write to different schemas.
isNoSQLConnection	"true"	The type of database used for storing CDRs: • **true**: Oracle NoSQL Database • **false**: Oracle Database
noSQLConnectionName	"@NO_SQL_CONNECTION@"	The connection name for the NoSQL database. This attribute applies only if you are using Oracle NoSQL Database for storing CDRs.
connectionName	"@ORACLE_PERSISTENCE_CONNECTION_NAME@"	The connection name of the Oracle database. This attribute applies only if you are using Oracle Database for storing CDRs.
cdrPort	"8084"	The port number for the CDR Gateway server.
cdrHost	"localhost"	The IP address, host name, or fully qualified domain name (FQDN) for the CDR Gateway server. **cdrHost** and **cdrPort** are also included in the **cdrGatewayList** field for the HTTP Gateway.
individualCdr	"false"	The type of CDR generation: • **true**: ECE generates an individual CDR for each event. • **false**: ECE aggregates requests until a trigger takes effect to write out the partial CDR or terminate the request. For more information, see "About Trigger Types".
cdrServerCorePoolSize	"32"	The number of threads in the CDR server pool.
cdrServerMaxPoolSize	"256"	The maximum number of threads allowed in the CDR server pool.

Table 19-2 (Cont.) Fields for Configuring CDR Gateway

Name	Default	Description
enableIncompleteCdrDetection	"false"	Whether to mark partially processed CDRs in the CDR Store as incomplete: • **true**: The CDR Formatter marks partially processed CDRs as incomplete. • **false**: Partially processed CDRs are not marked.
retransmissionDuplicateDetectionEnabled	"false"	Whether to detect and mark when CDRs contain duplicate usage updates. • **true**: The CDR Formatter marks when a CDR contains duplicate usage updates. • **false**: The CDR Formatter does not search for duplicate usage updates.

Configuring the CDR Formatter

You can configure the CDR Formatter to do the following:

- Retrieve completed CDRs from the CDR Store and pass them to a specified plug-in module for processing.
- Purge completed CDRs from the CDR Store that are older than a specified number of days.
- Purge orphan CDRs from the CDR Store. Orphan CDRs are incomplete CDRs that are older than a specified number of seconds. The CDR Gateway can create orphan CDRs if your ECE system goes down due to maintenance or failure.

  For example, if it is 12:00:00 and the configurable duration is 200 seconds, the CDR Formatter would purge from the CDR Store all incomplete CDRs that were last updated today at 11:56:40 or earlier.
- Mark partially processed CDRs as incomplete. This can occur in active-active disaster recovery systems when a site goes down while processing CDR sessions. See "About CDR Generator in an Active-Active System" in *BRM System Administrator's Guide* for more information.

To configure the CDR Formatter:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **cdrFormatters**.
4. Expand **Attributes**.
5. Specify values for the fields in Table 19-3.

Table 19-3 Fields for Configuring the CDR Formatter

Name	Default	Description
name	"cdrFormatter1"	The name of a CDR Formatter instance. You should name CDR Formatter instances consistently and uniquely (for example, cdrFormatter1, cdrFormatter2, and so on).
primaryInstanceName	"cdrFormatter1"	The name of the primary CDR Formatter instance.

Table 19-3 (Cont.) Fields for Configuring the CDR Formatter

Name	Default	Description
schemaNumber	"1"	The number of the database schema for processing CDRs.
isNoSQLConnection	"true"	The type of database used for storing CDRs: • **true**: Oracle NoSQL Database • **false**: Oracle Database
noSQLConnectionName	"@NO_SQL_CONNECTION@"	The connection name for the NoSQL database. This attribute applies only if you are using Oracle NoSQL Database for storing CDRs. This is the same connection you use for CDR Gateway.
connectionName	"@ORACLE_PERSISTENCE_CONNECTION_NAME@"	The connection name of the Oracle database. This attribute applies only if you are using Oracle Database for storing CDRs. This is the same connection you use for CDR Gateway.
threadPoolSize	"6"	The number of threads used by the CDR Formatter instance to process a set of CDRs 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.
retainDuration	"0"	The duration in seconds that processed CDRs are retained in the CDR Store before they can be purged. The default is **0**, which means that CDRs are purged immediately after being processed.
ripeDuration	"60"	The duration in seconds that CDRs must be stored in the CDR Store before the CDR Formatter can read them. Delaying the processing of CDRs up to the **ripeDuration** time allows time for resolving any duplicate CDRs that may have been persisted to the CDR Store.
checkPointInterval	"6"	The time interval in seconds that the CDR Formatter instance waits before reading a batch of CDR information. This value must be: • Less than or equal to the value of **ripeDuration** • Evenly divisible by the number of threads configured for **threadPoolSize** The CDR Formatter doesn't read CDR information when the time interval is less than the **ripeDuration** interval.
pluginPath	"ece-cdrformatter.jar"	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.
pluginType	"oracle.communication.brm.charging.cdr.formatterplugin.internal.SampleCdrFormatterCustomPlugin"	The type of plug-in used to format CDRs.
pluginName	"cdrFormatterPlugin1"	The class name with the package path for the formatter plug-in object to be called by the CDR formatter. You can write a custom plug-in and specify it here.
noSQLBatchSize	"25"	The number of CDR records to be read from the NoSQL Database in a single read operation.
cdrStoreFetchSize	"2500"	The number of CDR records to retrieve from the CDR Store and hold in memory at a time.

Table 19-3 (Cont.) Fields for Configuring the CDR Formatter

Name	Default	Description
cdrOrphanRecordCleanupAgeInSec	"200"	The amount of time, in seconds, at which an incomplete record in the CDR Store is considered an orphan. The CDR Formatter purges orphan records from the CDR Store that are this age or older.
cdrOrphanRecordCleanupSleepIntervalInSec	"200"	The sleep interval, in seconds, between each call to purge orphan CDRs from the CDR Store.
enableIncompleteCdrDetection	"false"	Whether to mark partially processed CDRs in the database as incomplete: • **true**: The CDR Formatter marks partially processed CDRs in the database as incomplete. • **false**: Partially processed CDRs are not marked.

Configuring the CDR Formatter Plug-in

You can configure the CDR Formatter plug-in to create JSON-formatted CDR files and then store them in your file system or send them to your Kafka messaging service.

To configure the CDR Formatter plug-in:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **cdrFormatterPlugins**.
4. Expand **Attributes**.
5. Specify values for the fields in Table 19-4.

Table 19-4 Fields for Configuring the CDR Formatter Plug-in

Name	Default	Description
name	"cdrFormatterPlugin1"	Name of the formatting plug-in. **cdrFormatterPlugin1** is the default plug-in that comes with ECE, but you can also specify a custom plug-in.
tempDirectoryPath	"/tmp/tmp"	Path for CDR files being formatted.
doneDirectoryPath	"/tmp/done"	Path for formatted CDR files.
doneFileExtension	".out"	Extension for CDR files.
enableKafkaIntegration	"false"	Whether to send records to Kafka. Because ECE uses Kafka for notifications, you set it up during installation and configure the HTTP Gateway to use it. See "Creating Kafka Topics for ECE" in *ECE Installation Guide* and "Connecting ECE to Kafka Topics".
enableDiskPersistence	"true"	Whether to save records as JSON files in the path specified for **doneDirectoryPath**.
maxCdrCount	"20000"	The maximum number of CDRs that can be written to a CDR output file.

Table 19-4 (Cont.) Fields for Configuring the CDR Formatter Plug-in

Name	Default	Description
enableStaleSessionCleanupCustomField	"false"	Whether to allow the plug-in to add a custom reason value to a CDR's **causeForRecordClosing** field. This field specifies why a CDR session was closed. • **true**: The custom value is allowed. • **false**: The custom value is not supported.
staleSessionCauseForRecordClosingString	"PARTIAL_RECORD"	The name of the custom reason for closing a CDR session. This value will be added to the CDR's **causeForRecordClosing** field.

About Trigger Types

If you configured ECE to aggregate events in the same CDR, ECE uses triggers to determine when a CDR remains open and when it closes. See:

- Triggers for Convergent Charging Events
- Triggers for Roaming Events

Triggers for Convergent Charging Events

When an event meets any of the trigger conditions in Table 19-5, the event is added to the open CDR. The CDR remains open with the same sequence number.

> **Note:**
>
> If the request type is TERMINATE, the CDR is closed regardless of the trigger type.

Table 19-5 Triggers for Adding to an Open CDR

Category	Trigger Type
Change of Charging Conditions	• QOS_CHANGE • USER_LOCATION_CHANGE • SERVING_NODE_CHANGE • CHANGE_OF_UE_PRESENCE_IN_PRESENCE_REPORTING_AREA • CHANGE_OF_3GPP_PS_DATA_OFF_STATUS • HANDOVER_CANCEL • HANDOVER_START • HANDOVER_COMPLETE
Limit per Rating Group	• TIME_LIMIT (The expiration of the data time limit per rating group) • VOLUME_LIMIT (The expiration of the data volume limit per rating group) • EVENT_LIMIT (The expiration of the data event limit per rating group)
Quota Management Triggers	• QUOTA_THRESHOLD (The validity time, volume, or unit threshold has been reached) • QUOTA_EXHAUSTED (The validity expiration time, volume, or unit quota is exhausted) • FORCED_REAUTHORISATION (Reauthorization request by CHF)

When an event meets any of the trigger conditions in Table 19-6, it is added to the open CDR and then the CDR is closed.

Table 19-6 Triggers for Closing a CDR

Category	Trigger Type
Change of Charging Conditions	• UE_TIMEZONE_CHANGE • PLMN_CHANGE PLMN • RAT_CHANGE RAT • SESSION_AMBR_CHANGE • REMOVAL_OF_UPF • MANAGEMENT_INTERVENTION • ADDITION_OF_ACCESS **Note:** If the Change of Charging condition is true for any trigger that comes at the rating group level, its information is added into a new usedUnitContainer block. It is not aggregated into the existing usedUnitContainer block.
Limit per PDU Session	• TIME_LIMIT (The expiration of data time limit per PDU session) • VOLUME_LIMIT (The expiration of data volume limit per PDU session) • EVENT_LIMIT (The expiration of data event limit per PDU session) • MAX_NUMBER_OF_CHANGES_IN_CHARGING_CONDITIONS (The maximum number of charging condition changes has expired)

The following sample CDR shows a trigger type of **MANAGEMENT_INTERVENTION**, which causes ECE to add the information to the open CDR and then close the CDR:

```
"quotaManagementIndicator": "ONLINE_CHARGING",
"triggers": [
   {
      "triggerType": "MANAGEMENT_INTERVENTION",
      "triggerCategory": "IMMEDIATE_REPORT"
   }
],
```

Triggers for Roaming Events

When a roaming event meets any of the trigger conditions in Table 19-7, it is added to the open CDR.

Table 19-7 Triggers for Adding to an Open CDR (Roaming Event)

Category	Trigger Type
Change of Charging Conditions	• QOS_CHANGE • USER_LOCATION_CHANGE • SERVING_NODE_CHANGE • CHANGE_OF_UE_PRESENCE_IN_PRESENCE_REPORTING_AREA • CHANGE_OF_3GPP_PS_DATA_OFF_STATUS • TARIFF_TIME_CHANGE
Limit per QoS Flow	• TIME_LIMIT (Expiration of data time limit per QoS Flow) • VOLUME_LIMIT (Expiration of data volume limit per QoS Flow)
Others	• TIME_LIMIT (Expiration of data time limit per QoS Flow) • VOLUME_LIMIT (Expiration of data volume limit per QoS Flow)

When a roaming event meets any of the trigger conditions in Table 19-8, it is added to the open CDR, and the CDR is then closed.

Table 19-8 Triggers for Closing a CDR (Roaming Event)

Category	Trigger
Change of Charging Conditions	• UE_TIMEZONE_CHANGE • PLMN_CHANGE • RAT_CHANGE • SESSION_AMBR_CHANGE • REMOVAL_OF_UPF • MANAGEMENT_INTERVENTION
Limit per PDU Session	• TIME_LIMIT (The expiration of data time limit per PDU session) • VOLUME_LIMIT (The expiration of data volume limit per PDU session) • EVENT_LIMIT (The expiration of data event limit per PDU session) • MAX_NUMBER_OF_CHANGES_IN_CHARGING_CONDITIONS (The maximum number of charging condition changes has expired)

The following sample CDR for a roaming event shows a trigger type of **QOS_CHANGE**, which causes ECE to add the information to the open CDR and keep the CDR open:

```
"quotaManagementIndicator": "ONLINE_CHARGING",
"triggers": [
   {
      "triggerType": "QOS_CHANGE",
      "triggerCategory": "IMMEDIATE_REPORT"
   }
],
```

20

Connecting ECE to a Diameter Client

You can set up network integration for online charging by using the Oracle Communications Elastic Charging Engine (ECE) Diameter Gateway.

Topics in this document:

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 notifications from the ECE Notification queue or topic, create Diameter notification messages from them, and send the notification messages to the appropriate Diameter clients.

For information about conformance with industry standards, see *ECE Diameter Gateway Protocol Implementation Conformance Statement*.

To implement Diameter Gateway:

1. When you install ECE, do this:
    - Add Diameter Gateway node instances required for your topology and configure each instance.
    - If you are using Oracle WebLogic for notification handling, specify to create WebLogic queues and enter the details for your ECE Notification queue and Suspense queue.

- If you are using Apache Kafka for notification handling, specify to create Kafka topics and enter the details for your ECE Notification topic, Suspense topic, ECE failure topic, and ECE overage topic.

> **Note:**
>
> Systems that support 5G networks must use Apache Kafka for notification handling.

For more information, see "Installing Elastic Charging Engine" in *Elastic Charging Engine Installation Guide*.

2. During the ECE postinstallation process, do this:
   - If you are using Oracle WebLogic for notification handling, run the **post_Install.pl** script to create your ECE Notification queue and Suspense queue. See "Creating WebLogic JMS Queues for BRM" in *Elastic Charging Engine Installation Guide*.
   - If you are using Apache Kafka for notification handling, run the **kafka_post_install.sh** script to create your ECE Notification topic and Suspense topic. Then, run the **post_Install.sh** script and choose to create only the Acknowledgment queue. See "Creating Kafka Topics for ECE" and "Creating WebLogic JMS Queues for BRM" in *Elastic Charging Engine Installation Guide*.
3. For all of the 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:
   - Defined any custom service/event mappings in PDC.
   - 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. You can set up Diameter Gateway to consume notifications from either:
   - Oracle WebLogic queues. To use WebLogic queues, see "Configuring WebLogic Queues for Notifications".
   - Apache Kafka topics. To use Kafka topics, see "Configuring ECE for Apache Kafka".
6. (Optional) Configure ECE to send information about failed usage requests to the ECE failure topic. See "Recording Failed ECE Usage Requests".
7. (Optional) Configure ECE to generate CDRs for any prepaid usage overage and send them to the ECE overage topic. See "Configuring ECE to Support Prepaid Usage Overage".
8. 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 start up, the Diameter Gateway instances read from the ECE Notification queue or topic for notifications.

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 retrieves Sp and Sy information from ECE charging servers and sends the information to the Policy and Charging Rule 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 associated with charge offers that have an active policy session, ECE charging servers publish asynchronous notifications to the JMS notification queue. Diameter Gateway receives the policy notifications at startup and processes them as follows:

- From spending-limit JMS notifications, Diameter Gateway creates Sy (Spending-Status-Notification-Request (SNR)) messages for all subscribed sessions and routes them to the appropriate Diameter clients.
- From subscriber-preferences JMS notifications, 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 uses the ECE policy management APIs to retrieve Sy-interface and Sp-interface data from the ECE server, see "Configuring Policy-Driven Charging".

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

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

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 inquiry
- 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 one service. Contain the following Gy interface request types in an MSCC group:

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

  For more information about MSCC requests and ECE, see "Configuring Multiple Services Credit Control".
- 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 "How Diameter Gateway Creates 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".

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 RAR messages and sends them to Diameter clients running the applicable active Gy sessions.

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

How Diameter Gateway Creates Usage Requests

Diameter Gateway creates usage requests based on the event definitions sent from PDC to ECE. Diameter Gateway includes a usage-request builder for creating usage requests (as well as different builders for building other requests ECE supports, such as balance query requests, top-up requests, and so on). When you start Diameter Gateway nodes, the usage-request builder reads the event definition data and sends requests that adhere to the specifications. See "About Usage Request Fixed Attributes" for more information on the attributes.

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).

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).

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 determine which service and event combination applies to an incoming Diameter request, enabling it to select the event definition that applies to the event to be rated.

You configure Diameter Gateway to base its selection of event definitions on any combination of the following AVPs in the request:

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

From the preceding AVP values, Diameter Gateway derives the following fields, which uniquely identify the event definition to use for building the ECE request:

- ProductType (service)
- EventType
- Version

You can configure Diameter Gateway to base its event definition on a custom AVP by using the Diameter Gateway Request-Received extension. You use that extension to modify one of the

AVP values in the request so that a different Diameter mediation mapping is produced for a service, event, and version.

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*/**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 service you have defined for the event.

   - EventType

     The event definition you have defined for the event.

   - Version

     The version number of the event definition object that you want to apply to the event.

   Define the Service-Identifier, Rating-Group, and Service-Context-Id for each event definition object 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, service, and event).

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 event definition object.

   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***/config/management/migration-configuration.xml** file is the directory where you save your configuration data.
6. Run 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 into the repository, Diameter Gateway recreates 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.
8. Load the mediation specification file into the ECE server by using the **configLoader** utility.

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 data types for custom balance-query AVP fields, see the *ECE_home***/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
- Return that the top-up succeeded or failed
- Return 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 data types for custom top-up-request AVP fields, see the *ECE_home***/config/diameter/dictionary_main.xml** file.

Sending Multiple-Service Credit Control (MSCC) Requests from Diameter Gateway

Diameter Gateway supports MSCC requests in which a Diameter application performs credit control for multiple services within the same session.

Diameter Gateway only supports Multiple-Service Credit Control (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 system. For example, you might have separate online charging systems for handling different subscription services. You can 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 system.

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, and 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**.

   The file is in the directory specified by the **configObjectsDataDirectory** parameter in the *ECE_home***/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=DEBUG
  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***/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. You also put a path to your AVP field to an MSCC group block.

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 the active session duration based on the units 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 creating your usage request builder, specify the accounting behavior 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.

> **Note:**
>
> 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 the attribute INPUT_VOLUME on the usage request, the RUMs must be defined with the same name.

Configuring Accounting Mode for Diameter Gateway

You can configure Diameter Gateway to use both incremental-based and cumulative-based accounting logic when processing usage requests. You can perform this by specifying the accounting mode in the mediation specification file. The accounting mode indicates to Diameter Gateway whether to use incremental-based or cumulative-based accounting logic.

To configure the accounting mode for Diameter Gateway:

1. Open the Diameter mediation specification file, **diameter_mediation.spec**.

   For the location of the **diameter_mediation.spec** file, see the **configObjectsDataDirectory** parameter in the *ECE_home***/config/management/migration-configuration.xml** file.

2. For each event to be rated (in each row), specify the accounting mode in the **UnitReportingMode** column. Valid values are:

   - **INCREMENTAL**
   - **CUMULATIVE**

   For example:

   ```
   Service-Context-Id       | Service-Identifier | Rating-Group | ProductType |
   EventType | Version | Subscription-Id-Type | ValidFrom                |
   UnitReportingMode |
   "gy.service@example.com" | "1"                | "10"         | "VOICE"     |
   "V_USAGE" | 1.0     | ""                   | "2024-3-31T12:01:01 PST" |
   "INCREMENTAL"     |
   "gy.service@example.com" | "1"                | "11"         | "DATA"      |
   "D_USAGE" | 1.0     | ""                   | "2024-3-31T12:01:01 PST" |
   "CUMULATIVE"      |
   ```

   The default accounting mode is **Incremental**. If you specify null or if you do not specify a mode in the **UnitReportingMode** column, Diameter Gateway uses the default accounting mode when processing usage requests. This supports backward compatibility.

   Your accounting mode configuration is applicable for the combination of the given values specified in the row for the event to be rated. You can also configure different accounting modes for the same product and event type combination.

3. Save and close the file.

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

   ```
   start configLoader
   ```

   When the 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.

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

   ```
   ./ecc
   ```

7. Do one of the following:
   - If the Diameter Gateway instance is *not* running, start it.

     The instance reads its configuration information by name at startup.
   - If the Diameter Gateway instance is running, stop and restart it.

   For information about stopping and starting Diameter Gateway instances, see "Starting and Stopping ECE" in *BRM System Administrator's Guide*.

Configuring WebLogic Queues for Notifications

To configure Diameter Gateway to listen for notifications from ECE, you must specify the types of notifications that ECE generates.

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".

To enable Diameter Gateway to consume notifications from an ECE notification queue set up in WebLogic:

> **Note:**
>
> The following steps assume that ECE is installed and that the required ECE postinstallation tasks have been completed.

1. On the Oracle WebLogic server, verify that the ECE notification queue (a JMS topic) was created.
2. In ECE, verify that JMS credentials were configured correctly so that ECE can publish notifications to the ECE notification queue.

   See "About ECE Notifications" for information.
3. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
4. Expand the **ECE Configuration** node.
5. Expand **charging.notification**.
6. Expand **Attributes**.

7. Set the appropriate type of notification (such as top-up or advice of charge) to the appropriate value. See "About ECE Notifications" for information.

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 configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.diameterGatewayPeerConfigurations**.
4. Expand **Attributes**.
5. For each peer connected to the Diameter Gateway, configure alternative peers by specifying values for the following attributes:
   - **peerName**: Enter the name of the Diameter peer.
   - **alternatePeerName**: Enter the name of the alternative peer for the specified Diameter peer. You can specify two alternative peers for each Diameter peer.
6. Go to the *ECE_home*/**bin** directory.
7. Start the Elastic Charging Controller:

   ```
   ./ecc
   ```

8. Do one of the following:
   - If the Diameter Gateway instance is *not* running, start it.
   - If the Diameter Gateway instance is running, stop and restart it.

Viewing Active Diameter Peers

To view all the active diameter peers:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **DiameterGateway** node.
3. Expand **PeerConnectionsTracker**.
4. Expand **Attributes**.
5. Click the **peerConnections** value.

   The diameter peers that are active (which are currently connected to the specific Diameter Gateway instance) are displayed.

Configuring ECE for Apache Kafka

You can connect ECE to the following ECE Kafka topics so that ECE can publish notifications, failed usage requests, and CDRs with usage overage information to them:

- **ECE notification topic**: Stores notifications from ECE.
- **Suspense topic**: Stores failed notifications from ECE.
- **ECE failure topic**: Stores details about failed ECE usage requests, such as the user ID and request payload. See "Recording Failed ECE Usage Requests" for more information.
- **ECE overage topic**: Stores overage records, which contain details about usage overage amounts for prepaid customers. See "Configuring ECE to Support Prepaid Usage Overage".

To configure ECE to work with Apache Kafka:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Enable ECE to send notifications to Kafka topics:
   a. Expand **charging.server**.
   b. Expand **Attributes**.
   c. Set the **kafkaEnabledForNotifications** property to **true**.
4. Connect ECE to your Kafka Server and Kafka topics:
   a. Expand **charging.kafkaConfigurations**.
   b. Expand **Attributes**.
   c. Specify the Kafka configuration values for the attributes in Table 20-1.

Table 20-1 kafkaConfiguration Properties

Property Name	Description
name	The name of your ECE cluster.
hostname	The host name and port number of the machine in which Apache Kafka is installed. If it contains multiple Kafka brokers, create a comma-separated list.
topicName	The name of the Kafka topic where ECE will publish notifications.
suspenseTopicName	The name of the Kafka topic where failed notifications are published.
failureTopicName	The name of the Kafka topic where ECE will publish details about failed usage requests. See "Recording Failed ECE Usage Requests".
overageTopicName	The name of the Kafka topic where ECE will publish overage records with information about your prepaid customer's usage overage during online sessions. See "Configuring ECE to Support Prepaid Usage Overage".

Table 20-1 (Cont.) kafkaConfiguration Properties

Property Name	Description
partitions	The number of Kafka partitions in your topics. The recommended number to create is calculated as follows: [(Max HTTP Gateway Nodes) + (Max Diameter Gateway Nodes * Max Diameter Clients) + (1 for BRM Gateway) + (1 for Internal Notifications)] For example, if you have 2 HTTP Gateway nodes, 4 Diameter Gateway nodes, 10 Diameter Gateway clients, and a BRM Gateway, you would need [(2 + (4 * 10) + 1 + 1) = 44 Kafka partitions. Arbitrarily, you can set this to a maximum value.
failurePartitions	The number of Kafka partitions in your ECE failure topic.
kafkaProducerReconnectionInterval	The amount of time, in milliseconds, the Notification Publisher waits before attempting to reconnect to the Kafka topic.
kafkaProducerReconnectionMax	The maximum amount of time, in milliseconds, the Notification Publisher waits before attempting to reconnect to a broker that has repeatedly failed to connect. The **kafkaProducerReconnectionInterval** will increase exponentially for each consecutive connection failure up to this maximum.
kafkaDGWReconnectionInterval	The amount of time, in milliseconds, Diameter Gateway waits before attempting to reconnect to the Kafka topic.
kafkaDGWReconnectionMax	The maximum amount of time, in milliseconds, Diameter Gateway waits before attempting to reconnect to a broker that has repeatedly failed to connect. The **kafkaDGWReconnectionInterval** will increase exponentially for each consecutive connection failure up to this maximum.
kafkaBRMReconnectionInterval	The amount of time, in milliseconds, BRM Gateway waits before attempting to reconnect to the Kafka topic.
kafkaBRMReconnectionMax	The maximum amount of time, in milliseconds, BRM Gateway waits before attempting to reconnect to a broker that has repeatedly failed to connect. The **kafkaBRMReconnectionInterval** will increase exponentially for each consecutive connection failure up to this maximum.
kafkaHTTPReconnectionInterval	The amount of time, in milliseconds, HTTP Gateway waits before attempting to reconnect to the Kafka topic.

Table 20-1 (Cont.) kafkaConfiguration Properties

Property Name	Description
kafkaHTTPReconnectionMax	The maximum amount of time, in milliseconds, HTTP Gateway waits before attempting to reconnect to a broker that has repeatedly failed to connect. The **kafkaHTTPReconnectionInterval** will increase exponentially for each consecutive connection failure up to this maximum.

5. Configure ECE to generate your desired notification types:
   a. Expand **charging.notification**.
   b. Expand **Attributes**.
   c. Specify which type of notifications to send to your ECE Notification topic. For example, to send asynchronous notifications when an ongoing session requires a reauthorization request, set **rarNotificationMode** to **ASYNCHRONOUS**. See "Enabling Specific Notification Types".

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.

Recording Failed ECE Usage Requests

ECE may occasionally fail to process usage requests. For example, a data usage request could fail because a customer has insufficient funds. You can configure ECE to publish details about failed usage requests, such as the user ID and request payload, to the ECE failure topic in your Kafka server. Later on, you can reprocess the usage requests or view the failure details for analysis and reporting.

To enable the recording of failed ECE usage requests:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.kafkaConfigurations**.
4. Expand **Attributes**.
5. Set the **persistFailedRequestsToKafkaTopic** property to **true**.

Including Loan Sub-Balance in Balance Queries

By default, Diameter Gateway includes information about the main currency balance in balance query responses. You can configure Diameter Gateway to include both the main and

loan balance information in balance query responses. To configure it to do so, in the ECE Balance Java API, set the **balancequerymode** to **TURBO**. For example:

```
ORA-Balance-Query-Mode (248,VM,v=3512,I=16) = TURBO_MODE (4)
```

For more information about **balancequerymode**, see the documentation for **oracle.communication.brm.charging.messages.query** in *Elastic Charging Engine Java API Reference*.

21

Connecting ECE to a RADIUS Client

You can use Oracle Communications Elastic Charging Engine (ECE) RADIUS Gateway for authenticating access requests, processing accounting requests from RADIUS clients, such as terminal servers or network access servers (NAS), and processing disconnect requests from the RADIUS server to RADIUS clients.

For information about conformance with industry standards, see *ECE RADIUS Gateway Protocol Implementation Conformance Statement*.

Topics in this document:

Overview of Authentication and Accounting Using RADIUS Gateway

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.

2. (Optional) Customize the RADIUS data dictionary to include custom vendor-specific attributes.
3. Load your event definitions from PDC into ECE.
4. Load customer data and data keys from BRM into ECE.
5. Load the RADIUS mediation specification data.
6. Map 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.

About RADIUS Gateway Authentication

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.
- **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.

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
```

Loading Data Keys Extracted from BRM into ECE

As part of the initial load of customer data into ECE, Customer Updater loads data keys into ECE. When RADIUS Gateway is started, the data keys are decrypted using the BRM root key in the Oracle wallet file and 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.

When you add or modify a data key in BRM, you must load the newly added or modified data key extracted from BRM into ECE.

To load data keys extracted from BRM into ECE:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **UpdateEventhandler** node.
3. Expand **Update**.
4. Expand **Operations**.
5. Select **updateDataKeys**.
6. Click the **updateDataKeys** button.

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

Customizing the RADIUS Data Dictionary

This section covers customizing the RADIUS data dictionary.

About the RADIUS Data Dictionary

The data dictionary includes a list of AVPs that are used by RADIUS Gateway to perform authentication and accounting operations. The RADIUS data dictionary contains the standard AVPs that are prescribed in RADIUS Request for Comments (RFC) 2865, 2866, and 2869, and also some sample vendor-specific attributes. You can use the sample vendor-specific attributes as a template for adding custom vendor-specific attributes. The default location of the RADIUS data dictionary file is *ECE_home***/config/radius/radiusDictionary.xml**.

> **Note:**
>
> Do not remove, rename, or move the RADIUS data dictionary file to a different location.

Creating a Custom Data Dictionary

You can create a custom data dictionary file by using the *ECE_home***/config/radius/radiusDictionary.xml** file as a template. The default location for your custom data dictionary file is *ECE_home***/config/radius/custom/***dictionary_file*, where *dictionary_file* is the name of your custom data dictionary file. You can add new vendor-specific attributes to your custom data dictionary file. See "Adding Custom Vendor-Specific Attributes".

Selecting a RADIUS Data Dictionary When Using Different NAS Vendors

If you must use NAS servers from multiple vendors, you have the following options:

- If your NAS is RFC 2865 compliant, you can use the RFC2865 data dictionary. This is the preferred solution. Update the dictionary file with any vendor-specific attributes associated with the NAS.
- If your NAS is not RFC 2865 compliant, you can use the RADIUS data dictionary files for adding vendor-specific attributes. See "Adding Custom Vendor-Specific Attributes" for more information.

Adding Custom Vendor-Specific Attributes

In special cases, where you are using NAS servers from multiple vendors, you must add the vendor attribute and code in your custom data dictionary file.

The syntax for adding a vendor-specific attribute is:

```
<?xml version="1.0" encoding="UTF-8"?>
    <dictionary schemaLocation= "radiusDictionary.xsd"
       <vendor value="vendor_ID"name="vendor_name"/>
       </attribute name="attribute_name" vendor="vendor_name" syntax="data_type"
code="attribute_ID"/>
    </dictionary>
```

Table 21-1 lists the vendor-specific attribute values and descriptions.

Table 21-1 Vendor-Specific Attribute Values

Parameters	Description
vendor_ID	Number used to identify the NAS or gateway vendor. These numbers are assigned by the Internet Advisory Board (IAB). See your vendor's documentation for details. Some common vendor identification numbers are: • 9 (Cisco) • 10415 (3GPP) • 2636 (Juniper)
vendor_name	Name of the vendor.
attribute_name	Name of the attribute. This must be unique. **Important:** Do not use the same attribute name as used in the default RADIUS data dictionary file. Using the same attribute name in the custom data dictionary file overrides the attribute values in the default RADIUS data dictionary file.
attribute_ID	Identification number assigned to the attribute in the dictionary.

Table 21-1 (Cont.) Vendor-Specific Attribute Values

Parameters	Description
data_type	Any one of the following data types: • **UnsignedInt** 32-bit unsigned value in big endian order (high byte first). • **Integer** 32-bit value in big endian order (high octet first). • **String** 0-253 octets • **Ipaddr** 4 octets in network octet order • **Binary** 0-254 octets • **Password** (n * 16) (>= 16) octets. This field is encrypted according to the User-Password AVP in RFC 2865. • **Short** 16-bit value • **Octet** 8-bit value • **ifid** IPv6 interface ID • **ipv6addr** IPv6 address • **date** Linux timestamp in seconds (since January 1, 1970 GMT)

Loading the RADIUS Mediation Specification Data

RADIUS Gateway uses the RADIUS mediation specification data to determine which product and event type combination and network mapping applies to an incoming request from the RADIUS client.

To load the RADIUS mediation specification data:

1. Create a mediation specification file or open the sample RADIUS mediation specification file.

   A sample mediation specification file (*ECE_home***/sample_data/config_data/specifications/ece_simple**) is available.

   > **Note:**
   >
   > Create only one RADIUS mediation specification file to represent the mediation specification for RADIUS Gateway.

2. Load the pricing data from PDC into ECE.

   For every event definition, which contains charging operation types (for example, **Initiate**) loaded into ECE from PDC, ECE generates network mapping files.

3. Add a row (in the table) for each new product to be rated that specifies the following information:

   - **Service-Identifier AVP**

     A unique identifier of the service. The Service-Identifier AVP value is sent by the RADIUS request. “null" is valid if the field is not expected to be present in the request.

   - **ProductType**

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

   - **EventType**

     The event type that 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.

   - **ValidFrom**

     A future date and time when you want RADIUS Gateway to recognize a newly deployed request specification.

     To have requests processed according to a new specification, you would enter:

     yyyy-*mm*-*dd*T*hh*:*mm*:*ss* [*timezone*]

     If *timezone* is not specified, it defaults to **UTC**.

   - **Network-Mapping-FileName**

     The name of the network mapping file generated for the product and event combination.

4. Open the *ECE_home*/**config/management/migration-configuration.xml** file.
5. Search the **configObjectsDataDirectory** parameter and copy the value. For example:

   ```
   configObjectsDataDirectory = ECE_home/sample_data/config_data
   ```

6. Save the mediation specification file to that same directory.
7. Load the file into the ECE server by running the following command:

   ```
   start configLoader
   ```

   The utility loads the RADIUS mediation specification data to the ECE cluster. The **configLoader** utility uses the location in the **configdata** parameter for loading the data. As mediation specification files have same names, so any existing RADIUS mediation specification data in the ECE cluster is overwritten.

Example 21-1 Sample RADIUS Mediation Specification Entry

```
RadiusMediationTable {
Service-Identifier| ProductType | EventType | Version |  ValidFrom |  Network-Mapping-
FileName|
 "1" | "TelcoGprs" | "EventDelayedSessionTelcoGprs" | 2.0 | "2010-12-31T12:01:01 PST" |
"EventDelayedSessionTelcoGprs_TelcoGprs.xml" |
}
```

When you load the RADIUS mediation specification data into the ECE cluster, RADIUS Gateway re-creates its in-memory usage-request builder map and uses the mapping definitions to send requests to ECE.

About Mapping RADIUS Network Attributes to Event Attributes

To process requests from RADIUS clients, you map network attributes from RADIUS clients to the corresponding event attributes in ECE. You do this by editing the network mapping file. When you load the pricing data from PDC into ECE, ECE generates the network mapping file for each product and event combination. Some default network mappings are already pre-configured in the files generated by ECE. You can update the default values in these files.

RADIUS Gateway uses this mapping in ECE to process requests by dynamically mapping the values of the network attributes in the RADIUS request to the corresponding event attributes in ECE.

Mapping RADIUS Network Attributes to Event Attributes

If you add or remove an event attribute from the event definition in PDC, you have to add or remove the corresponding network attributes in ECE. You do this by editing the network mapping file in ECE.

Before you map the attributes, load the RADIUS mediation specification file. See "Loading the RADIUS Mediation Specification Data" for more information.

To map network attributes to event attributes:

1. Load the pricing data from PDC into ECE.

   Mapping files will be automatically generated when the pricing data is published from PDC to ECE.

   For every event definition, which contains charging operation types (for example, **Initiate**) loaded into ECE from PDC, ECE generates the network mapping files. The network mapping files are stored in the directory specified by the **configObjectsDataDirectory** parameter in the *ECE_home***/config/management/migration-configuration.xml** file

   A sample network mapping file is available in the (*ECE_home***/sample_data/config_data/specifications/ece_end2end/network_mapping**) directory. You can use this as a reference for mapping the attributes.
2. Open a network mapping file in a text editor.
3. Ensure that the ORIGIN_NETWORK event attribute is added as a top-level attribute in the network mapping file.
4. Map the network attributes to the event attributes by doing the following:
   a. Search for the event attribute that you want to map to the network attribute.
   b. Add the following entry:

      <networkField>*NetworkAttribute***</networkField>**

      where *NetworkAttribute* is the attribute of the requests received from RADIUS clients.

      For example:

      ```
      <attributeMapping type="RadiusMediationEntries">
           <attribute>
             <name>TERMINATE_CAUSE</name>
             <networkField>Acct-Terminate-Cause</networkField>
           </attribute>
         </attributeMapping>
      ```
5. Save and close the file.

> **Note:**
>
> Verify that the name of this network mapping file is specified in the RADIUS mediation specification file.

6. Load the network mapping data by doing one of the following:
   - If RADIUS Gateway is running, run the following command:

     ```
     start configLoader loadNetworkMapping
     ```

   - If RADIUS Gateway is *not* running, run the following commands:

     ```
     start customerUpdater
     start radiusGateway
     ```

     The network mapping data is loaded into the ECE cluster. Any existing network mapping data available for the product and event specification in the ECE cluster is overwritten. ECE is now in a usage-processing state, where it can accept requests from RADIUS Gateway.

When you load the network mapping into the ECE cluster, RADIUS Gateway re-creates its in-memory usage-request builder map and begins using the latest mapping definitions to send requests to ECE.

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 service in ECE by using the **avpName** and **vendorId** parameters.

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 service in ECE by using the **avpName** and **vendorId** parameters.

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 indicates that it is 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 service in ECE by using the **avpName** and **vendorId** parameters.

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]
```

About RADIUS Gateway Disconnection

RADIUS Gateway processes disconnect requests to immediately end a customer's usage session as follows:

1. An administrator submits a disconnect request from the billing server. The request includes attributes to identify the network access server (NAS), such as an Ethernet router, to disconnect.
2. RADIUS Gateway translates the request to a Disconnect-Request message, and sends it to the NAS.
3. The NAS processes the Disconnect-Request and does one of the following:

a. If the processing is successful, the NAS returns a Disconnect-ACK message to RADIUS Gateway and ends the session.

b. If the processing is not successful, the NAS returns a Disconnect-NAK message to RADIUS Gateway and does not end the session.

4. If RADIUS Gateway does not receive a response, it will retransmit the Disconnect-Request message until it does.

Sample Disconnect-Request Message

```
Code: Disconnect-Request(40)
 Identifier: 12345678
 Length: 153
 Authenticator: 1234567891234567
 NAS-IP-Address: 192.0.2.1
 User-Name: MaryRobbins
 Acct-Terminate-Cause: 6
```

This Disconnect-Request message contains termination cause 6, which indicates that an administrator reset was the cause for the session's end.

Sample Disconnect-ACK Message

```
Code: Disconnect-ACK(41)
 Identifier: 12345678
 Length: 104
 Authenticator: 9876543219876543
 Acct-Terminate-Cause: 6
```

Sample Disconnect-NAK Message

```
Code: Disconnect-ACK(42)
 Identifier: 12345678
 Length: 97
 Authenticator: 9876543219876543
 Error-Cause: 402
```

This Disconnect-NAK message contains error code 402, which indicates that a required attribute was missing from the Disconnect-Request message.

22

Configuring Policy-Driven Charging

You can implement policy-driven charging in Oracle Communications Elastic Charging Engine (ECE).

> **Caution:**
>
> Deploying policy-driven charging for 5G events requires a cloud native deployment of ECE and BRM components. 5G PCF can be used only on an ECE cloud native system.

Topics in this document:

About Policy-Driven Charging

Policy-driven charging enables you to track a subscriber's service usage and, based on that usage, change the customer's quality of service (QoS) during online charging.

For example, a subscriber purchases a package for a specific QoS to download video content. The subscriber chooses from one of many packages that you have configured with gradations in the QoS based on usage amounts in MBs, such as 100-150, 150-200, and 200-250 MBs. When the subscriber starts downloading video content from the network, you can track the number of MBs the subscriber downloads during the session. When the downloaded quantity crosses the upper threshold set for the selected QoS (for example, 150 MBs), you can use BRM's policy-driven charging to make a seamless change in the policy set for the subscriber's (video downloading) session on the network and allow a shift in the QoS from the current to the next level.

ECE supports policy-driven charging. Policy-driven charging implements network, customer, and service policies that service providers can use to improve customer experience and efficiently use network resources. Service providers can use policies for various reasons, such as controlling data usage, setting QoS, allocating bandwidth to each service, enforcing parental controls, implementing charging rules, and so on.

When you integrate Policy and Charging Rules Function (PCRF) policy clients with ECE, ECE acts as the Subscriber Profile Repository (SPR) because it stores the customer profile

information used by the PCRF. ECE offers a combined Sp and Sy interface, which the PCRF uses to retrieve customer preferences and policy counter information.

Policies can be service and network aware. You can create network-aware policies for specific access technologies where the network condition can dynamically alter prices. You can develop service-aware policies to control how a customer consumes network resources.

ECE exposes the following information in its in-memory data grid to policy clients (such as Diameter Gateway or your third-party network mediation software for online charging) to support policy-driven charging. Policy clients use the ECE policy management APIs to retrieve the information and send it to the PCRF:

- **Policy label information**

  Policy enforcement programs on the PCRF use policy labels such as status labels. For example, a QoS label might be defined as **normal-QoS** or **low-QoS**, as shown below:

```
<policy_label>
  <label>Basic Subscription</label>
  <resource_code>MBU</resource_name>
  <resource_id>100012</resource_id>
  <unit>megabyte</unit>
  <tiers>
    <tier>
      <range_start>0</range_start>
      <range_end>300</range_end>
      <status_label>normal-QoS</status_label>
    </tier>
    <tier> <range_start>301</range_start>
       <status_label>low-QoS</status_label>
  </tiers>
</policy_label>
</policy_labels>
```

  Policy label information is stored in the policy specification (offer profiles in BRM) in PDC. ECE loads this information into its data grid when it loads pricing data from PDC.

- **Policy counter information**

  The Sy interface of the ECE Java policy API transfers policy counter information from ECE to the policy client. It provides policy counter status reporting and policy counter status change notifications.

  Policy counters track a customer's usage of a service. For example, ECE tracks how many megabytes a subscriber downloads. The policy client retrieves the policy counters from ECE and sends them to the PCRF for evaluation.

- **Subscriber preferences information**

  ECE stores subscriber preferences associated with how the customer would like to receive policy notifications. Policy clients can retrieve this data from ECE using the Sp interface of the ECE Java policy API. For example, they could retrieve:

  – A customer's charging-related information (for example, if the customer purchased a Gold, Platinum, or Bronze package)

  – A customer's preferred channel for receiving notifications (for example, email or SMS)

  – A customer's language

To support policy-driven charging, ECE publishes policy notifications. Policy specifications can store threshold definitions for specific balances. ECE can use the threshold definitions to post notifications when thresholds are breached (SpendingLimit notifications). When a subscriber's preferences change, ECE publishes notifications with the new or altered preference

information (SubscriberPreference notifications). ECE sends notifications to the JMS notification queue. The policy client listens on the queue and uses the data in the notifications to send Sy and Sp messages to the PCRF.

ECE publishes policy notifications only for charge offers that have active policy sessions. When the policy client (such as Diameter Gateway) initiates policy sessions, it subscribes to receive the policy notifications on behalf of the PCRF.

When a customer purchases a new charge offer, the PCRF re-queries the policy label and policy counter (Sy data) to subscribe to the additional counters associated with the new charge offer.

About Group-Based Policy-Driven Charging

ECE supports group-based policy-driven charging where a policy counter is shared by a group of users, enabling the PCRF to define rules for a group of users.

Group-based policy-driven charging in ECE works as follows:

- The owner of a discount sharing group shares a policy counter.
- A shared discount offer is used to impact the shared policy counter.
- The shared discount is associated with a policy specification that defines policy counter thresholds.
- When a policy threshold is breached, ECE generates a notification for all users in the group.

Policy-Driven Charging Example

The following is an example of how ECE supports policy-driven charging:

- A service provider allows a customer to download 300 MBs of data per month at a normal QoS.
- The customer's counter for data downloaded resets at the beginning of each month.
- The service provider defines policy thresholds in a policy specification in PDC with the label names **normal-QoS** and **low-QoS**. These policy threshold labels are also stored in ECE.
- The service provider configures the PCRF with a policy rule that defines what action to take based on the labels defined in the policy specification. The rule determines what action to take when the customer reaches 300 MBs of data before the end of the month.
- The PCRF rule uses the label names **normal-QoS** and **low-QoS** as follows:

```
If (status_label=normal-QoS) (Bandwidth=10 Mbps)
If (status_label=low-QoS) (Bandwidth=128 kbps)
```

When the customer reaches the 300 MB data quota, the PCRF makes a policy decision to configure the Policy and Charging Enforcement Function (PCEF) so that the data transfer speed is set to 128 kilobits per second, downgraded from 10 megabits per second. The PCEF enforces this decision by changing the data transfer speed on the network switch.

Configuring Policy-Driven Charging

ECE supports in-session notifications for policy-driven charging by publishing asynchronous external notifications during a policy session. Policy clients, such as Diameter Gateway or

HTTP Gateway, consume the data in these notifications for sending in-session notifications to the PCRF.

About ECE and Policy Clients

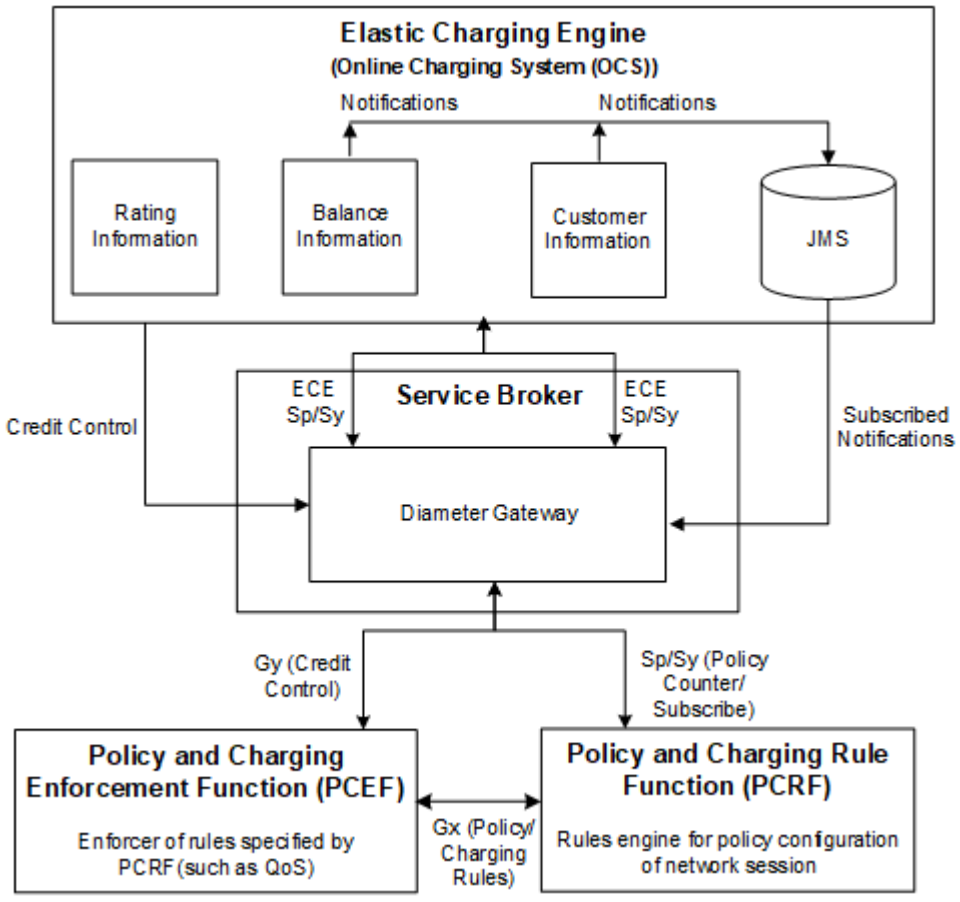
To support policy-driven charging, ECE offers a policy management API. Policy clients can use the API to retrieve data relevant to policy enforcement from its data grid.

Policy-driven charging in ECE is based on the PCRF, defined in the 3GPP TS 23.203 v9.9.0 specification. The PCRF integrates with ECE through your online network mediation software.

ECE exposes its in-memory cache so that your online network mediation software can retrieve policy counter information and policy-related subscriber preference information. ECE publishes notifications containing the policy information, and your online network mediation software uses the notifications to send the information to the PCRF for evaluation.

Figure 22-1 illustrates how ECE fits into a charging system that implements policy-driven charging.

Figure 22-1 ECE and Policy Client Integration



How ECE Processes Policy Requests for Online Network Mediation System

The following procedure describes how ECE processes requests for policy-driven charging from your online network mediation software (or from Diameter Gateway).

1. A customer starts to use a service, which initiates a network session.

   For example, the customer turns on a mobile phone that connects to a wireless network.

2. At the start of the network session, the PCEF obtains a policy configuration from the PCRF.

   The PCEF uses the Gx interface to get the policy configuration for the network session.

3. The PCRF requests policy counters and subscriber preferences from your online network mediation software (or Diameter Gateway).

   The PCRF uses the Diameter Sy/Sp interface.

4. Your online network mediation software (or Diameter Gateway) initiates a policy session with ECE that does the following:

   - Requests policy counter and status label information.

     Requests the policy counters for a specific charge offer and subscribes to receive notifications when the values of the policy counter information change.

   - Requests policy-related subscriber preferences by doing *one* of the following:

     – Retrieves the value for a specified set of subscriber preferences and subscribes to receive notifications when the values of the preferences change during the policy session.

     – Retrieves only the values for a specified set of subscriber preferences and does *not* subscribe to receive notifications when the values of the preferences change during the policy session.

   Your online network mediation software (or Diameter Gateway) uses the PolicySessionRequest ECE Java combined Sy/Sp (implemented as Sh) interface, which uses the SubscribeNotificationRequest procedure and the UserDataRequest procedure.

5. ECE sends a policy response to your online network mediation software (or to Diameter Gateway), which does the following:

   - Indicates whether the request succeeded or failed and provides a list of reasons supporting the response.

   - Sends the status of the policy counters for the specified service. If the service is not specified, returns the information for all services:

     – Sends the policy specification (offer profile) name configured for the service.

     – Sends the status label associated with the policy counter.

     – Sends an *effective time* for the values of the policy counters. After the effective time expires, the PCRF is expected to send another request for policy counter and status label information (send another SpendingLimitReportRequest).

     – Sends the label name of the next probable status that applies after the effective time expires. For example, **Medium_QoS**.

     – Sends the *delay interval*. The PCRF can use the delay interval and the effective time to determine when to query for the policy counters again.

     ECE uses the SpendingLimitReportResponse procedure of the ECE Java Sy interface.

   - Sends the subscriber preferences.

     ECE uses the SubscribeNotificationResponse procedure of the ECE Java Sp interface.

6. The PCRF rules engine interprets the information and installs a policy on the PCEF, which the PCEF enforces.

7. A charging session is established, and the PCEF sends a Ro message to your online network mediation software (or Diameter Gateway).
8. Your online network mediation software (or Diameter Gateway) initiates a charging session with ECE.
9. ECE publishes policy notifications for the following:
   - Changes to the policy counter status for the policy counters the PCRF subscribed for (Sy data) at the beginning of the policy session.
   - Changes to the subscriber preferences the PCRF subscribed for (Sp data), if any, at the beginning of the policy session.
10. Your online network mediation software (or Diameter Gateway) consumes the policy notifications and sends the data to the PCRF.
11. As the charging session continues, ECE performs credit control functions: rates events, authorizes usage events only if adequate balance is available, administers threshold checks based on the current balance and consumed reservation of the customer balance.
12. When ECE detects a policy threshold breach during the charging session, it publishes a policy notification to the JMS notification queue containing the policy counter's new status. Your online network mediation software (or Diameter Gateway) sends the data to the PCRF.

    The customer balance change that causes the policy threshold breach could occur as a result of any of the following:

    - Usage requests coming from the network mediation system
    - Update requests coming from BRM (a subscription activity in the customer management system)
    - Top-ups coming from top-up systems

    > **Note:**
    >
    > If ECE detects multiple breaches during a session, it sends notifications sequentially. That is, it sends a notification for the first breach. Then, ECE waits for an acknowledgment from PCRF that it has received the notification before sending the subsequent breach notification.

13. The PCRF evaluates the new policy counter values and the associated policy status labels and installs a new policy configuration on the PCEF.

    The new policy is established dynamically during the charging session.
14. The customer stops using his service, which ends the network session.
15. Your online network mediation software (or Diameter Gateway) terminates the charging session with ECE.
16. Your online network mediation software (or Diameter Gateway) terminates the policy session with ECE.

Configuring Breach Tolerance for Policy-Tier Thresholds

In policy-driven charging, policy-tier thresholds must be crossed to trigger the implementation of business rules, such as reduced QoS for subscribers who download excessive data.

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 available. 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. 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. The session ends when usage reduces the 80 MB balance to the point that the remaining balance cannot support the service.
- **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.

To configure a tolerance for policy-tier threshold breaches:

1. Before charging servers are started, open *ECE_home***/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***/bin** directory.
4. Start Elastic Charging Controller (ECC):

   ```
   ./ecc
   ```

5. Start your charging servers:

   ```
   start server
   ```

6. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
7. Expand the **ECE Configuration** node.
8. Expand **charging.policyConfig**.
9. Expand **Operations**.

10. Select **setPolicyTolerance**.
11. For *each* balance element (policy counter) to which policy specifications 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.

About Integrating Policy Clients with ECE

Policy clients such as Diameter Gateway integrate with ECE by using the ECE policy 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 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 Interface

To support policy-driven charging, ECE offers policy management APIs. The ECE Sy interface enables policy clients to subscribe for and retrieve spending limit information about policy counters from ECE. The ECE Sp interface enables policy clients to subscribe for and retrieve customer preference information relevant to policy enforcement from ECE.

The following sections describe each interface:

- About the ECE Sy Interface
- About the ECE Sp Interface

ECE also supports a combined ECE Sy and Sp interface that enables 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 such as Diameter Gateway 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.

- SpendingLimit Notification

  ECE uses this notification to report statuses of requested policy counters for one or more services 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 such as Diameter Gateway 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 customer-preference data updates to the policy client 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 services. The policy client, such as Diameter Gateway, uses the ECE policy Sp query procedure to retrieve extended customer and service information.

To retrieve extended information from ECE using the policy Sp query request, you must configure the extended service and customer information in ECE.

To configure the query for extended service and customer information:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.policyConfig**.

4. Expand **Operations**.
5. Select **setDsl**.
6. Do the following for each type of service or customer you want the policy client to query:
   a. For the **alias** parameter, replace String with the alias for the extended information to use in the policy query request.

      Configured aliases are included in the policy query request.
   b. For the **dsl** parameter, replace String with 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)
   c. 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.

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, such as Diameter Gateway, use this procedure to retrieve Sp and Sy information and subscribe or unsubscribe (for the PCRF) to receive updates to Sp and Sy data. This request combines 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 request processing results.

  The policy client transfers the information to the PCRF.

About Calculating Maximum Authorization for Policy-Driven Charging Sessions

For policy-driven charging sessions, ECE readjusts the requested quota based on the following data:

- Current balance
- Used reservation across all parallel sessions
- Nearest threshold in the policy specification

For example, consider this situation:

- Current balance: 80 MB
- Used reservation across all parallel sessions (iPhone, video, computer): 35 MB
- Nearest threshold in the policy specification: 140 MB

Under those conditions, if ECE receives an authorization request for an additional 30 MB, that request exceeds the 140 MB threshold by 5 MB (80 MB + 35 MB + 30 MB = 145 MB).

Therefore, unless a breach tolerance of 5 MB or more is configured, ECE authorizes only 25 MG.

Configuring ECE to Reject Spending Limit Requests Without Counters

For Sy subscriptions, you can configure ECE to reject a Spending Limit Request (SLR) if there are no policy counters available for the subscriber.

To configure ECE to reject SLRs when no policy counters are available:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.policyConfig**.
4. Expand **Attributes**.
5. Set the **syRejectNoCounters** attribute to **true**.

About the Policy Management API

To use the policy management API, clients call the submitPolicy API with PolicyRequest.

For details about the policy management API, see the documentation for **oracle.communication.brm.charging.brs**, **oracle.communication.brm.charging.messages.policy**, and **oracle.communication.brm.charging.messages.query** (for user data request/response information) in *Elastic Charging Engine Java API Reference*.

Part VI
Customizing ECE

This part provides information about customizing charging in Oracle Communications Elastic Charging Engine (ECE). It contains the following chapters:

- Customizing Rating
- ECE Sample Programs
- Testing ECE

23

Customizing Rating

You can use the Oracle Communications Elastic Charging Engine (ECE) extensions to customize BRM Gateway, Diameter Gateway, HTTP Gateway, RADIUS Gateway, pre-rating, post-rating, post-charging, and post-update processes. ECE extensions include sample implementations that guide you in implementing your custom business logic.

> **Caution:**
>
> Deploying charging for 5G with HTTP Gateway (5G CHF) requires a cloud native deployment of ECE and BRM components. The HTTP Gateway can be used only on an ECE cloud native system.

Topics in this document:

Operational Considerations

All pre-rating, post-rating, post-charging, and post-update extensions must be implemented in a single class respectively. This class can delegate to additional implementations if multiple extensions are being implemented.

Extensions data is loaded into a replicate cache in Coherence and the amount of data loaded into the cache must be taken into consideration when sizing for Java.

Configuring Extensions

You configure implementation classes for the diameter-request processing, HTTP-request processing, and usage-request processing extension points through JMX management by using a JMX editor.

To configure the implementation classes for the diameter-request processing, HTTP-request processing, and usage-request processing extension points:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **charging.extensions**.
4. Expand **Attributes**.
5. Specify values for the following attributes as needed:

- **brmGwExtension**
- **diameterGyExtension**
- **diameterSyExtension**
- **httpExtension**
- **ocsBypassExtension**
- **postChargingExtension**
- **postRatingExtension**
- **postRatingMidSessionExtension**
- **postUpdateExtension**
- **preRatingExtension**
- **preRatingMidSessionExtension**
- **radiusAccountingExtension**
- **radiusAuthExtension**
- **ratingExtension**

About Performance with Extensions

If extensions are activated, they are called for during every usage request. Always consider performance for the code you run in the extensions.

The extensions framework provides an extensions cache mechanism that provides the lowest latency access to the extensions data. It is recommended that you use the extensions cache mechanism rather than external data sources.

You can use the PerformanceMonitor MBean to monitor CPU usage of server nodes and client nodes. When building your charging extensions, the methods of the PerformanceMonitor MBean enable you to monitor the performance impacts of your extensions. For example, 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.

About Logging in Extensions

Logging is available in your extensions. You use the Log4j logger to write messages to the server node log file. For example:

```
extensionContext.getLogger().debug("Hello World!" + extensionContext);
```

About Extension Exceptions

If ECE needs to reject a usage request, an **ExtensionsException** error can be thrown to reject the usage request and report a "CUSTOM_EXTENSION_ERROR" reason code in the response.

For details about **ExtensionsException,** see *Elastic Charging Engine Java API Reference*.

About Extension Security

To ensure security for the extension, follow these best practices:

- Enable JMX security
- Enable ECE cluster node security
- Ensure strict governance of OS accounts
- Follow secure Java coding practices
- Implement string code review processes
- Run latency-sensitive performance tests on the extensions hooks
- Use JAR signing

Extension Points

You customize BRM Gateway, Diameter Gateway, HTTP Gateway, RADIUS Gateway, pre-rating, rating, post-rating, post-charging, and post-update processes with these extension points:

- BRM Gateway Request Processing Extension Points
- Diameter-Request Processing Extension Points
- HTTP Gateway Request Processing Extension Points
- RADIUS-Request Processing Extension Points
- Update-Request Processing Extension Points
- Usage-Request Processing Extension Points

BRM Gateway Request Processing Extension Points

You use BRM Gateway request processing extension points to update an external notification that is bound for the BRM Gateway. It adds data for calling a specific BRM opcode. This allows you to update attributes in the ECE cache through the BRM-ECE synchronization process. See "About Synchronizing Data Between BRM and ECE".

ECE publishes the external notification to the ECE Notification topic, where it is retrieved by BRM Gateway. BRM Gateway uses the information in the notification to call the specified opcode, which in turn updates customer data in the BRM database. BRM then resynchronizes the customer data with the ECE cache.

BRM Gateway provides these extension points for external notification flows that are targeted for the BRM Gateway:

- **PostCharging extension.** The role of the PostCharging extension is to retrieve data for a specific opcode.
- **BRMCustomOpCodeCall extension.** The role of the BRMCustomOpCodeCall extension is to enrich external notifications with an input flist for a specific BRM opcode. This extension is called before the external notification is published to the ECE Notification topic.

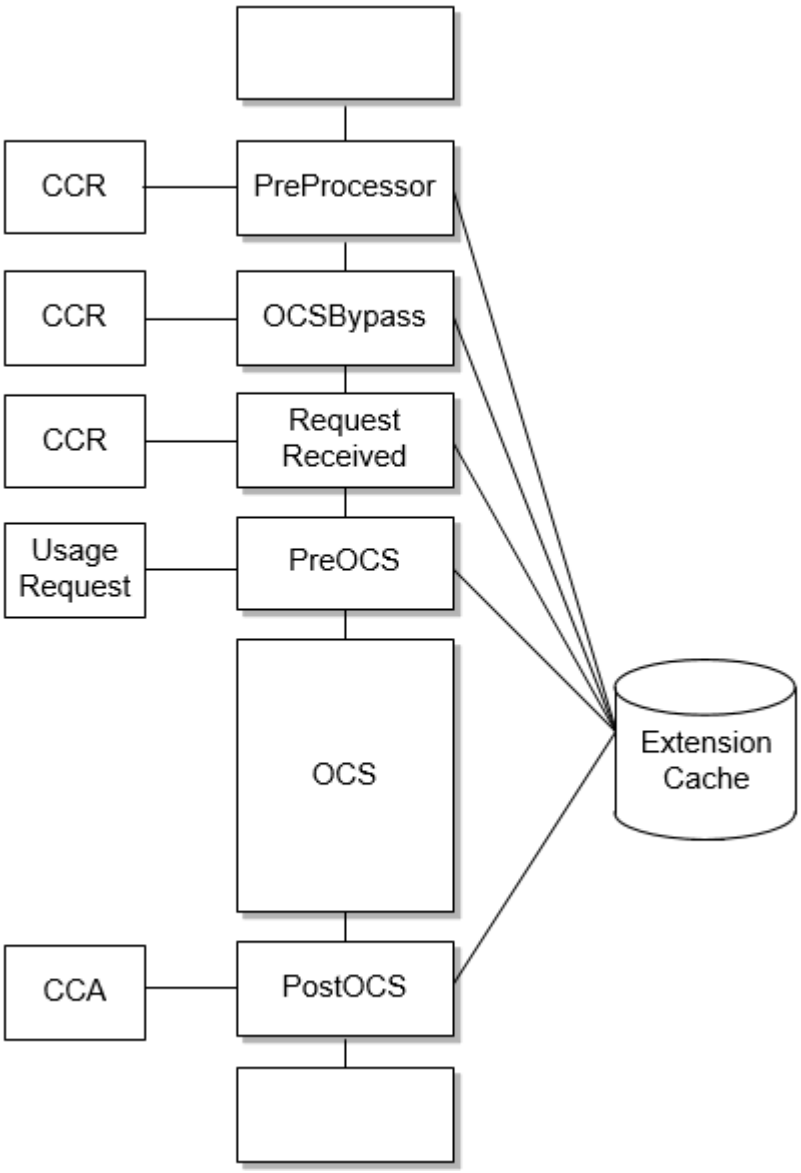
Diameter-Request Processing Extension Points

Diameter Gateway provides extension points for Credit Control Request (CCR) and Credit Control Answer (CCA) flows:

- **RequestReceived extension.** The role of the RequestReceived extension is to manipulate the CCR attribute-value pair (AVP) before the usage request is processed by Diameter Gateway and to provide an immediate response that bypasses the online charging system (OCS) completely.
- **PreProcessor extension.** The role of the PreProcessor extension is to add or remove any AVP in the custom response before the usage request is processed by Diameter Gateway and to provide an immediate response that bypasses the OCS completely.
- **PreOCS extension.** The role of the PreOCS extension is to manipulate the mapped ECE usage request payload and perform enrichment that is not possible in the RequestReceived extension.
- **PostOCS extension.** The role of the PostOCS extension is to manipulate the CCA AVPs before the diameter response is returned to the diameter client.
- **OCSBypass extension**. The role of the OCSBypass extension is to bypass the rating of Diameter CCRs received during a planned maintenance or an unplanned downtime of ECE and persist them to Oracle NoSQL.

Figure 23-1 shows the diameter-request processing extension points.

Figure 23-1 Diameter-Request Processing Extension Points



HTTP Gateway Request Processing Extension Points

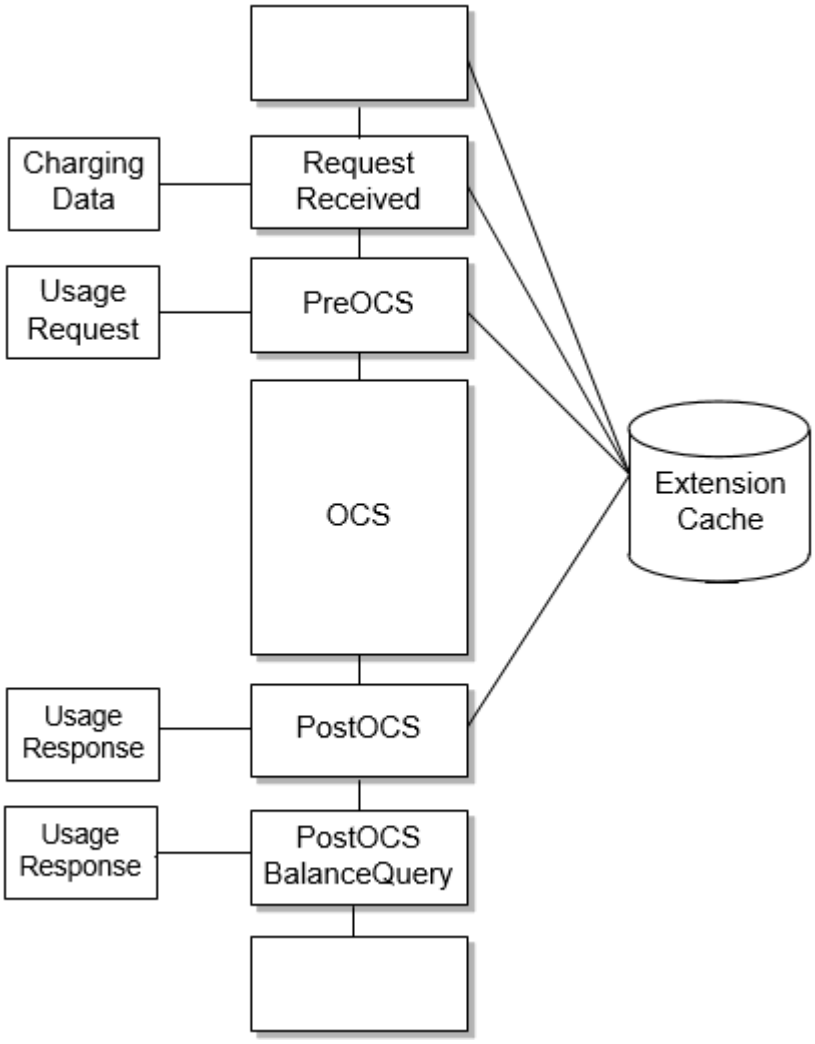
HTTP Gateway provides extension points for 5G flows:

- **RequestReceived extension.** The role of the RequestReceived extension is to manipulate the charging data before the usage request is processed by HTTP Gateway and to provide an immediate response that bypasses the online charging system (OCS) completely.
- **PreOCS extension.** The role of the PreOCS extension is to manipulate the mapped ECE usage request payload to perform enrichments that are not possible in the RequestReceived extension.
- **PostOCS extension.** The role of the PostOCS extension is to manipulate the ECE usage request before the HTTP Gateway response is returned to the 5G client.

- **PostOCSBalanceQuery** extension. The role of the PostOCSBalanceQuery extension is to manipulate the ECE usage response before the HTTP Gateway response is returned to the 5G client.

Figure 23-2 shows the HTTP Gateway request processing extension points.

Figure 23-2 HTTP Gateway Processing Extension Points



RADIUS-Request Processing Extension Points

RADIUS Gateway provides extension points for authentication and accounting flows.

Authentication Extension Points

RADIUS Gateway provides extension points for the authentication flow:

- **RequestReceived extension.** The role of the RequestReceived extension is to add or update a custom AVP before the authentication request is processed by RADIUS Gateway and to provide an immediate response that bypasses the OCS completely.

- **CustomEAPChallenge extension.** The role of the CustomEAPChallenge extension is to send custom access-challenge request to the RADIUS client when the Extensible Authentication Protocol (EAP) is used for authentication.
- **PreOCS extension.** The role of the PreOCS extension is to perform any actions related to authentication that are required before the RADIUS request is sent to ECE.
- **CustomAuth extension.** The role of the CustomAuth extension is to implement the custom EAP authentication methods.
- **CustomEncode extension.** The role of the CustomEncode extension is to implement the custom hashing algorithm that is used on passwords during authentication when the Password Authentication Protocol (PAP) is used for authentication.
- **PostOCS extension.** The role of the PostOCS extension is to add or update a custom AVP before the authentication response is returned to the RADIUS client.

Figure 23-3 shows the RADIUS-request processing extension points for EAP authentication.

Figure 23-3 Extension Points for EAP Authentication

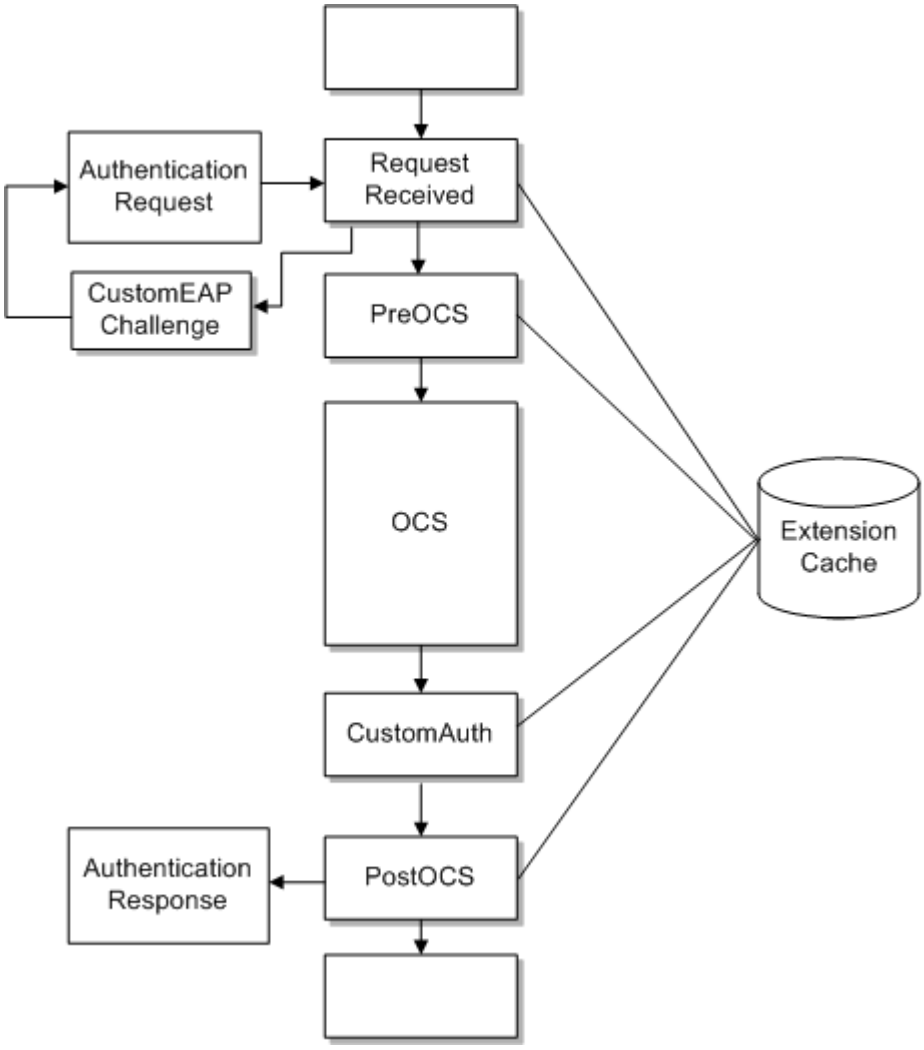
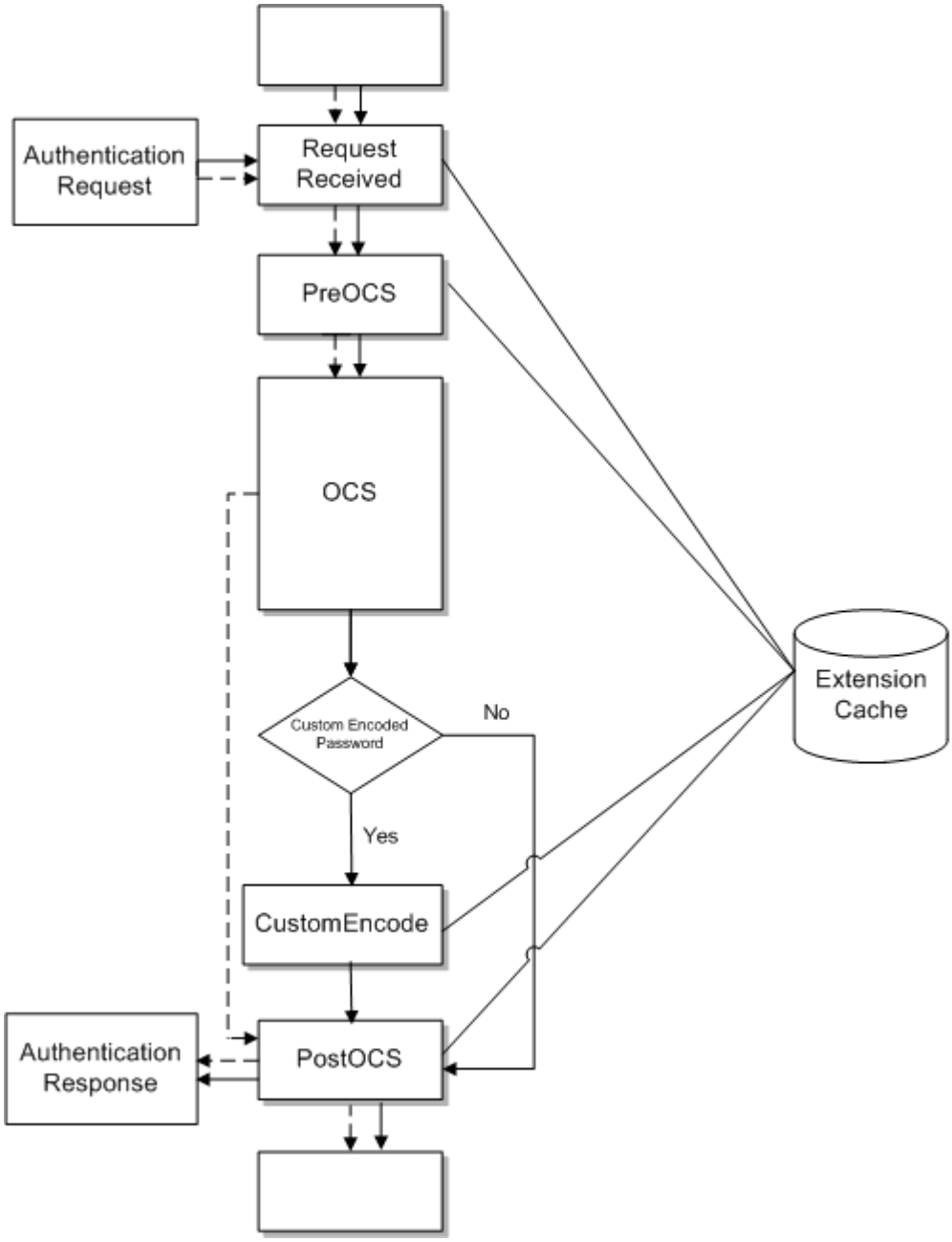


Figure 23-4 shows the RADIUS-request processing extension points for PAP and Challenge-Handshake Authentication Protocol (CHAP) authentication. The solid line depicts PAP authentication and the dotted line depicts CHAP authentication in this figure.

Figure 23-4 Extension Points for PAP and CHAP Authentication



Accounting Extension Points

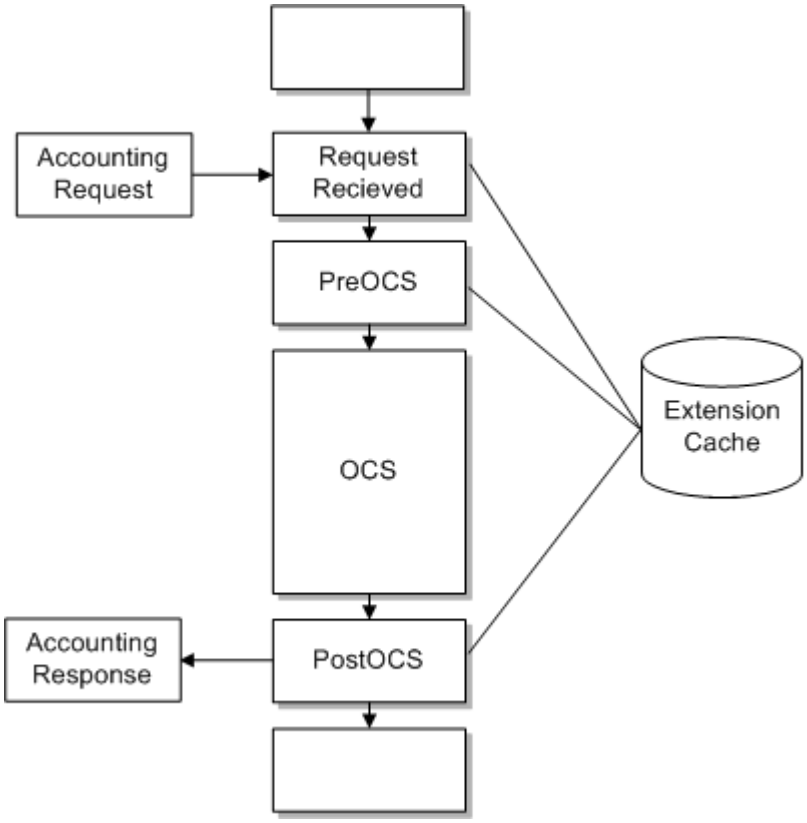
RADIUS Gateway provides extension points for accounting flow:

- **RequestReceived extension.** The role of the RequestReceived extension is to add or update a custom AVP before the accounting request is processed by RADIUS Gateway and to provide an immediate response that bypasses the OCS completely.
- **PreOCS extension.** The role of the PreOCS extension is to enrich the usage request before the usage request is sent to ECE for accounting purposes.

- **PostOCS extension.** The role of the PostOCS extension is to add or update a custom AVP before the accounting response is returned to the RADIUS client.

Figure 23-5 shows the RADIUS-request processing extension points for accounting.

Figure 23-5 Extension Points for Accounting



Update-Request Processing Extension Points

ECE provides an extension point for post-update extensions in the updates-processing flow. The role of the post-update extension is to enrich and filter external notifications. This extension is called after receiving update requests and before publishing the external notifications.

Usage-Request Processing Extension Points

ECE provides extension points in the rating flow: before charge calculation, after charge calculation (prior to making a balance impact), and after charging (after applying a balance impact).

Table 23-1 describes the role of each extension point in the rating flow.

Table 23-1 Rating Flow Extension Roles

Extension Point	Role
Pre-Rating Extension	Alter the usage request

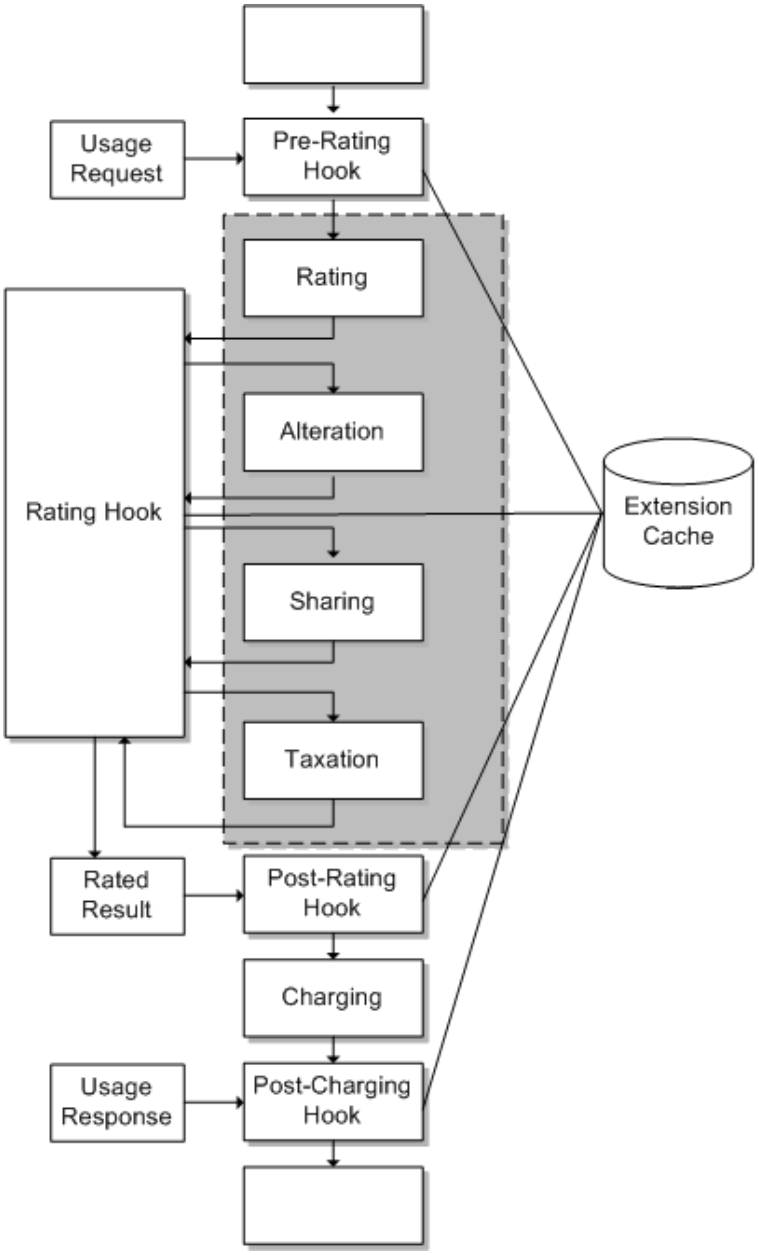
Table 23-1 (Cont.) Rating Flow Extension Roles

Extension Point	Role
Post Rating Extension	Alter the rated result
Rating Extension	Alter rated results after each of the following processes: rating, alteration, sharing, and taxation
Post-Charging Extension	Enrich the usage response

You cannot customize rating during the rating, alteration, and tax calculation processes, only before and after. Access is provided to a custom data store that provides low-latency access to data required for the extensions; for example, customer data and balance data.

Figure 23-6 shows the usage-request processing extension points.

Figure 23-6 Usage-Request Processing Extension Points



Implementing the Extensions Logic

The **BRMCustomOpCodeCallExtension**, **DiameterGyExtension**, **DiameterSyExtension**, **HTTPExtension**, **PreRatingExtension**, **PostRatingExtension**, **PostChargingExtension**, **RadiusRequest**, and **RadiusResponse** interfaces expose **initialize()** and **shutdown()** methods that are called by the hook framework when the server starts up and when it shuts down. Use these methods to configure your own internal data structures related to the extensions business logic.

For diameter-request processing extension points, a different method is called for each extension point.

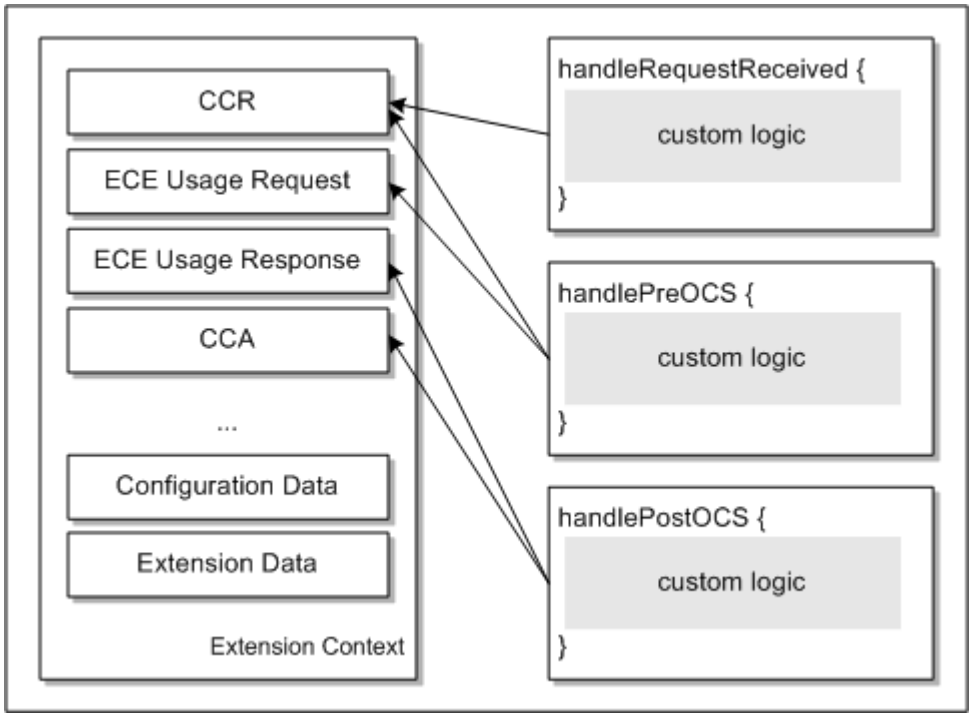
- **handleRequestReceived().** Called for every CCR that is processed by the charging flow.

- **handlePreOCS().** Called for every CCR and usage request that is processed by the charging flow.
- **handlePostOCS().** Called for every CCA and usage response that is processed by the charging flow.

All methods expose relevant ExtensionContext data for accessing the ExtensionsDataRepository, AppConfigRepository, and other extensions-related contexts.

Figure 23-7 shows the data used in the diameter-request processing extension points.

Figure 23-7 Data Used in Diameter-Request Processing Extension Points



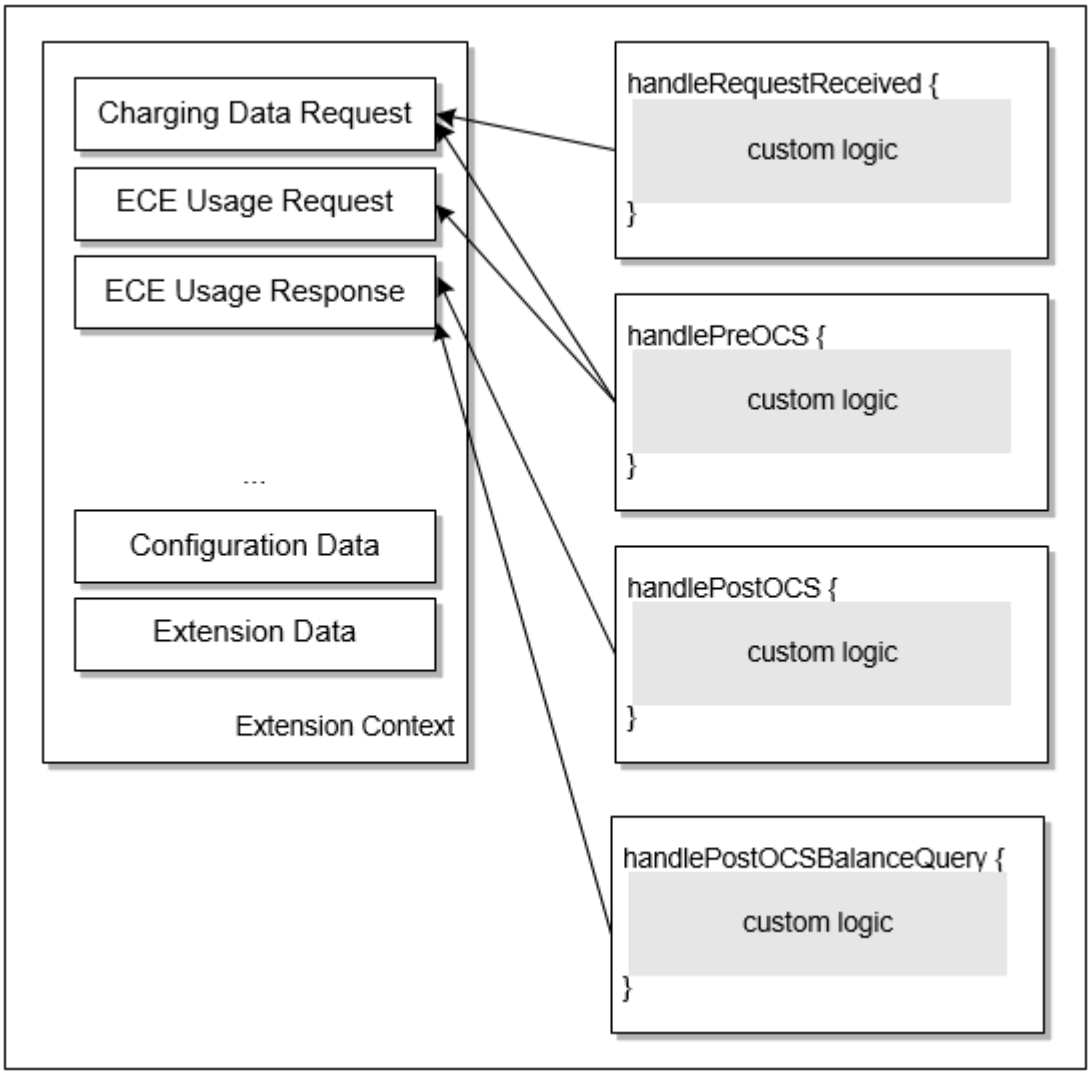
For HTTP-request processing extension points, a different method is called for each extension point.

- **handleRequestReceived().** Called for every charging data request that is processed by the charging flow.
- **handlePreOCS().** Called for every usage request that is processed by the charging flow.
- **handlePostOCS().** Called for every usage response that is processed by the charging flow.
- **handlePostOCSBalanceQuery().** Called for every usage response that is processed by the charging flow.

All methods expose relevant ExtensionContext data for accessing the ExtensionsDataRepository, AppConfigRepository, and other extensions-related contexts.

Figure 23-8 shows the data used in the HTTP-request processing extension points.

Figure 23-8 Data Used in HTTP-Request Processing Extension Points



For extension points that process requests from RADIUS clients, the **RadiusRequest** and **RadiusReply** interfaces are exposed to the extension points through the **ExtensionContext** methods.

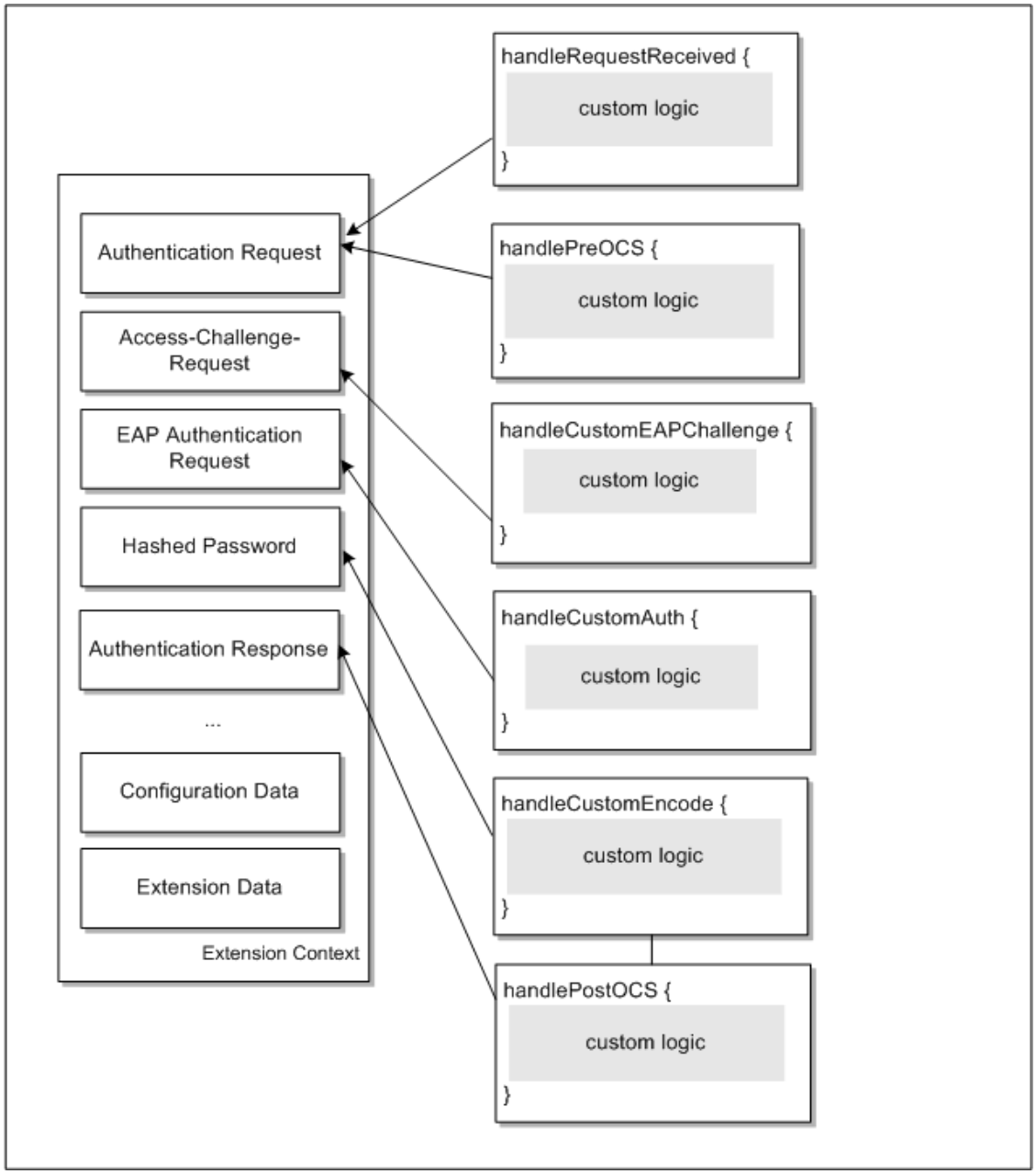
For authentication-related extension points, the following methods are called by the authentication flow:

- **handleRequestReceived().** Called for every authentication request that is processed by the authentication flow.
- **handlePreOCS().** Called to perform any actions related to authentication that are required in the authentication flow.
- **handlePostOCS().** Called for each authentication response that is processed by the authentication flow.
- **handleCustomEAPChallenge().** Called to send custom access-challenge requests to the RADIUS client in the EAP authentication flow.

- **handleCustomAuth().** Called to implement a custom EAP authentication method in the authentication flow.
- **handleCustomEncode().** Called to implement the custom hashing algorithm that is used on passwords in the PAP authentication flow.

Figure 23-9 shows the data used in the RADIUS-request processing extension points for authentication.

Figure 23-9 Data Used in RADIUS-Request Processing Extension Points for Authentication

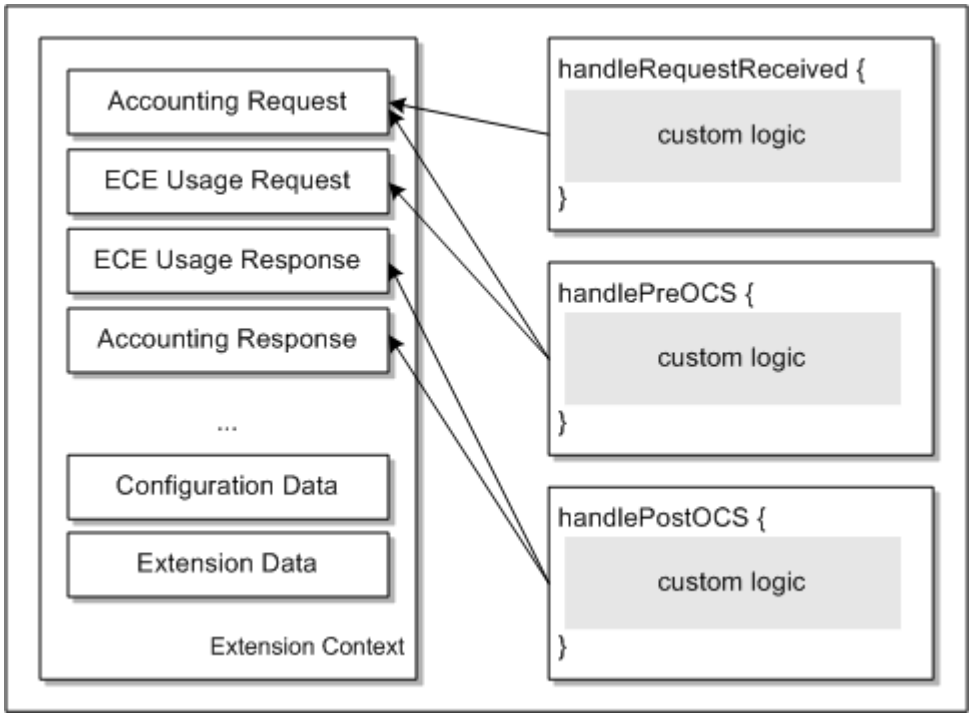


For accounting-related extension points, the following methods are called by the accounting flow:

- **handleRequestReceived().** Called for every accounting request that is processed by the accounting flow.
- **handlePreOCS().** Called for every accounting request and usage request that is processed by the accounting flow.
- **handlePostOCS().** Called for every accounting response and usage response that is processed by the accounting flow.

Figure 23-10 shows the data used in the RADIUS-request processing extension points for accounting.

Figure 23-10 Data Used in RADIUS-Request Processing Extension Points for Accounting



For usage-request processing extension points, the **execute()** method is called for every usage request, rated result, usage response, and notification that is processed by the charging flow.

For the rating extension point, the following methods are called by the charging flow:

- **handlePostApplyCharge().** Called to alter rated results after calculating charges (rating).
- **handlePostApplyAlteration().** Called to alter rated results after calculating discounts (alteration).
- **handlePostApplyDistribution().** Called to alter rated results after calculating charge distribution (sharing).
- **handlePostApplyTaxation().** Called to alter rated results after calculating taxes (taxation).

All methods expose relevant ExtensionContext data for accessing the ExtensionsDataRepository, AppConfigRepository, and other extensions-related contexts.

Figure 23-11 shows the data used in the pre-rating extension point.

Figure 23-11 Data Used in Pre-Rating Extension Point

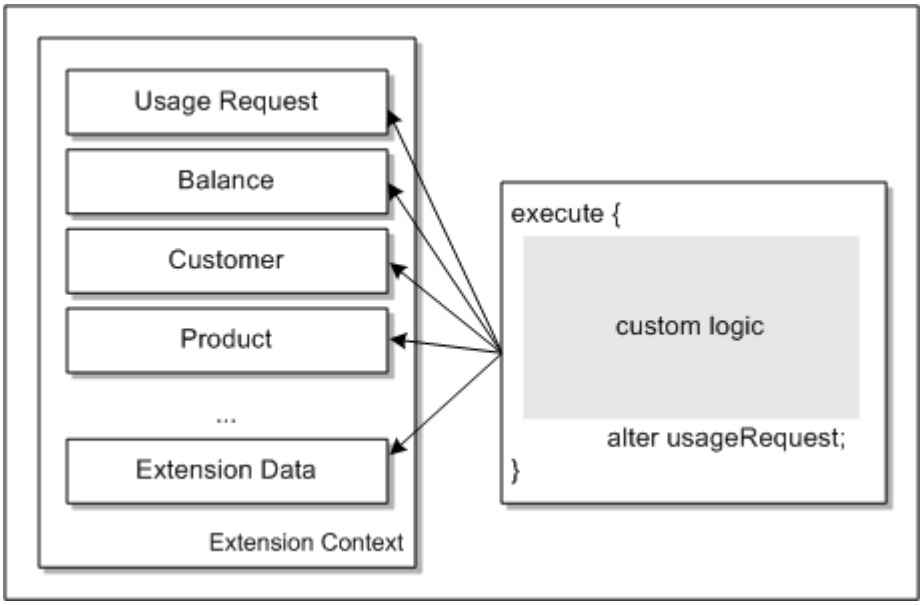


Figure 23-12 shows the data used in the rating extension point.

Figure 23-12 Data Used in Rating Extension Point

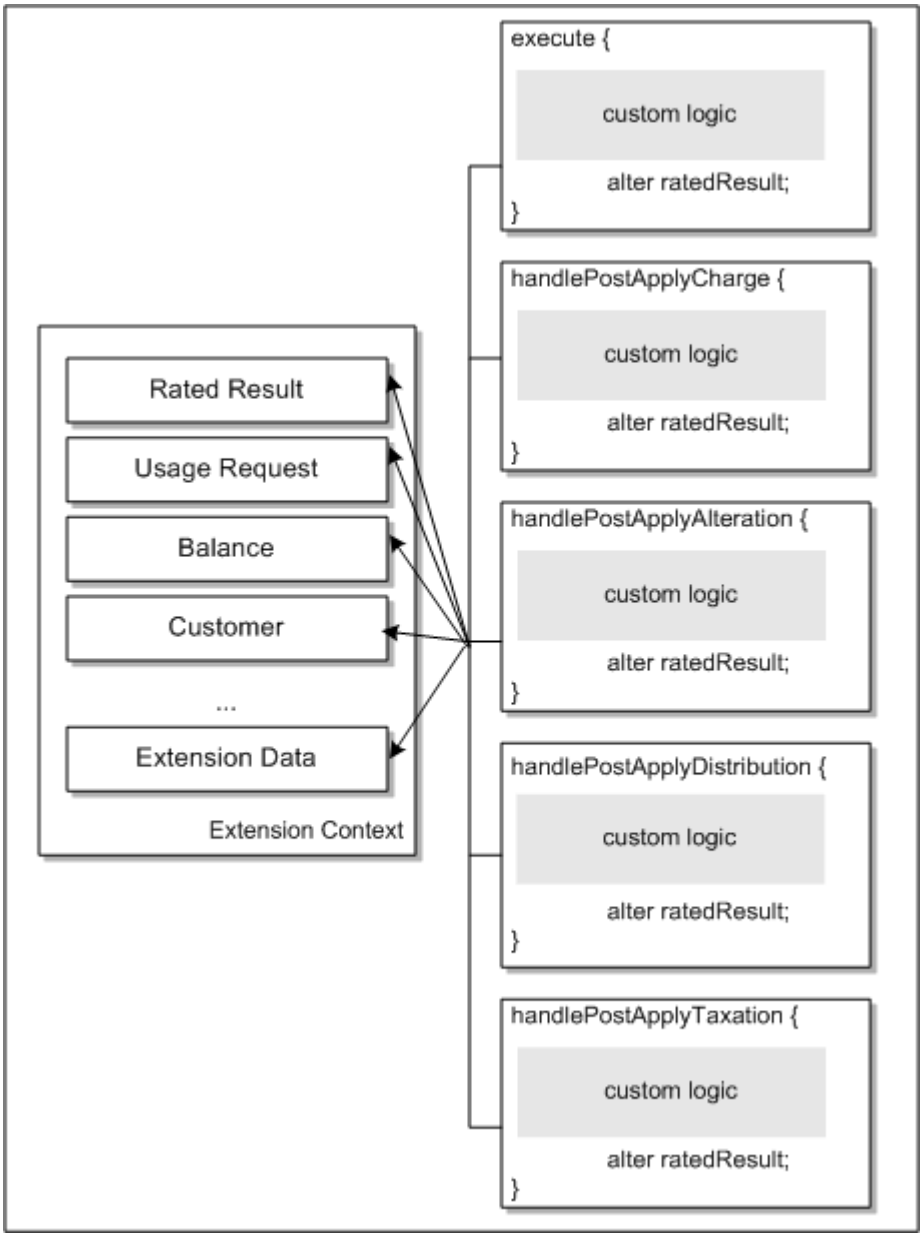


Figure 23-13 shows the data used in the post-rating extension point.

Figure 23-13 Data Used in Post-Rating Extension Point

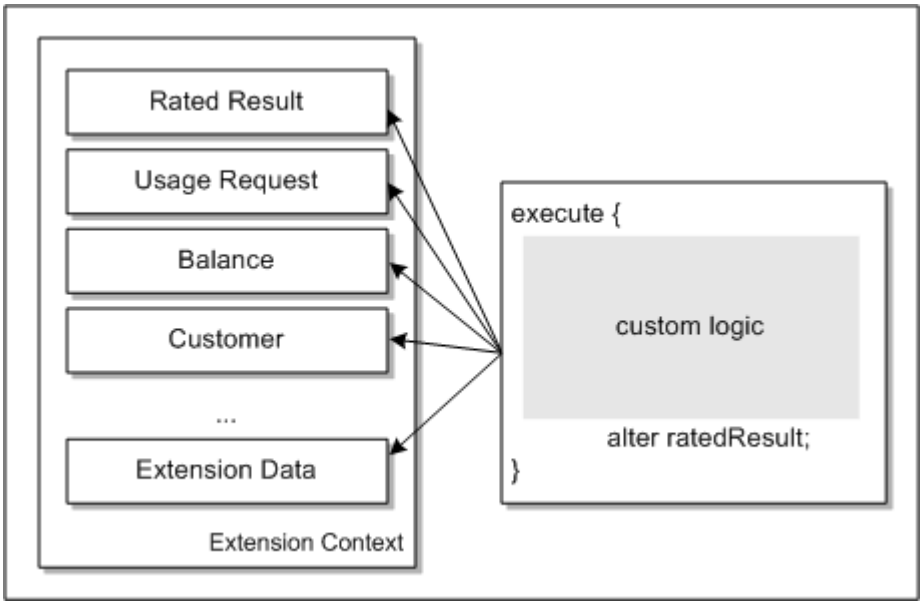
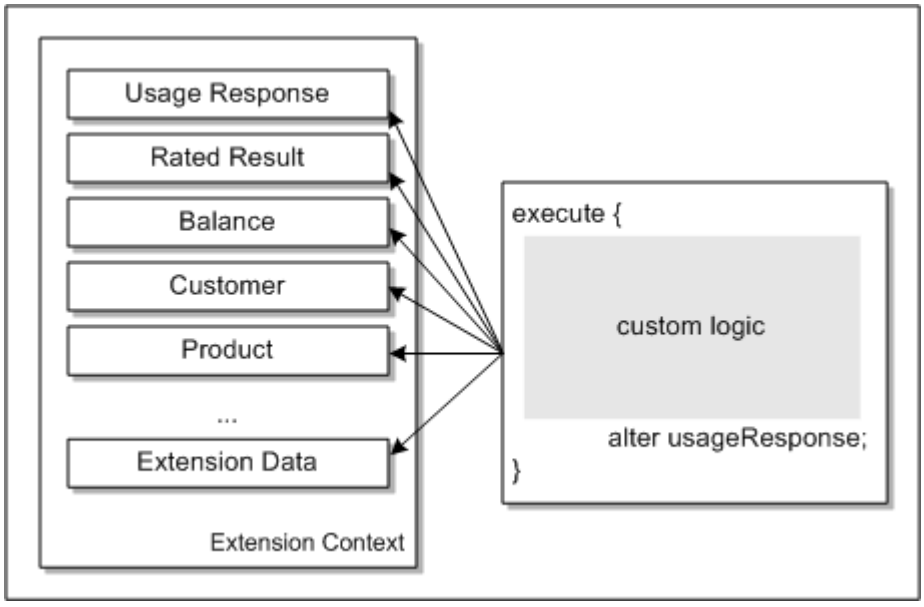


Figure 23-14 shows the data used in the post-charging extension point.

Figure 23-14 Data Used in Post-Charging Extension Point



ECE provides build and deployment capabilities in the form of shell scripts. If any third-party libraries need to be used inside the custom extensions logic, copy the third-party JAR files to the *ECE_home***/lib** directory, where *ECE_home* is the directory in which ECE is installed. After the JAR files have been copied, they need to synchronize across to the other servers in the cluster. Synchronization is done by running the **sync** command in Elastic Charging Controller (ECC).

Custom extensions logic implementation classes that implement the **GyExtension**, **PreRatingExtension**, **RatingExtension**, **PostRatingExtension**, **PostChargingExtension**, **RadiusRequest**, and **RadiusResponse** interfaces and their dependencies must be packaged in JAR format. Ensure the packaged extensions JAR files are available to the ECE runtime environment in the *ECE_home***/lib** directory.

BRMCustomOpCodeCall Extension

The BRMCustomOpCodeCall extension enriches external notifications with an input flist for a BRM opcode. The extension is called before publishing the external notification to the ECE Notification topic, where it will be retrieved by the BRM Gateway.

You can modify the external notifications to include an input flist for a different BRM opcode.

CustomAuth Extension

The CustomAuth extension implements custom EAP authentication methods; for example, EAP-POTP, EAP-PSK etc.

The extension can access this data:

- EAP-Authentication-Request
- System configuration

You can use a custom EAP authentication method if the RADIUS client does not support EAP-TTLS or EAP-MD5.

CustomEAPChallenge Extension

The CustomEAPChallenge extension sends a custom access-challenge request to the RADIUS client when custom EAP authentication mechanisms are used for authentication.

The extension can access this data:

- Access-Challenge-Request
- System configuration
- Extensions data

You can use the extension point to send the custom access-challenge request to the RADIUS client when EAP is used for authentication.

CustomEncode Extension

The CustomEncode extension implements the custom hashing algorithm that is used on passwords for authentication.

The extension can access this data:

- Encoded Password
- System configuration
- Extensions data

You can use the custom hashing algorithm on passwords for authentication. For example, typically the password from the RADIUS client is hashed (stored in the hash format) for PAP authentication. However, if the password is hashed in any other format, you implement the CustomEncode extension point to hash the incoming password.

OCSBypass Extension

The OCSBypass extension bypasses rating of CCRs received during ECE downtime or a planned maintenance activity. The CCRs are persisted to Oracle NoSql. When ECE is restored, the persisted requests are replayed to the ECE charging server for rating and updating balance impacts along with the real-time requests. The entire diameter request is accessible and modifiable and the entire request information is passed to the custom extension. When rating bypass is enabled and the custom extension is run, Diameter Gateway returns responses that you configured in the custom extension. If the extension is not enabled but bypassing of rating is enabled, ECE returns Diameter Result Code 5012 for the described conditions, which must be handled in your custom extension implementation.

PreOCS Extension

The PreOCS extension manipulates usage request payloads before the usage request is sent to ECE, so that the request can match the business requirement. In addition, the PreOCS extension performs any actions related to authentication that are required before the request is sent to ECE. This extension is called before any rating, discounting, or alteration logic has been invoked.

The extension can access this data:

- Credit Control Request
- Authentication Request
- Accounting Request
- ECE Usage Request
- System configuration
- Extensions data

You can modify the ECE usage request payload. For example, certain usage request manipulations can be made only when the ECE usage request payload is accessible. The usage request manipulations are done in this extension.

PreProcessor Extension

The PreProcessor extension manipulates CCR requests before the usage request is created, so that the request can match the business requirement. The PreProcessor extension can bypass charging or mutate the contents of the Sy message. This extension is called before any rating, discounting, or alteration logic has been invoked.

The extension can access this data:

- Credit Control Request
- Request AVPs
- ECE payload
- System configuration
- Extensions data

PostOCS Extension

The PostOCS extension manipulates CCA, accounting, usage, or authentication responses to match the business requirement before returning the response to the client. This extension is called after charging, authentication, and accounting has been completed and recorded.

The extension can access this data:

- Credit Control Request
- Accounting Response
- Authentication Response
- ECE Usage Response
- Diameter Credit Control Answer
- System configuration
- Extensions data

You can modify the CCA, accounting response, and authentication response. For example, you can manipulate AVPs to adapt to non-standard diameter and RADIUS implementations.

PostOCSBalanceQuery Extension

The PostOCSBalanceQuery extension manipulates ECE usage responses to match the business requirement before returning the response to the HTTP client. This extension is called after a balance query has been completed and recorded.

Pre-Rating Extension

The pre-rating extension enhances the usage request based on customer, service, balance, product, and system data so that the usage request can match the business requirement. This extension is called before any rating, discounting, or alteration logic has been invoked.

The extension can access this data:

- ECE usage request information
- Customer data (including profile data)
- Service data (including profile data)
- Balance information
- Product information
- System configuration
- Extensions data

You can modify usage requests. For example, you modify usage requests to:

- Alter the requested quota. This is implemented in the sample extensions provided.
- Apply special rates or discounts (such as birthday discounts) for calls based on the extended rating attributes of both calling customers and called customers.
- Generate a midsession-rated event when an update operation occurs. This is implemented in the sample extensions provided.

You can also modify the values of the pricing attributes with custom logic. This enables you to override a product price.

Post-Rating Extension

The post-rating extension modifies the rated event based on customer, service, balance, product, and system data. This extension is called after any rating, discounting, or alteration logic has been invoked.

The extension can access this data:

- Balance information
- Customer data (including profile data)
- ECE usage request information
- Product information
- Shared customers (if part of a sharing relationship)
- Service data (including profile data)
- System configuration
- Extensions data
- Rated result

You can modify rated events in the following ways:

- You can modify the balance impact amount, GL code, tax code, balance element or invoice data for rating impacts generated from ECE.
- Generate a midsession-rated event when an update operation occurs. This is implemented in the sample extensions provided.

You can also create new tax rating impacts, such as implementing a tax on tax.

Rating Extension

The rating extension modifies the rated results after each of the following processes: rating, alteration, sharing, and taxation.

The extension can access this data:

- Customer (including profile)
- Shared customer (if part of a sharing relationship)
- Product (including profile)
- Balance information
- System configuration
- Extensions
- Rated result

You can alter rated results to modify charges, discounts, charge sharing, taxes, and item assignments. For example:

- After rating, you can alter charges based on the zones, such as standard and geographic zones.

- After taxation, you can alter custom item types for the rating impacts generated from ECE, such as charge, alteration, and distribution rating impacts.

RequestReceived Extension

The RequestReceived extension manipulates the CCR, charging data request, authentication request, or accounting request so that the request can match the business requirement and provides an immediate response that bypasses the OCS completely. This extension is called before any rating, discounting, or alteration logic has been invoked.

The extension can access this data:

- Credit Control Request
- Authentication Request
- Accounting Request
- System configuration
- Extensions data

You can modify the CCR, authentication, or accounting request. For example, you can manipulate AVPs to adapt to non-standard diameter implementations. Certain CCR, authentication, and accounting request types may not be supported by ECE, Diameter Gateway, or RADIUS Gateway, so a response can be created in this extension and returned immediately, bypassing the OCS.

Post-Charging Extension

The post-charging extension enriches usage responses and external notifications, excluding Advice of Charge (AOC) notifications. This extension is called after charging is completed but before the usage response is generated.

The extension can access this data:

- Customer (including profile data and subscriber preferences)
- Shared customers (if part of a sharing relationship, subscriber preferences)
- Service (including profile data, subscriber preferences, life cycle state)
- Balance information (including current request impacts)
- Business profile
- System configuration
- Extension data
- Rated result

You can modify the usage responses and external notifications. You can use the post-charging extension point to:

- Enrich usage responses and external notifications. You can add custom data as AVPs to the response and notification. For example, you can add a custom language preference to a customer's subscriber preferences. The custom values will be available as diameter hooks for further propagation.
- Filter out external notifications that you do not want to be published to external systems.

Post-Update Extension

The post-update extension enriches the external notifications; excluding the AoC notifications. This extension is called after receiving update requests and before publishing the external notifications.

The extension can access this data:

- Customer (including profile data and subscriber preferences)
- Shared customers (if part of a sharing relationship, subscriber preferences)
- Service (including profile data, subscriber preferences, life cycle state)
- Balance information (including current request impacts)
- Business profile
- System configuration
- Extension data
- Rated result

You can modify the external notifications with custom logic. You can use the post-update extension to:

- Enrich external notifications. For example, you can add custom data to any external notification that is generated to provide additional data, such as spending limit notifications.
- Filter out external notifications that you do not want to be published to external systems. For example, when billing is run, ECE generates subscribe-notification-request (SNR) notifications for all impacted resources. You can filter out unneeded SNR notifications and publish only required notifications to external systems.

To use the post-update extension, you must define the post-update extension's fully qualified class name in the *ECE_home***/config/management/charging-settings.xml** file.

Extensions Cache

The extensions framework provides a generic repository from which data required for the pre-rating, post-rating, and post-charging extensions can be uploaded and used. The data format is described in a specifications file that describes the format of the data. The extensions specification allows a DataLoader to load the data into the ECE extensions cache. Example 23-1 is an example of a specifications file for the post-rating extension:

Example 23-1 Sample Tax Table

```
/*
 * Sample tax table
 */
ExtensionDataSpecification
   Info {
      Name "tax_table_0001"
   }
   Payload {
      Block "TAX_ROW" {
         String "TAXCODE"
         String "PKG"
         Decimal "RATE"
         DateTime "START"
         DateTime "END"
```

```
            String "LEVEL"
            String "LIST"
            String "DESCRIPTION"
            String "RULE"
         }
      }
   }
```

Example 23-2 shows the associated data to load into the cache using the specification file above:

Example 23-2 Example Data File

```
# This is a sample csv file containing typical tax configuration data.
#
#TaxCode |Pkg |Rate    |Start      |End        |Level |List  |Description |Rule
 usage   |U   |0.05    |01/01/2013 |12/31/2014 |Fed   |US    |USF         |Std
 usage   |U   |0.08    |01/01/2013 |12/31/2014 |Sta   |CA    |USTA        |Std
 usage   |U   |0.06    |01/01/2013 |12/31/2014 |Fed   |US    |USF         |Std
 usage   |U   |0.085   |01/01/2013 |12/31/2014 |Sta   |CA,AZ |USTA        |Std
 purchase|V   |0.08525 |01/01/2013 |12/31/2014 |Sales |CA    |PSLS        |Std
```

Extensions Cache API

The extensions repository provides the following APIs for managing extensions data:

- **putExtensionsData().** Takes a single key-value pair of string as a key and value being an ExtensionsData object.
- **putExtensionsDataCollection().** Takes a map of key-value pairs of string keys and value being ExtensionsData objects.
- **findExtensionsData().** Returns an ExtensionsData object for a given key.
- **getAllExtensionsData().** Returns a read-only collection of all extensions data from the repository.

The extension includes these repository constraints:

- You must generate a unique key as a string for one ExtensionsData object (entry in the extensions cache) at the time of retrieval of the extensions data from the cache.
- Because the extensions data is replicated across the whole cluster, the amount and size of data is limited to what a given Java heap can manage. You can also adjust the Java heap size. Refer to the Java provisioning guidelines.
- Changes made to the extensions data after it is loaded are expensive to make due to its cache topology. Avoid frequent updates to the extensions data, especially in a larger cluster.
- The framework does not dictate the type of data source that extensions data are loaded from. The provided **SampleExtensionsDataLoader** SDK demonstrates loading the data from a comma-separated-value (CSV) file using extensions domain-specific language APIs. This sample is a recommended design, but it should not be used as a reference about how to store data.

Sample Extensions

The BRM SDK includes sample extensions. For information about how to use the samples, see "How To Use the Sample Extensions" and "Validating Sample Extensions".

For information about each sample extension, see the following:

- BRM Gateway Extension – Creating Opcode Flist
- Diameter Gateway Extension – Gy Service
- Diameter Gateway Extension – Sy Service
- HTTP Gateway Extension – Service
- OCSBypass Extension – Bypassing Rating
- Pre-Rating Extension – Dynamic Quota Management
- Pre-Rating Extension – Retrieving Function Values for Discount Expressions
- Pre-Rating Extension – Generating Midsession-Rated Event
- Pre-Rating Extension – Overriding Price in Product Offerings
- Post-Rating Extension – Complex Taxation
- Post-Rating Extension – Generating Midsession-Rated Events
- Post-Rating Extension – Adding or Deleting Rating Periods
- Post-Charging Extension – Adding Custom Data to Usage Responses and Notifications
- Post-Charging Extension – Overriding Dynamic Quota
- Post-Charging Extension – Adding or Modifying Redirection Rules
- Post-Charging Extension – Creating Custom Notifications for Top Ups
- Post-Update Extension – Enriching External Notifications
- Rating/Charging Extension – Triggering RAR Notifications
- Rating Extension – Custom Item Assignment
- Extensions Data Load Sample

How To Use the Sample Extensions

To use the sample extensions:

1. ECE SDK is installed under **$SDK_HOME**. The directory listing is shown below:

```
$ ls -l
total 124
drwxr-xr-x 2 ecsuser ecsuser  4096 Jun 21 10:47 bin
drwxr-xr-x 2 ecsuser ecsuser  4096 Jun 21 10:47 bin
drwxr-xr-x 3 ecsuser ecsuser  4096 Jun 21 10:47 config
-rw-r--r-- 1 ecsuser ecsuser     5 Jun 21 10:47 VERSION
```

2. Under the source directory, create a pre-extensions or post-extensions Java Class using the Extensions API and other libraries (samples are provided as a part of the ECE SDK.)

```
$ cd source
$ cd oracle/communication/brm/charging/sdk/extensions
$ ls -l
total 28
-rw-r--r-- 1 ecsuser ecsuser  6427 Jun 21 10:47 SampleExtensionsDataLoader.java
-rw-r--r-- 1 ecsuser ecsuser XXXXX Jun 21 10:47 SampleOCSByPassExtension.java
-rw-r--r-- 1 ecsuser ecsuser 12194 Jun 21 10:47 SamplePostRatingComplexTaxation
-rw-r--r-- 1 ecsuser ecsuser  6066 Jun 21 10:47 SamplePreRatingExtension.java
-rw-r--r-- 1 ecsuser ecsuser XXXXX Jun 21 10:47 SamplePostChargingExtension.java
```

3. Write custom logic in Java and copy it under the directory. The Java source is under the package **oracle.communication.brm.charging.sdk.extensions**:

```
$SDK_HOME/source/oracle/communication/brm/charging/sdk/extensions
```

4. Change ECE_HOME in the **script build_deploy_extension.sh** file under **$SDK_HOME/bin/extensions**:

```
### configuration begin
ECE_HOME=$ECE_HOME
### configuration end
```

5. Compile the extensions class using the shell script: **build_deploy_extension.sh**.

   a. Each extensions file has to be compiled individually (similar to SDK programs).

   b. Any additional ECE or third-party library required for the extensions needs to be added to the CLASSPATH in the **build_deploy_extension.sh.** script

```
$sh $SDK_HOME/bin/extensions/build_deploy_extension.sh build
SampleDiameterGyExtension
$sh $SDK_HOME/bin/extensions/build_deploy_extension.sh build
SamplePostRatingComplexTaxationExtension
$sh $SDK_HOME/bin/extensions/build_deploy_extension.sh build
SamplePreRatingExtension
$sh $SDK_HOME/bin/extensions/build_deploy_extension.sh build
SamplePostChargingExtension
$sh $SDK_HOME/bin/extensions/build_deploy_extension.sh build
SampleOCSByPassExtension
```

   Do the following optional step if external data needs to be loaded. To compile the sample extensions loader use the **sample_extensions_loader.sh** shell script:

```
$sh $SDK_HOME/bin/extensions/sample_extensions_loader.sh build
SampleExtensionsDataLoader
$sh $SDK_HOME/bin/extensions/sample_extensions_loader.sh run
```

6. Deploy creates a single JAR file (**ece.extensions-VERSION-SNAPSHOT.jar**) with all the extensions classes and copies the JAR file under **$ECE_HOME/lib**. The JAR file is copied only to the driver node. It has to be propagated to other ECE nodes in the grid manually or use a rolling upgrade.

```
$sh $SDK_HOME/bin/extensions/build_deploy_extension.sh deploy
```

7. Define the fully-qualified class names of the pre-rating, bypass rating, rating, post-rating, post-charging, and post-update extensions by configuring the **charging.extensions** MBean in the **ECE Configuration** node using a JMX editor. For instructions, see "Configuring Extensions".

```
preRatingExtension="oracle.communication.brm.charging.sdk.extensions.
SamplePreRatingExtension"
RatingExtension="oracle.communication.brm.charging.sdk.extensions.
SampleRatingExtension"
postRatingExtension="oracle.communication.brm.charging.sdk.extensions.
SamplePostRatingComplexTaxationExtension"
postChargingExtension="oracle.communication.brm.charging.sdk.extensions.SamplePostCha
rgingExtension"
diameterGyExtension="oracle.communication.brm.charging.sdk.extensions.
SampleDiameterGyExtension"
postUpdateExtension="oracle.communication.brm.charging.sdk.extensions.SamplePostUpdat
eExtension"
ocsBypassExtension="oracle.communication.brm.charging.sdk.extensions.SampleOCSByPassE
xtension"
```

8. Start or restart the ECE server nodes and enable logging for the extensions by setting **oracle.communication.brm.charging.extensions.client** to DEBUG via JMX and verify that the custom extensions are run as a part of rating logic. You can also turn on debug logging for the RATING module using the JMX console.

Validating Sample Extensions

After the server nodes are bought up initially or by using a rolling upgrade, send a sample SDK usage request. Enable debug for the RATING module and verify the server log contains these messages:

```
SamplePreRatingExtension invoked
PostRatingComplexTaxationSampleExtension executed
```

BRM Gateway Extension – Creating Opcode Flist

The sample program **SampleBRMGwCustomOpCodeExtension** shows how to use the ECE Extensions API to enrich an external notification with an input flist for a specified opcode.

The sample program builds an input flist for the PCM_OP_BILL_DEBIT opcode.

Diameter Gateway Extension – Gy Service

The sample program **SampleDiameterGyExtension** shows how to use the immediate-response feature based on an incoming AVP value.

The logic: If Service-Context-Id is OFFLINE, respond with Diameter Code DIAMETER_REDIRECT_INDICATION and set the Redirect-Host AVP value

Diameter Gateway Extension – Sy Service

The sample program **SampleDiameterSyExtension** shows how to use the ECE extensions API to suspend the Sy interface when a subscriber is suspended. The sample program does the following:

- Bypasses rating by calling the **setBypassOCS()** method and setting the DiameterResultCode to DIAMETER_REDIRECT_INDICATION.

  > **Note:**
  >
  > To skip the bypass, comment or remove the **setBypassOCS()** method invocation under the **handleRequestReceived()** method.

- Adds or removes AVPs in the Custom Sy Response.
- Modifies the request data time and product type in the policy request.
- Modifies the status and reason codes in the policy response.

HTTP Gateway Extension – Service

The sample program **SampleHTTPExtension** shows how to use the ECE Extensions API to override the requested number of units and perform call screening.

- The **handleRequestReceived()** method overrides the requested units for a call.
- The **handlePreOCS()** method sets the timestamp of when the call originally occurred.

- The **handlePostOCS()** method logs a message when call screening is done and returns a response to the 5G client.
- The **handlePostOCSBalanceQuery()** method logs a message when a balance query occurs and returns a response to the 5G client.

OCSBypass Extension – Bypassing Rating

The sample program **SampleOCSByPassExtension** shows how to use the ECE extensions API to bypass rating during ECE downtime. When bypass rating is enabled, the CCRs are persisted to Oracle NoSQL. When ECE is restored, the persisted requests are replayed to the ECE charging server for rating and updating balance impacts along with the real-time requests. Using this extension, you can write your own logic for modifying the AVP of incoming diameter messages and for creating the diameter responses.

Pre-Rating Extension – Dynamic Quota Management

The sample program **SamplePreRatingExtension** shows pre-rating custom logic. It illustrates sample logic for the pre-rating scenarios.

Dynamic Quota Management – Modifying Quota Based on Network Type

The **SamplePreRatingExtension** program shows how to use the ECE extensions API to modify the input request quantity based on the input network type where the customer balance is greater than a predefined amount.

Logic: If the ORIGIN_NETWORK network field is:

"*3G_UTRAN*" and USD balance greater than 50 **then** set quota to 10 MB

or

"*4G_UTRAN*" and USD balance greater than 50 **then** set quota to 100 MB

Dynamic Quota Management – Modifying Requested Quota

The **SamplePreRatingExtension** program shows how to use the ECE extensions API to update the input request to modify the requested quota. You can use this sample program to access the ECE cache to derive a quota and then update the requested quota in the input request. ECE then uses the derived quota for allocation.

Dynamic Quota Management – Modifying Default Quota Configuration

The **SampleDynamicQuotaExtension** program shows how to use the ECE extensions API to update the following attributes in the input request to modify the quota configuration based on your requirements:

- **Quota holding time.** Specifies how long a granted quota can be idle before the reservation is released.
- **Volume quota threshold.** Specifies how much of the granted quota must be consumed before a subscriber can request additional quota. This attribute is configured per service, event, and number of granted units.
- **Validity time.** Specifies whether the validity time can be set to a fixed value per service-event combination at runtime. This attribute is independent of the number of units in the granted quota.

Pre-Rating Extension – Retrieving Function Values for Discount Expressions

The **SamplePreRatingExtension** program shows how to use the ECE extensions API to retrieve the value referenced by the function in a discount expression. You create a custom function in ECE that defines an event profile attribute. You can use the **SamplePreRatingExtension** program to call the custom function. ECE then adds the defined event profile attribute and its value to the usage request.

If the PDC pricing specifies a 10% discount for all accounts active less than 12 months, the logic is the following:

If **customerActiveMonths** value is:

- < 12 then apply a discount of 10%
- > 12 then apply a discount of 0%

Pre-Rating Extension – Generating Midsession-Rated Event

The **SamplePreRatingMidSessionExtension** program shows how to use the ECE extensions API to generate a midsession-rated event when an update request contains the following:

- A specified balance, such as greater than or equal to $500
- A changed field in a usage request
- A specified account field
- A specified product type, such as TelcoGsmTelephony

When an update request matches the criteria, the extension generates a midsession-rated event and adds the reason why it was created. It adds these reason codes to the event's **midSessionCDRSplitReason** field: CONFIGURED_DURATION_REACHED and RATING_CONDITION_CHANGE.

Pre-Rating Extension – Overriding Price in Product Offerings

The **SamplePreRatingExtension** program shows how to use the ECE extensions API to override the price specified in product offerings. You create a custom function in ECE that overrides the default value of the pricing attributes in dynamic tags, which are the XML elements configured in PDC. You can use the **SamplePreRatingExtension** program to call the custom function. The overridden values are then populated in the event profile map in the request specification data. ECE uses the overridden values to determine the price when processing usage requests.

Post-Rating Extension – Complex Taxation

The sample program **SamplePostRatingComplexTaxationExtension** shows how to use the ECE extensions API to override or augment post-rating results using complex taxation as an example. The program iterates over the tax rating periods and overrides tax impacts by modifying the rating periods for federal tax and then generates new tax periods for the state tax.

It applies the tax rate based on the pre-loaded tax configuration data in the extensions cache. The tax rate is determined based on tax code, tax time, and validity, which are all based on the request start time. The default configuration for the tax code used in the extension must exist in the ECE configuration.

The extension uses the following logic:

1. Determines the federal tax rate from the tax configuration table using the tax code, request start time.
2. Calculates the federal tax based on this tax rate.
3. Modifies the original impact in the tax rating period based on the taxable impact from the linked charge, alteration, or distribution rating period.
4. Determines the state tax rate from the tax configuration table using the tax code, request start time.
5. Calculates state tax based on this tax rate.
6. Creates new tax rating period for the state tax and link it to the original charge/alteration/distribution rating period.

This program also shows how to use the extensions API to override the invoice data in the rating result. The overridden value is persisted into the CDR output file.

Post-Rating Extension – Generating Midsession-Rated Events

The **SamplePostRatingMidSessionExtension.java** sample program shows how to use the ECE extensions API to generate a midsession-rated event when a rated event contains the following:

- A balance granted by an offer is exhausted
- A balance bucket is expired
- The charge offer used during rating is different from that of the ongoing session

When an event matches the criteria, the extension generates a midsession-rated event and adds the reason why it was created. It adds these reason codes to the event's **midSessionCDRSplitReason** field: CONFIGURED_VOLUME_REACHED and CONFIGURED_TIME_OF_THE_DAY_CROSSED.

Post-Rating Extension – Adding or Deleting Rating Periods

The **PostRatingConsolidateRatingPeriods** sample program shows how to use the ECE extensions API to:

- Add a single rating period with the consolidated charge for all the rating periods of type CHARGE.
- Delete all the existing rating periods of type CHARGE.

You can use this sample program to access the ECE cache and override the rating periods in the final rated results by adding or deleting rating periods.

Post-Charging Extension – Adding Custom Data to Usage Responses and Notifications

The sample program **SamplePostChargingExtension** shows how to use the ECE extensions API to add custom data to the following:

- Usage responses. You add the data as AVPs. For example, you can add a custom language preference to a customer's subscriber preferences. The custom values are available as diameter hooks for further propagation.

- External notifications. You add the data as key-value pairs. For example, you can add information such as calling number, called number, event type, and balance group to these notifications, such as credit threshold notifications.

Post-Charging Extension – Overriding Dynamic Quota

The **SamplePostChargingExtension** program shows how to use the ECE extensions API to override the quota attributes, such as quota holding time and volume quota threshold, in the usage response. You provide the data as name-value pairs. ECE then accesses the data and updates the usage response.

Post-Charging Extension – Adding or Modifying Redirection Rules

The **SamplePostChargingExtension** program shows how to use the ECE extensions API to add or modify rules for redirecting a subscriber session to a service portal applicable to the business scenario. You can add or modify them based on the customer conditions, such as whether the customer has insufficient funds or whether the customer has an inactive account.

Post-Charging Extension - Enriching Notifications

The **SamplePostChargingExtension** program shows how to enrich the notification. You can use this sample program and update the payload with the following data:

- Session ID
- Notification event type
- Even timestamp
- Original balance
- Current balance

The sample program uses the **getCustomDataMap()** method to retrieve custom data and put it in the Notification's Payload. **CustomDataMap** is a map of String-String, where our custom name-value pair is included in the response.

Post-Charging Extension – Creating Custom Notifications for Top Ups

The sample program **SamplePostChargingExtension** shows how to use the ECE extensions API to create a custom notification for topping up a customer's balance element through the BRM PCM_OP_BILL_DEBIT opcode.

The sample program retrieves data for a specified service type, request type, and balance element combination. It also specifies the BRM opcode to call. The sample program includes the following:

- The **getUsageRequest()** method, which fetches the input service usage request.
- The **getBalance()** method, which specifies that the logic applies to the USD balance element.
- The **getCustomDataMap()** method, which fetches the **customDataMap** for the custom data set by the user.
- The **addCustomServiceEventWithOpCode()** method, which specifies to call the PCM_OP_BILL_DEBIT opcode.

- The **getOriginalBalance()** method, which calculates and returns the original balance valid at the start time of the current usage session.
- The **getCurrentBalance()** method, which returns the current balance for the given resource considering the balance items valid during the specified validity range.
- The **getServiceEventType()** method, which retrieves the **ServiceEventType** associated with a service event.

Post-Update Extension – Enriching External Notifications

The sample program **SamplePostChargingExtension** shows how to use the ECE extensions API to add custom data to external notifications that are generated to provide additional data. You add the data as name-value pairs. ECE then accesses the data and updates the external notifications. For example:

```
s.getCustomDataMap().put("NotificationType",
      String.valueOf(s.getServiceEventType()));
```

In the above code snippet, we fetch the notification type from the Service Event and store its value in the **customDataMap**. This will subsequently be populated in the Notification Payload.

Rating/Charging Extension – Triggering RAR Notifications

The following sample programs show how to use the ECE extensions API to trigger server-initiated reauthorization request (RAR) notifications in the rating and charging flow:

- **SampleRarPreRatingExtension**
- **SampleRarPostRatingExtension**
- **SampleRarPostChargingExtension**

These programs access the ECE cache data and trigger RAR notifications to retrieve the exact reservation balance for performing any business operation.

In the custom logic, if the **sendGenericRARNotification** is set to **true**, ECE generates generic RAR notifications for all Diameter sessions for the client. The Rating-Group and Service-Identifier are not set in those notifications. If **sendGenericRARNotification** is set to **false**, ECE generates service-specific RAR notifications with Rating-Group and Service-Identifier set in the notifications.

If **sendRarForSharedBalances** is set to **true**, ECE generates RAR notifications for all active sessions using a sharing group balance. For example, assume accounts A, B, C, and D and in a sharing group, and accounts A and C have active sessions using the sharing group balance. If an RAR is triggered for account C, ECE sends RAR notifications for accounts A's and C's active sessions. If **sendRarForSharedBalances** is **false**, ECE generates RAR notifications for all active sessions initiated by the subscriber. Using the previous example, ECE would generate RAR notifications for account C's active sessions only.

Rating Extension – Custom Item Assignment

The sample program **SampleRatingExtension** shows how to use the ECE extensions API to alter the custom item type for rating impacts.

It alters custom item types for the rated results based on the data accessible through the rating extension. The default configuration for the custom item type used in the extension must exist in the ECE configuration.

The extension uses the following logic:

1. After taxation, determines the custom item type to be used based on the data accessible through the rating extension.
2. Assigns the rating impacts to the custom bill items based on the new custom item type.

Extensions Data Load Sample

The sample program **SampleExtensionsDataLoader** demonstrates how the extensions data repository can be used and how to load data into the repository.

The data loader used for extensions is located at *ECE_home***/ocecesdk/source/oracle/communication/brm/charging/sdk/extensions**.

The following SDK artifacts are provided:

- **tax_configuration.spec**
  - This is a specification for tax codes. The specification expects a single block with a cardinality of 1 per ExtensionsData.
  - Contains the following attributes:
    - Tax code (String)
    - Pkg (String)
    - Rate (Decimal)
    - Start (DateTime)
    - End (DateTime)
    - Level (String)
    - List (String) Description (String)
    - Description (String)
    - Rule (String)
- **tax_configuration_data.csv**
  - A pipe-delimited CSV file. This file acts as a data source for tax codes.
- **SampleExtensionsDataLoader**
  - A class that reads the CSV file, prepares the payload as per tax specification, and uses the extensions repository to put a collection of ExtensionsData.
  - Also asserts if the number of ExtensionsData put in the Repository are the same as the total being read.

24

ECE Sample Programs

You use the sample programs included in the Oracle Communications Elastic Charging Engine (ECE) SDK to learn how to call the ECE APIs.

See Elastic Charging Engine Java API Reference for more information about the ECE SDK.

Topics in this document:

About the ECE Sample Programs

The ECE SDK includes sample programs that demonstrate how to use the ECE API for sending requests to ECE.

You can 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).

You can also look at the sample program source code to see how it works. For example, if you want to write a program that sends a unit-based debit request to ECE, examine SampleDebitRefund to:

- View the methods to use in your code.
- How to use the libraries and calls.

Finding the Sample Programs

Table 24-1 shows the ECE SDK software directory structure, where *ECE_home* is the directory in which the ECE Server software is installed.

Table 24-1 Elastic Charging Engine Sample Program Directories

Directory	Description
ECE_home/ococesdk/bin	Directories that contain shell scripts for compiling and running various types of sample programs.
ECE_home/ococesdk/bin/extensions	Shell scripts for extension-implementation sample programs.
ECE_home/ococesdk/bin/notification	Shell scripts for notification sample programs.
ECE_home/ococesdk/bin/plugin	Shell scripts for the custom plug-in sample programs.
ECE_home/ococesdk/bin/policy	Shell scripts for policy sample programs.
ECE_home/ococesdk/bin/query	Shell scripts for query sample programs.
ECE_home/ococesdk/bin/update	Shell scripts for update sample programs.
ECE_home/ococesdk/bin/usage	Shell scripts for usage sample programs.
ECE_home/ococesdk/config	Configuration files common to all sample programs.
ECE_home/ococesdk/config/extensions	Configuration files for extension-implementation sample programs.
ECE_home/ococesdk/source	All Java sample programs.
ECE_home/ococesdk/source/oracle/communication/brm/charging/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/ococesdk/source/oracle/communication/brm/charging/sdk/notification	Source files for notification sample programs.
ECE_home/ococesdk/source/oracle/communication/brm/charging/sdk/plugin	Source files for custom plug-in sample programs.
ECE_home/ococesdk/source/oracle/communication/brm/charging/sdk/policy	Source files for policy sample programs.
ECE_home/ococesdk/source/oracle/communication/brm/charging/sdk/query	Source files for query sample programs.
ECE_home/ococesdk/source/oracle/communication/brm/charging/sdk/update	Source files for update sample programs.
ECE_home/ococesdk/source/oracle/communication/brm/charging/sdk/usage	Source files for usage sample programs.

Descriptions of the Sample Programs

All of the sample programs can work with the ready-to-use sample data included with the ECE Server software installation. The sample programs are supported on the Linux platform.

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**.

> **Note:**
>
> The ECE sample programs do not work well with data you load using the simulator **loader** utility.

For a list of each sample program, their descriptions, the shell scripts used to compile and run them, and the applicable **.ecc** script, see:

- Usage Request Sample Programs
- Update Request Sample Programs
- Policy Request Sample Programs
- Query Request Sample Programs
- Extension Implementation Sample Programs
- Notification Sample Programs
- Custom Plug-In Sample Programs

To determine which parameter values you must use for running a sample program, you can use the sample script's help option. For descriptions of the methods the sample programs use, see **oracle.communication.brm.charging.sdk** in *Elastic Charging Engine Java API Reference*.

Usage Request Sample Programs

Table 24-2 lists the usage sample programs, their descriptions, the shell scripts used to compile and run them, and the applicable **.ecc** script.

Table 24-2 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 "Example of 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 24-2 (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 subrequests 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 a reauthorization request (RAR) message. 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**	Enables you to send custom voice usage requests. Customer ID, number of requests to send, request type (INITIATE/UPDATE...) and duration must be given as arguments (e.g. 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.

Update Request Sample Programs

Table 24-3 lists the update sample program.

Table 24-3 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.

Policy Request Sample Programs

Table 24-4 lists the policy sample programs.

Table 24-4 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 configured for a customer's service.

Query Request Sample Programs

Table 24-5 lists the query sample programs.

Table 24-5 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.

Extension Implementation Sample Programs

Table 24-6 lists the extension sample programs.

Table 24-6 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.

Notification Sample Programs

Table 24-7 lists the notifications sample programs.

Table 24-7 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.

Custom Plug-In Sample Programs

Table 24-8 lists the custom plug-in sample programs.

Table 24-8 ECE Sample Programs for Custom Plug-In

Sample Program	ECC Script	Shell Script	Description
SampleRatedEventFormatterCustomPlugin	-	**build_deploy_plugin.sh**	Writes rated events into CDR records.
SampleRatedEventFormatterKafkaCustomPlugin	-	**build_deploy_plugin.sh**	Writes fully rated events into a JSON file that is published to Kafka topics.

Compiling and Running the Sample Programs

You compile and run a sample program with the shell script provided for that sample program.

To compile and run a sample program:

1. Open the *ECE_home*/**config/eceTopology.conf** file.
2. Uncomment the line where the **sdkCustomerLoader** node is defined.

   You are required to uncomment this line to be able to run the SDK sample programs.

> **Caution:**
>
> Do *not* run the **customerLoader** utility without the **-incremental** parameter in a production environment.

3. Go to the ECE server **bin** directory:

   cd *ECE_home*/**bin**

4. Load the ECE runtime environment:

   ./ecc 'load sdk_production_loader.ecc'

5. Go to the ECE SDK **bin** subdirectory that contains the shell script for compiling and running the sample program you want to run:

   cd *ECE_home*/**ocecesdk/bin/***sample_program_directory*

   where *sample_program_directory* is **extensions**, **notification**, **plugin**, **policy**, **query**, **update**, or **usage**. See Table 24-1 for more information.

   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. You can also run the command with no parameters to use 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

Example of SampleDebitRefundSession

The following shows an example of how to run the SampleDebitRefundSession sample program.

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 can supply the parameters in the command line or provide no parameters to run the 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.
- The 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.
- 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 Included Minutes.
- DEBIT_UNIT or REFUND_UNIT, TOTAL, IN, and OUT must be specified with numbers in MB (megabytes).

For example, the following command debits 10 USD and 25 Included Minutes:

```
sample_debit_refund_session.sh run 650999777 DEBIT_AMOUNT CORR_ID 840,10 95,25
```

This example command refunds 50 USD and 5 Included Minutes:

```
sample_debit_refund_session.sh run 650999777 REFUND_AMOUNT CORR_ID 840,50 95,5
```

Compiling and Deploying SampleRatedEventFormatterCustomPlugin

To compile and deploy the **SampleRatedEventFormatterCustomPlugin** sample program:

1. Configure Rated Event Formatter by doing the following:
   a. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
   b. Expand the **ECE Configuration** node.
   c. Expand **charging.ratedEventFormatters.***Instance_Name*, where *Instance_Name* is the name of the instance you want to configure such as ratedEventFormatter1.
   d. Expand **Attributes**.
   e. Set the **pluginType** attribute to **oracle.communication.brm.charging.sdk.plugin.SampleRatedEventFormatterKafkaCustomPlugin**.
2. Go to the *ECE_home***/ocecesdk/bin/plugin** directory and then build the sample program:

   ```
   sh ./build_deploy_plugin.sh build SampleRatedEventFormatterKafkaCustomPlugin
   ```
3. Deploy the sample program:

   ```
   sh ./build_deploy_plugin.sh deploy SampleRatedEventFormatterKafkaCustomPlugin
   ```
4. Ensure that your Kafka Server is up and running.
5. Create a topic in your Kafka Server for the rated events.

   For example, this command creates a Kafka topic named **RatedEvent**:

   ```
   bin/kafka-topics.sh --create --bootstrap-server kafkaHost:port --replication-factor
   1 --partitions 1 --topic RatedEvents
   ```

where *kafkaHost* and *port* are the host and port that the Kafka client will connect to in a bootstrap Kafka cluster the first time it starts.

6. Wait until the entire configuration is ready and all components are up and running.
7. Perform usage, such as initiate, update, or terminate, for one of your customers.

   Rated events will be published to the Kafka topic in JSON format. Also, JSON information will be added to the **ratedEventFormatter1.log** file under the *ECE_home***/log** directory.
8. Confirm that the event was published to the Kafka topic by running the Kafka console consumer:

   ```
   bin/kafka-console-consumer.sh --bootstrap-server kafkaHost:port --topic RatedEvents
   ```
9. When you are done with the sample program, shut down the ECE runtime environment:

   ```
   ecc stop server
   ```

25

Testing ECE

You learn how to test your implementation of Oracle Communications Elastic Charging Engine (ECE).

Topics in this document:

About ECE Testing Utilities

ECE offers the testing utilities in Table 25-1 that you can use when implementing ECE in a charging system.

Table 25-1 ECE Testing Utilities

Utility	Description
Simulator	Emulates the role of a client application sending requests to ECE. The simulator enables 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. See "Running the Simulator to Send Usage Requests".
query	Enables you to run queries on ECE data for development or debugging purposes. See "Using the query Utility to Test ECE". **Note:** 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** utility 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.

Table 25-1 (Cont.) ECE Testing Utilities

Utility	Description
customerGenerator	Creates XML files that represent sample customer data, which can then be loaded into ECE. **Note:** The ECE customer XML data file must conform to the format of the ECE customer XML schema file (*ECE_home***/odi_transformation/ECE_Schema.xsd**).
customerLoader	Loads customer data from BRM incrementally, in batches or in bulk. In addition to using this utility in a development system, you can use it in a production system to correct data migration errors. To do so, run the utility with the -incremental parameter. Caution! Do not run the customerLoader utility without the -incremental parameter in a production environment.
PerformanceMonitor MBean	Monitors the performance of your ECE deployment during testing. See "About Performance MBean".

About Loading Sample Data

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

- Sample data for integrating with BRM
- Sample data for integrating with PDC
- Sample data for integrating with clients that send policy requests (used for policy testing)

Sample data includes sample event definitions, 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.

About 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 nodes, such as the simulator, 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:

- **startTrackingCPU()**

  The startTrackingCPU() method starts tracking CPU usage for the running process.

- **stopTrackingCPU()**

  Use the stopTrackingCPU() method to stop 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.

- **getTrackedCPU()**

  Use the getTrackedCPU() method to get 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 MBean 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.

> **Note:**
>
> 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 configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE configuration** node.
3. Expand **charging.server**.
4. Expand **Attributes**.
5. 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.

Using the query Utility to Test ECE

The **query** utility provides access to ECE cache content, enabling you to run queries on ECE Coherence caches. The **query** utility is meant to be used for debugging purposes only.

You can use the **query** utility to write scripts that interact with the ECE domain objects, creating CohQL queries. The **query** utility supports all ECE caches and objects.

> **Note:**
>
> Oracle recommends that you not query the cache for the entire list of customer information. Filtered/ selected query is recommended because the complete customer list would cause ECE to crash.

The **query** utility is included with the ECE Server software in *ECE_home***/bin**.

To learn about **query** utility options, use the help command:

```
$ ./query.sh -h
```

The following shows non-interactive use of the **query** utility:

```
$ ./query.sh -s -c -l "select sum(getAvailableBalance(\'USD\',null).getQuantity()) from
Balance"
```

The following shows interactive use of the **query** utility:

```
$ ./query.sh
Coherence Command Line Tool
CohQl> select count () from Customer;
Results
1000
CohQL> select key(), value().getCode().toString() from BalanceElement
Results
840, "USD"
```

The **query** utility log file is *ECE_home***/logs/query_out.log**.

The query statement history is contained in *ECE_home***/bin/.cohql-history**. You can use the up and down arrows to move through the command history.

> **Note:**
>
> Oracle does not recommend that client applications directly use the Coherence API or the **query** utility 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.

Example: 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"
```

Example: Query a Customer Balance

Following is an example of how to query a customer's balance. You first 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}]),]
```

Verifying that Usage Requests Can Be Processed

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** utility to verify that the usage has impacted the customer balance.

> **Note:**
>
> If you installed online or offline network mediation software, you can use that software instead of the ECE simulator to send usage requests for online or offline charging. 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.

To verify that usage requests can be processed, perform the steps described in these sections:

- 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 nodes in the cluster:

1. Log on to the driver machine.
2. Go to the ECE_home**/bin** directory:
3. Start the Elastic Charging Controller:

   ```
   ./ecc
   ```

4. Start the ECE nodes:

   ```
   start
   ```

To verify that the ECE nodes are running:

1. Access the ECE configuration MBeans in a JMX editor, such as JConsole. See "Accessing ECE Configuration MBeans".
2. Expand the **ECE Configuration** node.
3. Expand **ECE State Machine**.
4. Expand **StateManager**.
5. Expand **Attributes**.
6. 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 takes a few seconds to finish processing the workload.

4. Open the **invocation.log** file located in *ECE_home***/logs**. 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** utility.

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 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 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***/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.AsynchronousN
otificationPublisher</param-value>
```

with this:

```
<init-param>
  <param-name>cache-store</param-name>
    <param-
value>oracle.communication.brm.charging.util.coherence.internal.NoPersistenceCacheSto
re</param-value>
```

ECE external notifications are disabled.

3. Save the file.

Verifying that Friends and Family Calls Are Processed

To verify that your ECE deployment is processing friends and family calls, perform prerequisite tasks in BRM and PDC and then generate usage for the friends and family call for the customer.

To verify that friends and family calls are processed:

1. Ensure the appropriate provisioning tag is available in BRM as follows:
   a. Ensure you define a provisioning tag that includes the **Friends&Family** extended rating attribute (ERA).
   b. Ensure the provisioning tag in BRM contains the same profile specification labels provided in PDC.

      The profile specification labels that come ready-to-use in the PDC installation are **MYFRIENDS** and **MYFAMILY**. Specify these labels in the provisioning tag when using the ready-to-use profile specification labels in PDC.
   c. (Optional) If you create a new provisioning tag in BRM, rather than using the ready-to-use 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 friends and family into PDC.

   The sample XML file is available at *PDC_home***/apps/Samples/Examples/PDCSampleProfileSpec.xml** where *PDC_home* is the directory in which you installed PDC.

   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/PDCSampleCAR.xml**.

   PDCSampleCAR.xml contains two custom rules: **Friends&Family** 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 as follows:
   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 run time, ECE uses this result in the charge to apply different rates for calls to friends and family.

5. Verify that the ECE Pricing Updater is started.
6. Publish the PDC pricing data to ECE.

   The Pricing Updater synchronizes the pricing data to ECE.
7. Verify that 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.
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 ready-to-use 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).
   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.

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:

> **Note:**
>
> 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**.

   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.

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 Are Correctly Formed

To debug rating errors, you may need to verify that payloads in usage requests are correctly formed. You can view payloads 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, 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 event definition in ECE in the RequestSpecification cache accordingly.

Part VII
ECE Utilities

This part provides information about the utilities provided with Oracle Communications Elastic Charging Engine (ECE). It contains the following chapters:

- Charging Utilities

26
Charging Utilities

You learn about syntax and parameters used by the Oracle Communications Elastic Charging Engine (ECE) charging utilities.

Topics in this document:

query

Use the **query** utility to launch the Oracle Coherence query client and run queries against the ECE Coherence cache. You can use the utility to write scripts that interact with the ECE domain objects, creating CohQL queries. The utility supports all ECE caches and objects. For more information, see "Using the query Utility to Test ECE".

You can run this utility in interactive mode or non-interactive mode.

Location

ECE_home/**bin**

Syntax: Interactive Mode

```
./query.sh
Coherence Command Line Tool
CohQl>
```

Syntax: Non-Interactive Mode

```
./query.sh [-t] [-c] [-s] [-e] [-l "statement"] [-f filename] [-h]
           [-P "CohQLquery"]
```

Parameters: Non-Interactive Mode

-t
Enables trace mode for printing debugging information.

-c
Specifies to exit the utility after processing all command-line arguments.

> **Note:**
>
> Do not use this argument when redirecting from standard input, because the utility exits as soon as the command-line arguments have been processed and the redirected input will never be read.

-s
Runs the utility in silent mode. This allows the utility to be used in pipes or filters by redirecting to standard input and standard output.

-e
Runs the utility in extended language mode. This mode allows object literals in update and insert commands.

-l "*statement*"
Runs the specified CohQL statement. Enclose your CohQL statements in single or double quotes.
For more information about CohQL statements, see "Using Coherence Query Language" in *Oracle Fusion Middleware Developing Applications with Oracle Coherence*.

-f *filename*
Processes the CohQL statements in the specified file. The statements in the file must be separated by a semicolon (**;**). The file is an operating system-dependent path and must be enclosed in single or double quotes.

-P "*CohQLquery*"
Runs the specified CohQL SELECT query against the database. You can run the **select Count()** and **select Value()** queries. If you include a WHERE clause, it supports the condition **Customer_id = '***value***'**. For example:

```
./query.sh -P "select value() from Customer where Customer_id = 'Cust#6500000242'"
```

For more information about CohQL queries, see "Using Coherence Query Language" in *Oracle Fusion Middleware Developing Applications with Oracle Coherence*.

-h
Displays the syntax for this utility.

Results

Look in the *ECE_home***/logs/query_out.log** file for errors.

> **Note:**
>
> Oracle recommends that you not query the cache for the entire list of customer information. Filtered/ selected query is recommended because the complete customer list would cause ECE to crash.

A

Sample Notification Payloads

You can use the sample notification payloads to view the format of notifications that Oracle Communications Elastic Charging Engine (ECE) publishes into the JSM notification queue.

Topics in this document:

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>
```

Billing 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>
```

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>
```

Custom Notification for BRM Gateway

The payload published for a custom notification that is targeted for BRM Gateway uses the following format:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Notification version="3.0.0.0.0">
<CustomNotification>
   <NotificationType>CUSTOM_EVENT_NOTIFICATION_EVENT</NotificationType>
   <OpCode>CUST_MODIFY_CUSTOMER<OpCode>
   <CustomDataMap>
      <CustomDataKey>CustomerId</CustomDataKey>
      <CustomDataValue>123</CustomDataValue>
   </CustomDataMap>
   <CustomDataMap>
      <CustomDataKey>BalanceId</CustomDataKey>
      <CustomDataValue>456</CustomDataValue>
   </CustomDataMap>
   <CustomDataMap>
      <CustomDataKey>CustomerId</CustomDataKey>
      <CustomDataValue>123</CustomDataValue>
   </CustomDataMap>
   <CustomDataMap>
      <CustomDataKey>BalanceId</CustomDataKey>
      <CustomDataValue>456</CustomDataValue>
   </CustomDataMap>
</Notification>
```

External Top-up 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

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>
```

Life-Cycle Transition

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--153892112-
slc06bui:TelcoGsmTelephony, 0049100120:TelcoGsmTelephony">
         <SubscriberPreferencesInfo>
           <PreferenceName>Language</PreferenceName>
```

```
            <PreferenceValue>French</PreferenceValue>
          </SubscriberPreferencesInfo>
                </SubscriberPreference>
      </SubscriberPreferences>
  </LifeCycleTransitionNotification>
</Notification>
```

Replenish POID ID 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

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>
```

Subscriber Preference 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>
```

Threshold Breach Event (Breach Direction Down)

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 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>
    <BalanceElementId>840</BalanceElementId>
    <BalanceElementCode>USD</BalanceElementCode>
    <CurrentBalance>-4.00</CurrentBalance>
    <ThresholdAmount>-4.5</ThresholdAmount>
    <ThresholdPercent>55.0</ThresholdPercent>
    <BreachDirection>THRESHOLD_BREACH_UP</BreachDirection>
    </CreditThresholdBreachNotification>
</Notification>
```

Top-up 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>
    </RARNotification>
</Notification>
```

Enriched Notification

The payload published for a custom enriched notification with the customDataMap populated with attributes like original balance, current balance, event timestamp, etc:

```
[ThresholdBreachServiceEventImpl{ balanceId='BALANCE', beCode='USD', beId=840,
      currentBalance=-179.00, thresholdAmount=-180, publicUserIdentities='[6500000001]',
      thresholdPercentage=10.0, breachDirection='THRESHOLD_BREACH_UP', alertType='2',
reason='0x01',
      validityUpdateOnly='false', customerId='CUSTOMER',
eventTimeStamp='2024-01-17T08:50:17.096Z',
      operationType='USAGE',
      subOperationType='UPDATE'[ExternalNotifiableServiceEventImpl{sessionId='null',
      clientRoutingData=null[ThresholdBreachServiceEventImpl{serviceEventId=null,
      serviceEventType=THRESHOLD_BREACH_EVENT, notificationVersion=7.0.0.0.0,
      customDataMap={OriginalBalance=UnitValue{quantity=0, unit=Occ},
CalledNumber=650000002,
      NotificationType=THRESHOLD_BREACH_EVENT, CurrentBalance=UnitValue{quantity=-50.0,
unit=Occ},
      EventTimeStamp=2024-01-17T08:50:17.096Z,
ExtensionThresholdKey1=ExtensionThresholdValue1,
      CustomerId=CUSTOMER, SessionId=SESSION ID, BalanceId=BALANCE,
CallingNumber=6500000001},
      puidPrefMap={}}]'}]'}]
```

B

Specifications and Standards Compliance in ECE

Learn about the specifications and standards used in Oracle Communications Elastic Charging Engine (ECE).

Topics in this document:

About Specifications and Standards Compliance

The ECE charging API aligns with the Remote Authentication Dial In User Service (RADIUS) Accounting Request for Comments (RFC) specifications and with the standards described in the 3rd Generation Partnership Project (3GPP) Technical Specifications (TS). ECE charging supports any 3GPP sub-domain; some are listed here as examples:

- PS (Packet Switched) connections
- CS (Circuit Switched) connections
- WLAN (Wireless Local Area Network)
- IMS (IP-Multimedia Subsystem)
- PCRF (Policy and Charging Rules Function) and Sy/Sp (Sh) interfaces

The ECE charging API is extensible; it can accommodate proprietary extensions of the standards.

The ECE Java API aligns with the Diameter Ro, Diameter CCA, Diameter Rf, and RADIUS message formats. Network mediation software programs (client applications) that support these protocols can send usage requests to ECE.

The following 3GPP Technical Specifications (TS) relate to ECE charging functionality.

- "3GPP TS 32.240 Telecommunication management; Charging management; Charging architecture and principles"

  For online charging, ECE exposes a Java API based on Diameter Ro, which is extensible for supporting any extension or variation.

  ECE implements the following functionality for online charging:

  - Online Charging Function modules:
    * Session Based Charging Function (SBCF)
    * Event Based Charging Function (EBCF)
  - Rating Function (RF)
  - Account Balance Management Function (ABMF)
- "3GPP TS 32.260 Telecommunication management; Charging management; IP Multimedia Subsystem (IMS) charging"

- "3GPP TS 32.290 Telecommunication management; Charging management; 5G system; Services, operations and procedures of charging using Service Based Interface (SBI)
- "3GPP TS 32.299 Telecommunication management; Charging management; Diameter charging applications"

  For offline charging, ECE exposes a Java API based on DIAMETER Rf, which can be called from the offline mediation system. The Java interface has functionality close to that of the Rf interface described in 3GPP 32.299 and is extensible for supporting any extension or variation.

  Oracle Communications Offline Mediation Controller uses this interface to load CDRs into ECE for charging.
- GB922 TM Forum Information Framework (SID).

The following RADIUS RFCs relate to ECE charging functionality.

- RFC 2865, "Remote Authentication Dial In User Service (RADIUS)," June 2000, RADIUS. Updated by: RFC 2868, RFC 3575, RFC 5080.
- RFC 2866, RFC 2867, RFC 2868, RFC 2869, RFC 3579

ECE aligns with the Diameter Credit-Control Application charging functionality described in Internet Engineering Task Force (IETF) Network Working Group RFC 4006.

The following 3GPP Technical Specifications (TS) relate to the Policy and Charging Rules Function (PCRF) and ECE:

- "3GPP TS 29.219 Policy and charging control: Spending limit reporting over Sy reference point"

  The Sy interface is located between the PCRF and Online Network Mediation Controller. It enables the transfer of customer spending information.
- "3GPP TS 29.329 Sh interface based on the Diameter protocol"

  The Sp (implemented as Sh) interface is located between the SPR (Subscription Profile Repository) and PCRF. It enables the retrieval of customer identities and profile information.