Oracle® Communications Billing and Revenue Management

Elastic Charging Engine 11.3 Extensions

Release 7.5

E70771-03

December 2016

This document describes how to customize the Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE) functionality through custom extension points.

This document is for developers.

Introduction

Use the ECE extensions to customize Diameter Gateway, RADIUS Gateway, pre-rating, rating, post-rating, and post-charging processes. ECE extensions include sample implementations that guide you in implementing your custom business logic.

Extension Points

The following sections describe the extension points to customize Diameter Gateway, RADIUS Gateway, pre-rating, rating, post-rating, and post-charging processes:

- Diameter-Request Processing Extension Points
- RADIUS-Request Processing Extension Points
- Usage-Request Processing Extension Points

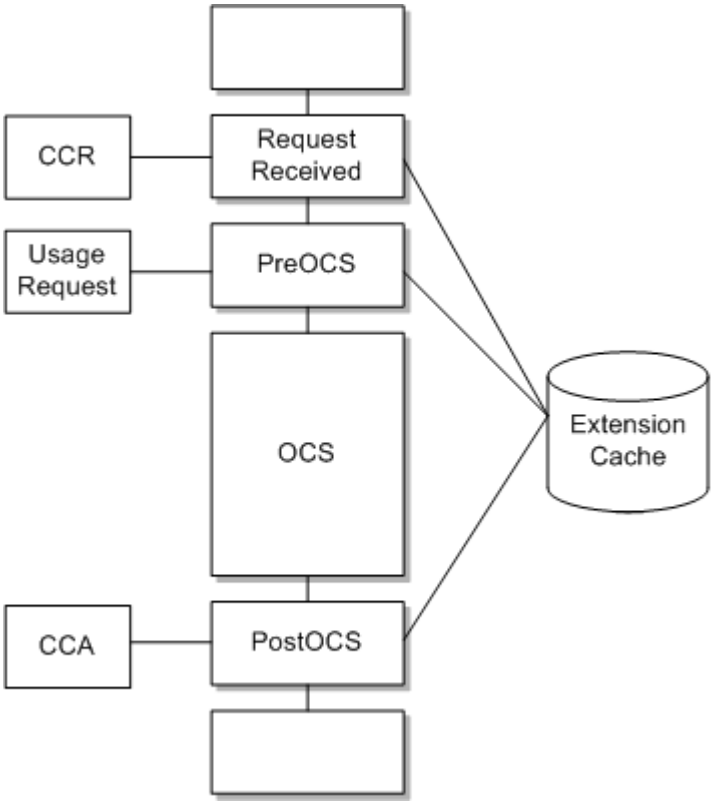
Diameter-Request Processing Extension Points

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.
- **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 CCA AVPs before the diameter response is returned to the diameter client.

Figure 1 shows the diameter-request processing extension points.

Figure 1 Diameter-Request 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 2 shows the RADIUS-request processing extension points for EAP authentication.

Figure 2 Extension Points for EAP Authentication

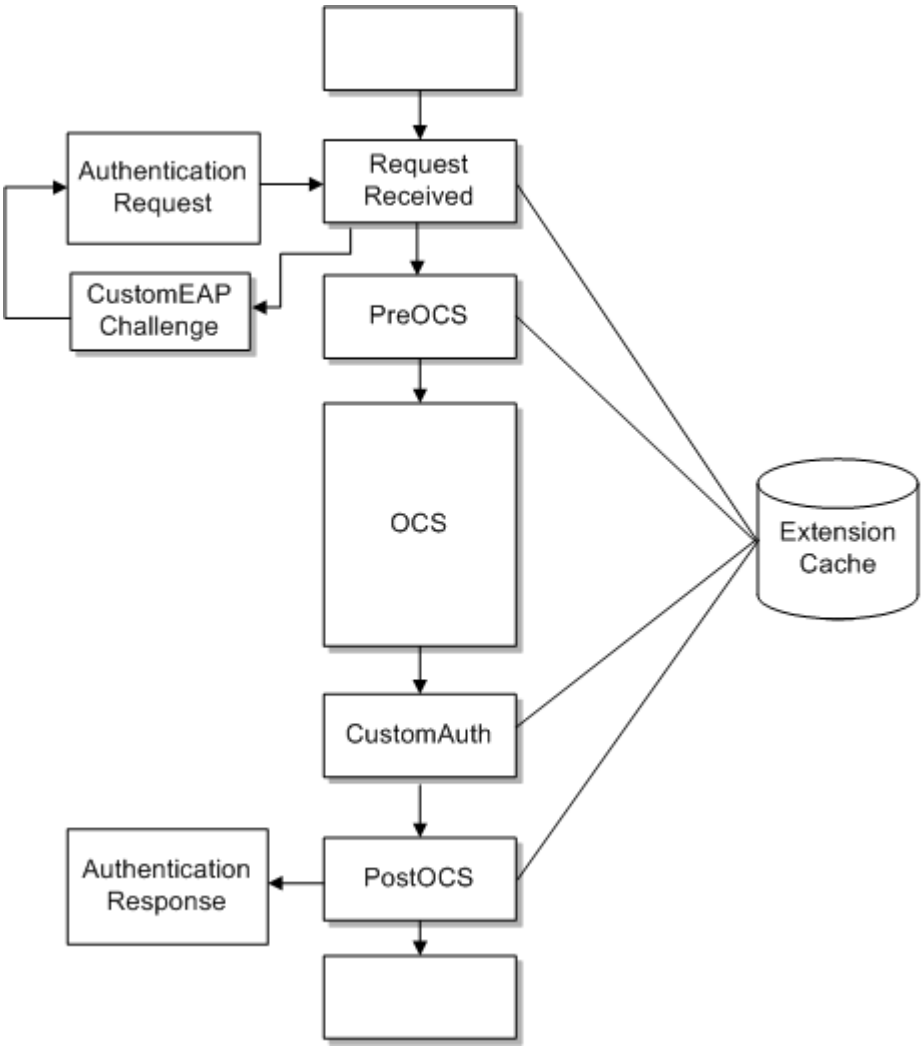
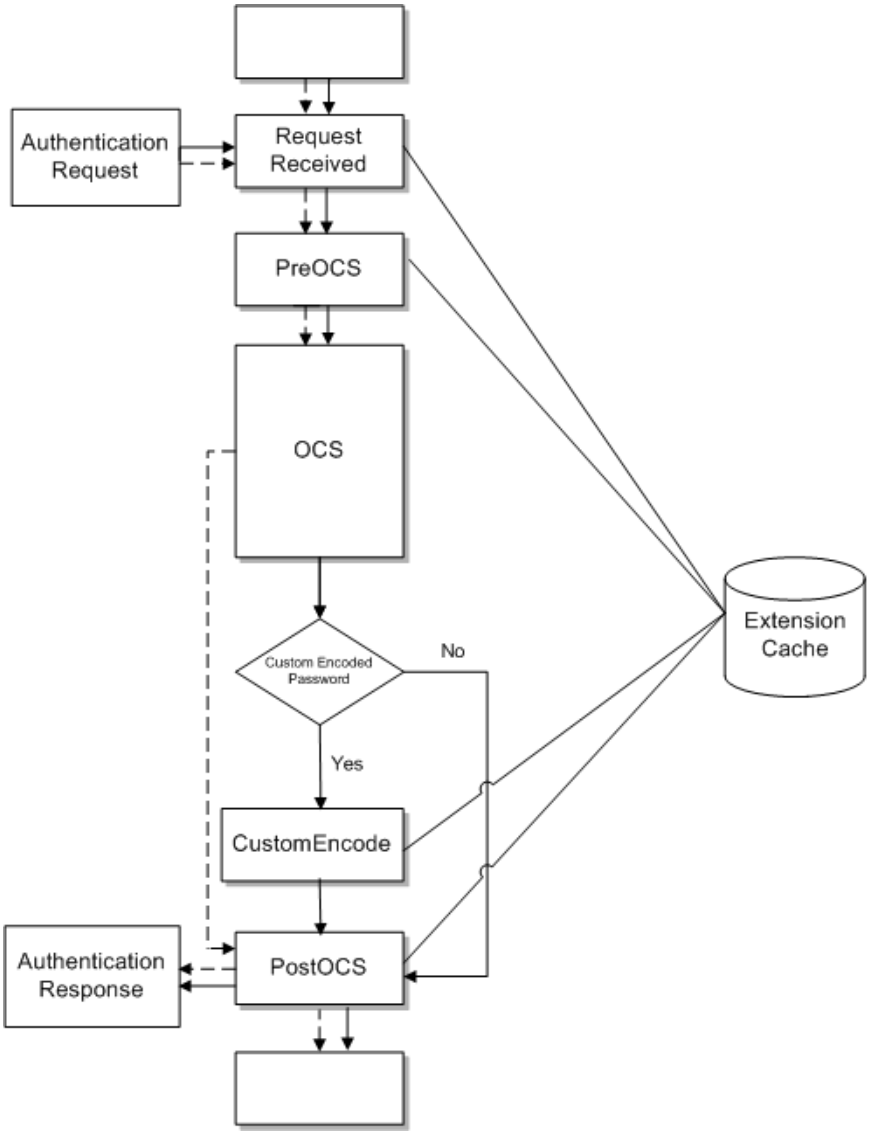


Figure 3 shows the RADIUS-request processing extension points for PAP and Challenge-Handshake Authentication Protocol (CHAP) authentication. The solid line depicts PAP authentication and the dotted line depicts CHAP authentication in this figure.

Figure 3 Extension Points for PAP and CHAP Authentication



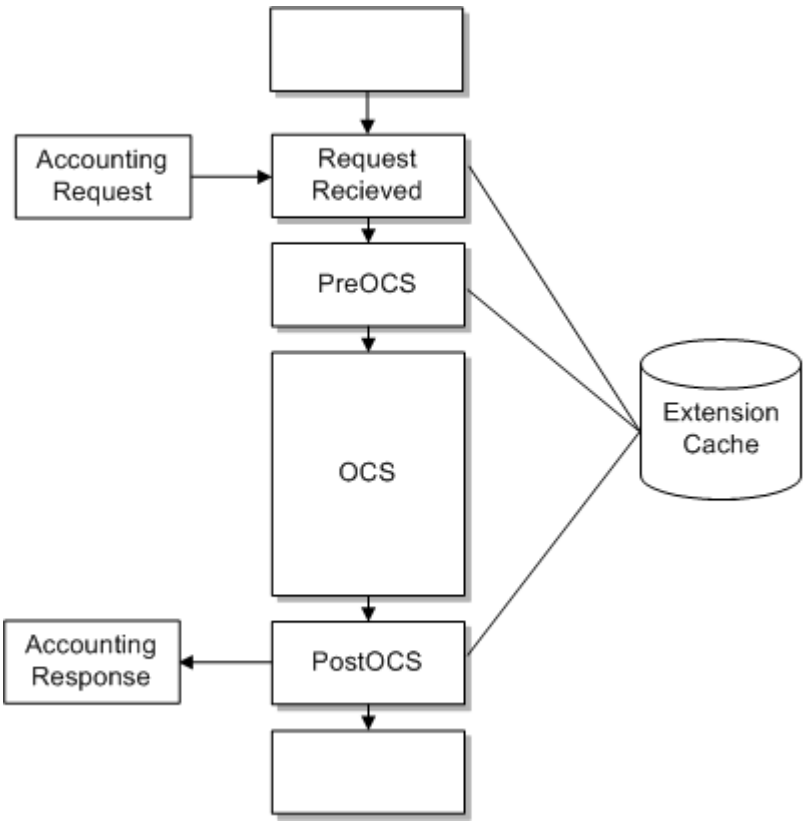
Accounting Extension Points

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 4 shows the RADIUS-request processing extension points for accounting.

Figure 4 Extension Points for Accounting



Usage-Request Processing Extension Points

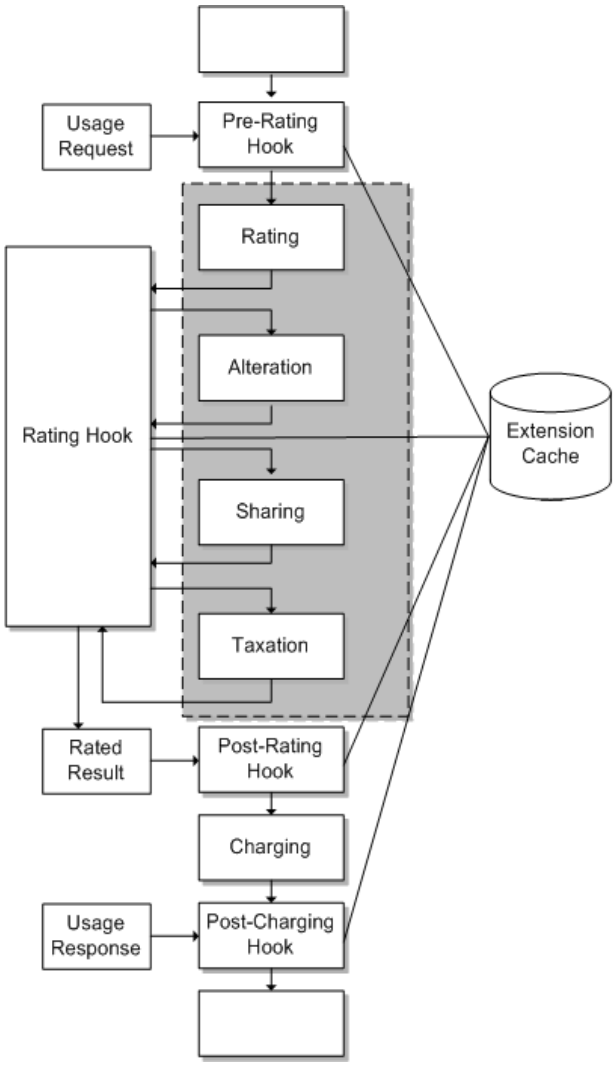
ECE provides extension points in the rating flow: before charge calculation, after charge calculation (prior to applying a balance impact), and after charging (after applying a balance impact):

- **Pre-rating extension.** The role of the pre-rating extension is to alter the usage request.
- **Rating extension.** The role of the rating extension is to alter rated results after each of the following processes: rating, alteration, sharing, and taxation.
- **Post-rating extension.** The role of the post-rating extension is to alter final rated results (after rating, alteration, sharing, taxation, and item assignment) and add new tax rating impacts.
- **Post-charging extension.** The role of the post-charging extension is to enrich the usage response and notification.

You cannot customize rating during the rating, alteration, sharing, and taxation 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, balance data, and ExtensionServiceContext data.

Figure 5 shows the usage-request processing extension points (called *hooks* in the figure).

Figure 5 Usage-Request Processing Extension Points



Implementing the Extensions Logic

The **GyExtension**, **PreRatingExtension**, **RatingExtension**, **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, the **GyExtension** interface is exposed to the extension points through the **ExtensionContext** methods.

For diameter-request processing extension points, the following methods are called by the charging flow:

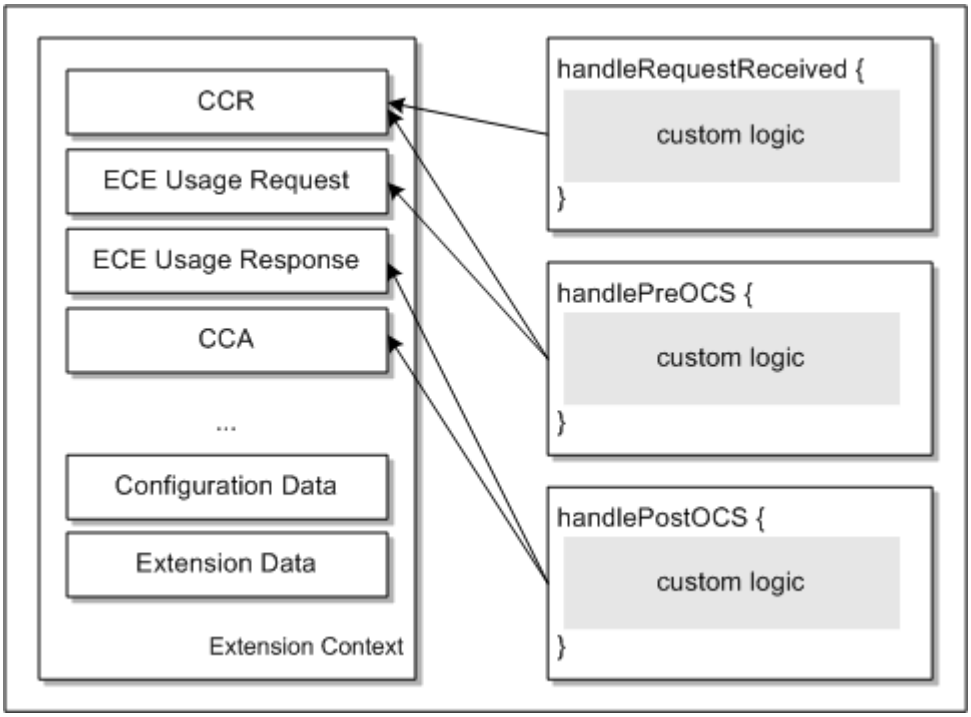
- **handleRequestReceived().** Called for every CCR that is processed by the charging flow.

- **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 the methods expose relevant ExtensionContext data for accessing the ExtensionsDataRepository, AppConfigRepository, and other extensions-related contexts.

Figure 6 shows the data used in the diameter-request processing extension points.

Figure 6 Data Used in Diameter-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.

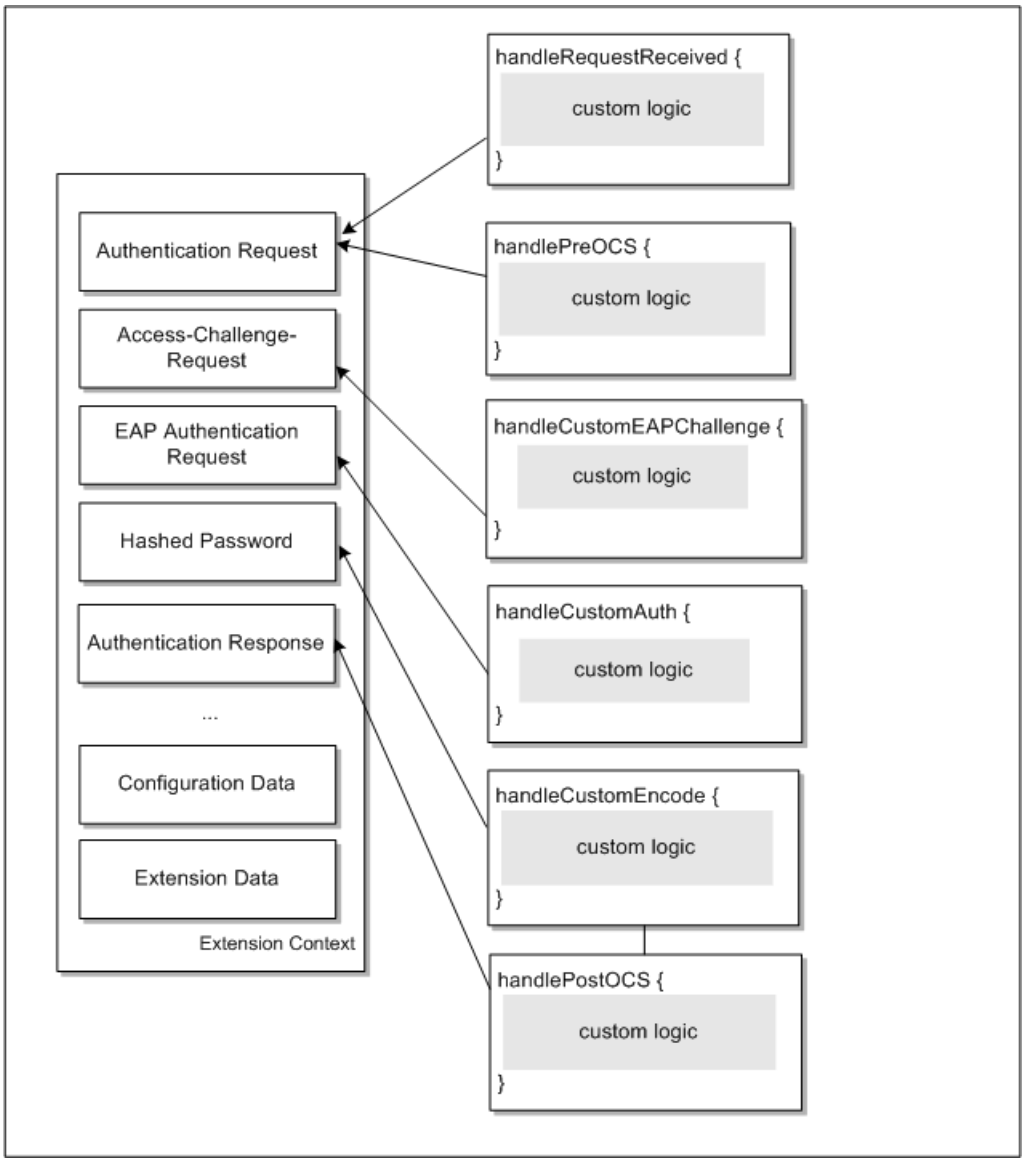
For authentication-related extension points, the following methods are called by the authentication flow:

- **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 7 shows the data used in the RADIUS-request processing extension points for authentication.

Figure 7 Data Used in RADIUS-Request Processing Extension Points for Authentication

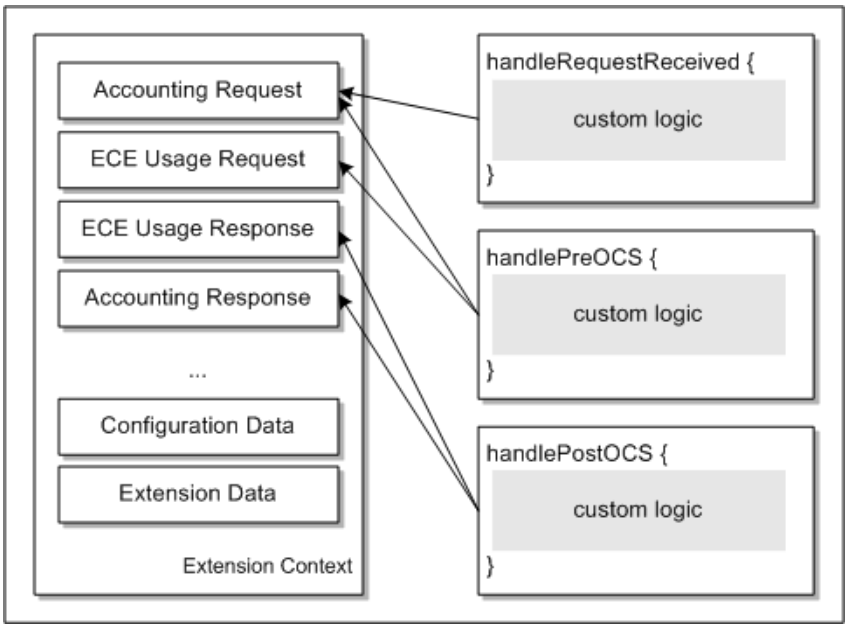


For accounting-related extension points, the following methods are called by the accounting flow:

- **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 8 shows the data used in the RADIUS-request processing extension points for accounting.

Figure 8 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 9 shows the data used in the pre-rating extension point.

Figure 9 Data Used in Pre-Rating Extension Point

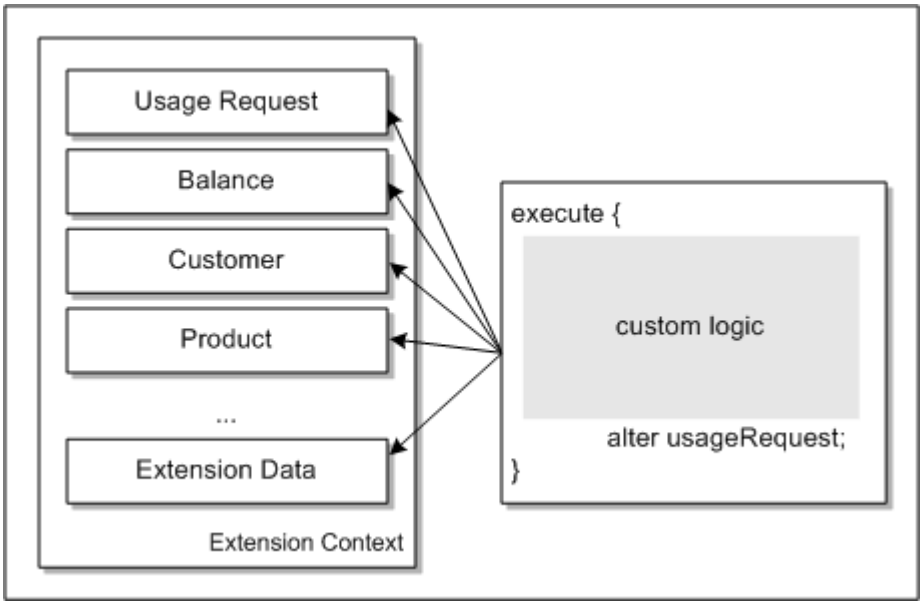


Figure 10 shows the data used in the rating extension point.

Figure 10 Data Used in Rating Extension Point

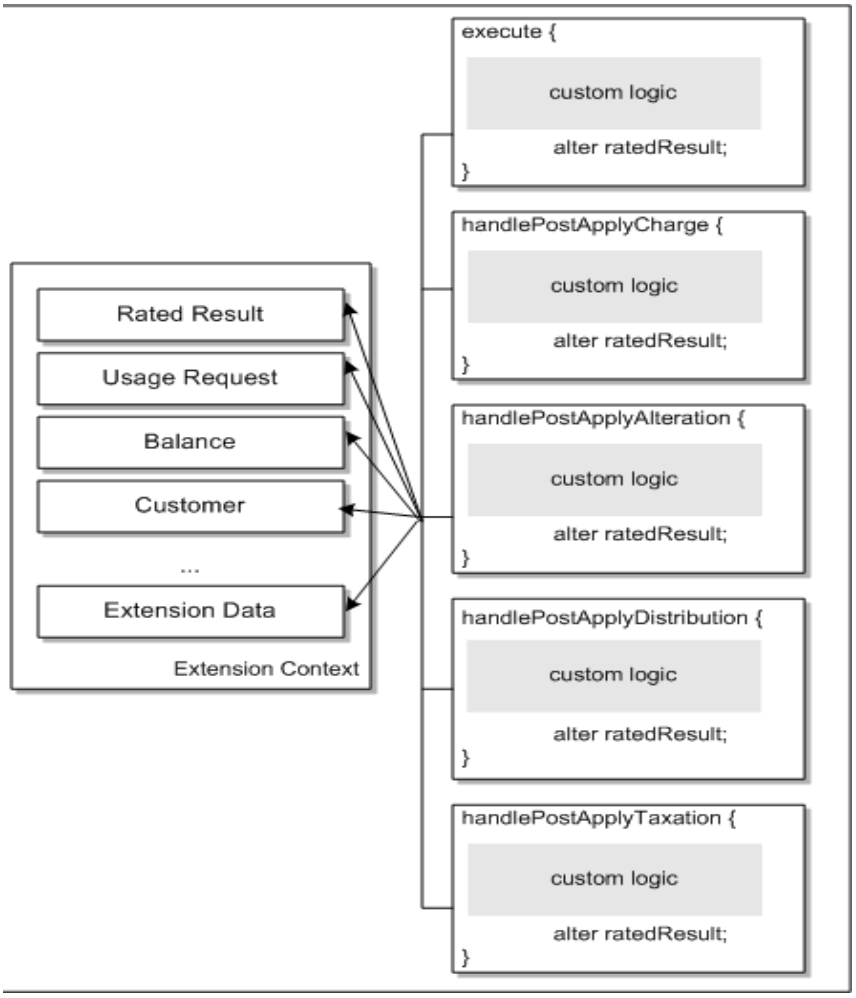


Figure 11 shows the data used in the post-rating extension point.

Figure 11 Data Used in Post-Rating Extension Point

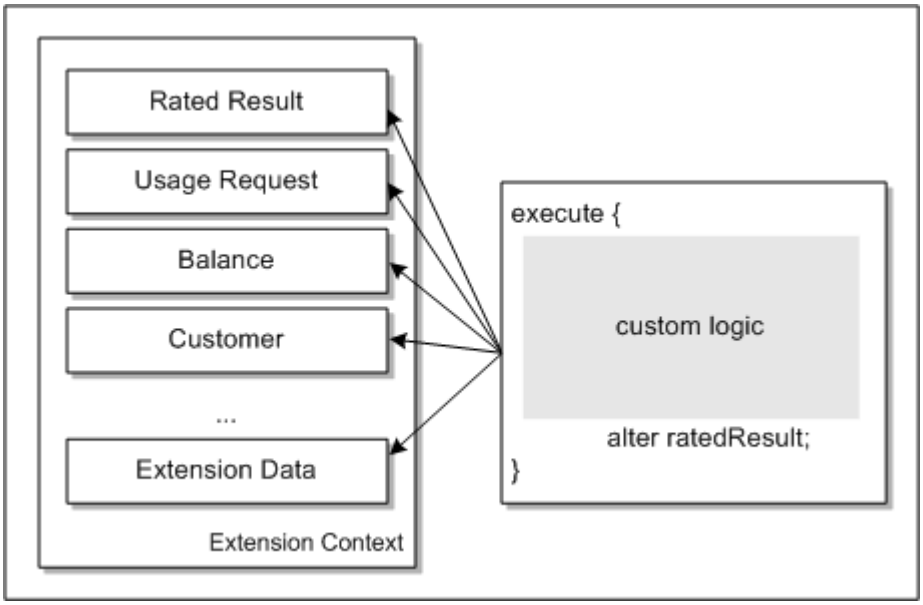
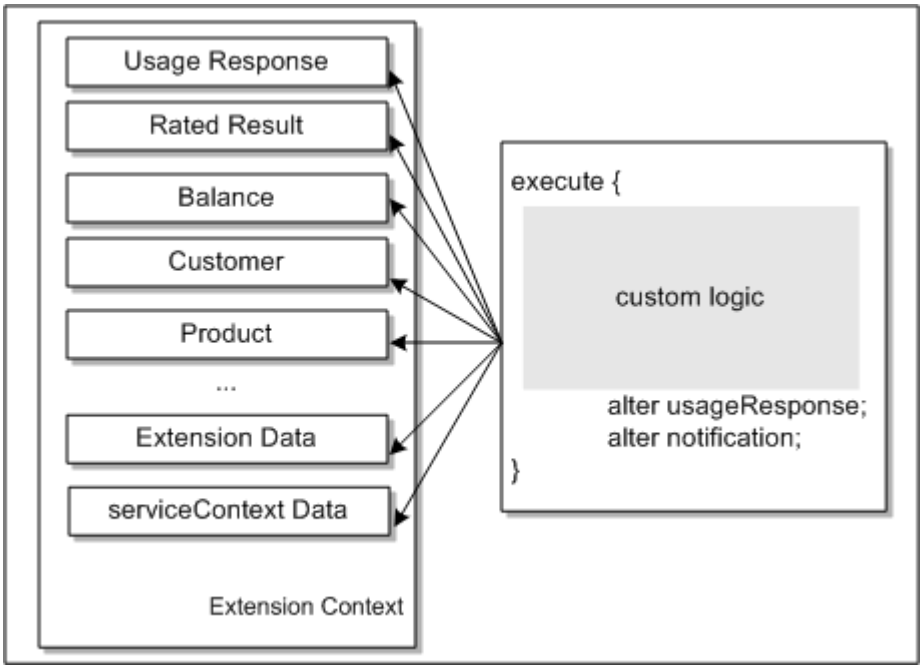


Figure 12 shows the data used in the post-charging extension point.

Figure 12 Data Used in Post-Charging Extension Point



For more information, see the ECE Extension Javadocs in *BRM Elastic Charging Engine Java API Reference*.

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). For more information, see the discussion about the ECC **sync** command in *ECE System Administrator's Guide*.

Custom extensions logic implementation classes that implement the **GyExtension**, **PreRatingExtension**, **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.

RequestReceived Extension

The RequestReceived extension manipulates the CCR, authentication, or accounting request so that the CCR, authentication, or accounting 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.

Accessible Data

- Credit Control Request
- Authentication Request
- Accounting Request
- System configuration
- Extensions data

Modifiable 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.

CustomAuth Extension

The CustomAuth extension implements custom EAP authentication methods; for example, EAP-POTP, EAP-PSK etc.

Accessible Data

- EAP-Authentication-Request
- System configuration

Modifiable Data

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.

Accessible Data

- Access-Challenge-Request
- System configuration
- Extensions data

Modifiable Data

You use the extension point to send the custom access-challenge request to the RADIUS client when EAP is used for authentication.

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. And, the PreOCS extension performs any actions related to authentication that are required before the RADIUS request is sent to ECE. This extension is called before any rating, discounting, or alteration logic has been invoked.

Accessible Data

- Credit Control Request
- Authentication Request
- Accounting Request
- ECE Usage Request
- System configuration
- Extensions data

Modifiable 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.

PostOCS Extension

The PostOCS extension manipulates CCA, accounting, or authentication responses to match the business requirement before returning the CCA, accounting, or authentication responses to the diameter client or the RADIUS client. This extension is called after charging, authentication, and accounting has been completed and recorded.

Accessible Data

- Credit Control Request
- Accounting Response
- Authentication Response
- ECE Usage Response
- Diameter Credit Control Answer
- System configuration
- Extensions data

Modifiable 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.

CustomEncode Extension

The CustomEncode extension implements the custom hashing algorithm that is used on passwords for authentication.

Accessible Data

- Encoded Password
- System configuration
- Extensions data

Modifiable 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.

Pre-Rating Extension

The pre-rating extension enhances the usage request based on the customer, product, and balance 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.

Accessible Data

- ECE Usage Request
- Calling and called customer (including profile)
- Product (including profile)
- Balance information
- System configuration
- Extensions

Modifiable 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.
- You can modify the values of the pricing attributes with custom logic. This enables you to override a product price.

Rating Extension

The rating extension modifies the rated results after each of the following processes: rating, alteration, sharing, and taxation.

Accessible Data

- Customer (including profile)
- Shared customer (if part of a sharing relationship)
- Product (including profile)
- Balance information
- System configuration
- Extensions
- Rated result

Modifiable Data

- 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.

Post-Rating Extension

The post-rating extension modifies final rated results (after rating, alteration, sharing, taxation, and item assignment) and creates new tax rating impacts.

Accessible Data

- Customer (including profiles)
- Shared customers (if part of a sharing relationship)
- Product (including profiles)
- Balance information
- System configuration
- Extensions
- Rated result

Modifiable Data

- Modify the balance impact amount, GL code, tax code, balance element or invoice data for rating impacts generated from ECE; for example, charge, alteration, or distribution rating impacts.
- Create new tax rating impacts; for example, implement tax on tax.

Post-Charging Extension

The post-charging extension enriches the usage response, Diameter notification, and credit threshold and credit breach notifications. This extension is called after charging is completed but before the usage response is generated.

Accessible Data

- Customer (including profiles and subscriber preferences)
- Shared customers (if part of a sharing relationship, subscriber preferences)
- Product (including profiles, subscriber preferences, life cycle state)
- Balance information (including current request impacts)
- Business profiles
- System configuration
- Extensions
- ExtensionServiceContext
- Rated result

Modifiable Data

You can modify the usage response, Diameter notification, and credit threshold and credit breach notifications with custom logic. You can add custom data to the following:

- Usage responses and Diameter notifications. 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.
- Credit threshold and credit breach 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.

You can configure the post-charging extension to reject the current response without impacting balances.

Extensions Cache

The extensions framework provides a generic repository from which data required for the pre-rating, 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 1 is an example of a specifications file for the post-rating extension:

Example 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 2 shows the associated data to load into the cache using the specification file above:

Example 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.

Extensions 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

This section documents the sample extensions.

Diameter Gateway Extension - Service

The sample program **SampleDiameterGyExtension** shows how to use the immediate-response feature based on an incoming AVP value.

Logic:

If Service-Context-Id is OFFLINE:

Then respond with Diameter Code DIAMETER_REDIRECT_INDICATION and set the Redirect-Host AVP value

Pre-Rating Extension - Dynamic Quota Management and Retrieving Function Values

The sample program **SamplePreRatingExtension** shows pre-rating custom logic. It illustrates sample logic for the following pre-rating scenarios:

- Dynamic Quota Management
- Retrieving Function Values for Discount Expressions

Dynamic Quota Management

The **SamplePreRatingExtension** program shows how to modify the input request quantity based on input network type where the customer balance is greater than a predefined amount.

Logic:

If 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

Retrieving Function Values for Discount Expressions

The **SamplePreRatingExtension** program shows how 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.

Logic:

If the PDC rate plan specifies a 10% discount for all accounts active less than 12 months, then the logic is the following:

If **customerActiveMonths** value is:

< 12 then apply a discount of 10%

or

> 12 then apply a discount of 0%

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.

Logic:

1. After taxation, determine the custom item type to be used based on the data accessible through the rating extension.
2. Assign the rating impacts to the custom bill items based on the new custom item type.

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.

Logic:

1. Determine the federal tax rate from the tax configuration table using the tax code, request start time.
2. Calculate the federal tax based on this tax rate.
3. Modify the original impact in the tax rating period based on the taxable impact from the linked charge, alteration, or distribution rating period.
4. Determine the state tax rate from the tax configuration table using the tax code, request start time.
5. Calculate state tax based on this tax rate.
6. Create 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 saved into the CDR output file.

Post-Charging Extension - Adding Custom Data to Usage Responses and Notifications

The sample program **SamplePostChargingExtension** shows how to add custom data to the following:

- Usage responses and Diameter notifications. You add the data as name-value pairs. Diameter extensions can then access the data by translating the name-value pairs into AVPs.
- Credit threshold and credit limit notifications. You add the data as name-value pairs. ECE then accesses the data and updates the notifications for credit threshold breach and credit limit breach.

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 in the *ECE_Home*/**ocecesdk/source/oracle/communication/brm/charging/sdk/extensions** directory.

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)
    * 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 specifications, and uses the extensions repository to add a collection of ExtensionsData.
  - Asserts if the number of extensions data added to the repository are the same as the total being read.

How to Use the Sample Extensions

The following procedure shows how 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 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
```

   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 **ece.extensions-VERSION-SNAPSHOT.jar** with all the extensions classes and copies the JAR 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 pre-rating, rating, post-rating, and post-charging extensions fully qualified class names in the application configuration file **charging-settings.xml** under **$ECE_HOME/config/management** (the configuration can also be changed using the Extensions MBeans):

```
<extensions>
config-class="oracle.communication.brm.charging.appconfiguration.beans.
extensions.ExtensionsConfig"
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.SampleP
ostChargingExtension"
diameterGyExtension="oracle.communication.brm.charging.sdk.extensions.
SampleDiameterGyExtension"
</extensions>
```

8. Start/restart the ECE server node(s) 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 executed 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 brought 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 the "SamplePreRatingExtension invoked" and "PostRatingComplexTaxationSampleExtension executed" messages.

Operational Considerations

This section includes information about using the ECE extensions.

Configuration

You configure implementation classes for the diameter-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 and usage-request processing extension points:

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

Note: Ensure that the extension code is provided in **Classpath**, typically under the **$ECE_HOME/lib** directory, when the ECE Server running the JMX Management console is started. See "Sample Extensions" for more information.

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

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

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

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

2. Expand **charging.extensions**.
3. Expand **Attributes**.
4. Specify values for the following attributes as needed:
   - **diameterGyExtension**
   - **postRatingECEExtension**
   - **preRatingECEExtension**
   - **radiusAuthExtension**
   - **radiusAccountingExtension**

Note: Ensure that the extension code is provided in **Classpath**, typically under the **$ECE_HOME/lib** directory, when the ECE Server running the JMX Management console is started. See "Sample Extensions" for more information.

Performance

If extensions are activated, they are called for during every usage request. Always consider performance for the code you execute in the extensions.

The extensions framework provides an extensions cache mechanism that provides the lowest latency access to the extensions data. Oracle recommends 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.

Logging

Logging is available in the extensions using the log4j logger to server node log file; for example:

```
extensionContext.getLogger().debug("Hello World!" + extensionContext);
```

Exceptions

If there is a need to have the usage request rejected by ECE, it is possible to throw an **ExtensionsException**, which will cause the usage request to be rejected and report a "CUSTOM_EXTENSION_ERROR" reason code in the response.

For more information on the **ExtensionsException,** see the ECE Extensions JavaDocs in *BRM Elastic Charging Engine Java API Reference*.

Security

The following are the recommended best practices to ensure security for the extensions:

- Enable JMX security
- Enable ECE cluster node security
- Ensure strict governance of OS accounts
- Follow secure Java coding practices
- Implement string code review process
- Run latency-sensitive performance tests on the extensions hooks
- Use JAR signing

Best Practices

All pre-rating, rating, post-rating, and post-charging 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 will need to be taken into consideration when doing the sizing for Java.

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.

Oracle Communications Billing and Revenue Management Elastic Charging Engine 11.3 Extensions, Release 7.5
E70771-03