Oracle® Communications
Offline Mediation Controller

Record Enhancement Charging Cartridge User's Guide

Release 6.0

E64449-01

June 2015

ORACLE®

Oracle Communications Offline Mediation Controller Record Enhancement Charging Cartridge User's Guide, Release 6.0

E64449-01

Contents

Preface

This document describes how to use the Oracle Communications Offline Mediation Controller record enhancement charging Enhancement Processor (EP) cartridge to enhance call detail records (CDRs) before sending the records to the next node in the node chain.

Audience

This document is intended for solution designers who configure Offline Mediation Controller cartridges.

Downloading Oracle Communications Documentation

Product documentation is located on Oracle Help Center:

http://docs.oracle.com

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

https://edelivery.oracle.com

Related Documents

For more information, see the following documents:

- *Offline Mediation Controller Cartridge Development Kit Developer's Guide*: For information about how to develop a cartridge.
- *Offline Mediation Controller Cartridge Development Kit NPL Reference Guide*: For information about how to use the Node Programming Language for developing or extending a cartridge.

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.

1

Record Enhancement Charging Cartridge Overview

This chapter provides an overview of the Oracle Communications Offline Meditation Controller record enhancement charging Enhancement Processor (EP) cartridge to enhance call detail records (CDRs) before sending the records to the next node in the mediation node chain.

Before reading this chapter, you should be familiar with:

- Offline Mediation Controller cartridge concepts. For more information, see *Offline Mediation Controller Cartridge Development Kit Developer's Guide*.

About Record Enhancement Charging

Offline Mediation Controller collects CDRs from different input sources and normalizes the CDRs by mapping the following external data to the internal data formats that are supported by Oracle Communication Billing and Revenue Management (BRM) before sending the enhanced records to the next node in the mediation node chain:

- Social number
- Prefix description
- Service code
- Usage class
- Access point name (APN)

Workflow for CDRs

When a record enhancement charging EP node is configured in the mediation node chain, depending on the configuration in the NPL rule file, the records are processed as follows:

1. The collection cartridge (CC) node processes the CDR input file.
2. If the input file is successfully processed by the CC node, the record enhancement charging EP node normalizes the CDR data by mapping the data to the internal data formats supported by BRM.
3. The next node in the mediation node chain processes the returned CDR data, which is distributed to the target charging system.

2

Creating and Configuring the Record Enhancement Charging Enhancement Processor Cartridge Node

This chapter describes how to create and configure Oracle Communications Offline Mediation Controller record enhancement charging Enhancement Processor (EP) cartridge node for enhancing call detail records (CDRs).

Before reading this chapter, you should be familiar with:

- Offline Mediation Controller cartridge concepts and Node Programming Language (NPL).
- Setting up event data record (EDR) enrichment for rating and setting up pipeline price list data in the Oracle Communications Billing and Revenue Management (BRM) Pipeline Manager.

Configuring the Database Connection

Before you create the record enhancement charging EP node, configure the database connection information to Oracle Communication Billing and Revenue Management (BRM) database. The record enhancement charging EP cartridge node uses the database connection information to retrieve the mapping information while enhancing the CDR data.

To configure the database connection:

1. Open the *OMC_Home*/**config/nodemgr/prc.properties** file in a text editor, where *OMC_Home* is the directory in which Offline Mediation Controller is installed.
2. Add the following text:

```
driver=oracle.jdbc.driver.OracleDriver
url= jdbc:oracle:thin:@server_name:port_number:database_alias
username=login
password=password
```

   where:

   - *server_name* is the system on which the BRM database is installed.
   - *port_number* is the port number of the system running the BRM database.
   - *database_alias* is the BRM database alias of the schema you are connecting.
   - *login* is the user name for the database schema you are connecting.
   - *password* is the encoded password for *login*.

For information about encoding the password, see "Encoding the Database Connection Password".

3. Save and close the file.

Encoding the Database Connection Password

To encode the database connection password:

1. Go to the *OMC_Home*/**bin** directory.
2. Run the following command:

   ./encode *password*

   where *password* is the password for the user name for the database schema you are connecting.

   The command window displays *password* in encoded form. For example:

   ```
   72,-46,62,71,41,-115,14,-68,-34,-4,-105,-113,-125,1,-18,-70
   ```

Creating a Record Enhancement Charging EP Node

To create a record enhancement charging EP node:

1. Log on to Offline Mediation Controller Administration Client.

   The Node Hosts & Nodes (logical view) screen appears.
2. In the **Mediation Hosts** table, select a host.
3. In the **Nodes on Mediation Host** section, click **New**.

   The Create a Node dialog box appears.
4. Select **Wireless** and click **Next**.
5. Select **Enhancement Processor (EP)** and click **Next**.
6. Select **Record Enhancement Charging EP cartridge** and click **Finish**.

   The New Node dialog box appears.
7. In the **Name** field, enter a name for the node.
8. From the **Rule File** list, use the default sample rule file.

   To edit the rule file, see "Configuring the NPL Rule File for the Record Enhancement Charging EP Node".
9. Click the **General** tab and do the following:
   a. From the **Debug** list, select one of the following:

      To log short debug messages in the node log file, select **OFF**.

      To log detailed debug messages in the node log file, select **ON**.

   b. In the **Max Log File Size** field, enter the maximum size in bytes for the log file. When the log file reaches its limit, the node closes the file and opens a new file. The minimum value is **50000** and the maximum value is **2000000000**.

   c. Select the **Enable Statistics** check box, which enables node statistics.

   d. Select the **Enable bulk read/write** check box, which enables the node to read or write files in bulk.

e. In the **Read Timer** field, enter the number of seconds Offline Mediation Controller waits before checking for incoming records. The minimum value is **1** and the maximum value is **3600**.

f. In the **NARs Per File** field, enter the maximum number of NARs allowed in an output file. The minimum value is **1** and the maximum value is **10000**.

g. In the **Idle Write Time** field, enter the number of seconds Offline Mediation Controller waits before transferring the NAR output file to the output directory of the processing node, whether or not it has reached its maximum size. The minimum value is **1** and the maximum value is **3600**.

h. Select the **Backup NAR Files** check box, which enables the node to back up each processed NAR file.

i. In the **NAR File Retention Period** field, enter the number of days to retain the backup NAR files.

j. Select the **Input Stream Monitoring** check box, which enables the node to monitor the input stream of records and trigger an alarm when no records are received for the set interval.

k. In the **Interval** field, enter the duration of time that the node waits for the input stream of records before triggering an alarm when no records are received for the set interval. You also select the time unit: **Day**, **Hour**, or **Minute**.

10. (Optional) If you want to enable file-level transactions or multi-threading, click the **Advanced** tab and do the following:

    a. Select the **File Level Transaction** check box, which enables file-level transactions.

    b. Select the **Multi Threaded** check box, which enables multi-threading in the node to process multiple files in parallel.

       When the **Multi Threaded** check box is selected, the **Processing Threads** field and the **Enable Ordering** check box are enabled.

    c. In the **Processing Threads** field, enter the number of processing threads you require for your thread pool. The maximum value is **20**.

    d. If you require the order of the output data across all threads to be processed in the same order as the input data, select the **Enable Ordering** check box.

11. Click the **Destination** tab and do the following:

    a. Select the **Enable** check box, which enables the connection between the EP node and any destination cartridge node.

    b. From the **Routing** list, select one of the following:

       If the **Enable** check box is not selected, **Routing** is set to **None**.

       To enable multicast routing between the EP node and the destination cartridge node, select **Multicast**.

       To enable round-robin routing between the EP node and the destination cartridge node, select **Round Robin**.

12. Click **Save**.

Configuring the NPL Rule File for the Record Enhancement Charging EP Node

To configure the NPL rule file for record enhancement charging EP node:

1. Log on to Offline Mediation Controller Administration Client.

   The Node Hosts & Nodes (logical view) screen appears.

2. In the **Mediation Hosts** table, select the mediation host that contains the record enhancement charging EP node.

3. In the **Nodes on Mediation Host** section, select the record enhancement charging EP node that you want to configure, and click **Edit**.

   The Node dialog box appears.

4. From the **Rule File** list, select the sample rule file.

5. Click **Edit**.

   The NPL Editor dialog box appears.

6. (Optional) To normalize the social numbers, add the following Java hook and the corresponding output record configuration:

   JavaHook sn=oracle.communications.brm.nm.prc.nplhook.SocialNumberMethodHandler;

   For example:

```
JavaHook sn=oracle.communications.brm.nm.prc.nplhook.SocialNumberMethodHandler;
OutputRec {
    Map map;
    String sn;
    String name;
    String empty;
    String exist;
} snOut;
String str;
str="0014085555556";
snOut.exist=sn.VALUE(sn.exists(str));
snOut.map=sn.get(str);
snOut.sn=sn.getMapField(snOut.map, "socialnumber");
snOut.name=sn.getMapField(snOut.map, "name");
if (sn.isEmpty(snOut.map)==sn.TRUE()) {
snOut.empty="get "+str+" map is empty";
} else {
snOut.empty="get "+str+" map is NOT empty";
}
write(snOut);
```

7. (Optional) To normalize the call destination numbers or prefix descriptions, add the following Java hook and the corresponding output record configuration:

   JavaHook
   pd=oracle.communications.brm.nm.prc.nplhook.PrefixDescriptionMethodHandler;

   For example:

```
JavaHook
pd=oracle.communications.brm.nm.prc.nplhook.PrefixDescriptionMethodHandler;
OutputRec {
    Map map;
    String areacode;
```

```
    String type;
    String name;
    String empty;
    String exist;
} pdOut;
String str;
str="00004570";
pdOut.exist=pd.VALUE(pd.exists(str));
pdOut.map=pd.search(str);
pdOut.areacode=pd.getMapField(pdOut.map, "areacode");
pdOut.type=pd.getMapField(pdOut.map, "type");
pdOut.name=pd.getMapField(pdOut.map, "name");
if (pd.isEmpty(pdOut.map)==pd.TRUE()) {
pdOut.empty="serach "+str+" map is empty";
} else {
pdOut.empty="search "+str+" map is NOT empty";
}
write(pdOut);
```

8. (Optional) To normalize the service code maps, add the following Java hook and the corresponding output record configuration:

```
JavaHook
scm=oracle.communications.brm.nm.prc.nplhook.ServiceCodeMapMethodHandler;
```

For example:

```
JavaHook
scm=oracle.communications.brm.nm.prc.nplhook.ServiceCodeMapMethodHandler;
OutputRec {
    Map map;
    String mapGroup;
    String rank;
    String extServicecode;
    String usageclass;
    String locarindVasevent;
    String qosRequested;
    String qosUsed;
    String recordtype;
    String intServicecode;
    String intServiceclass;
    String empty;
    String exist;
} scmOut;
String str;
String str2;
String str3;
str="ALL_RATE";
str2="TELEPHONY";
str3="MBI";
scmOut.exist=scm.VALUE(scm.exists(str));
scmOut.map=scm.search(str, str2, str3, "", "", "", "");
scmOut.mapGroup=scm.getMapField(scmOut.map, "MAP_GROUP");
scmOut.rank=scm.getMapField(scmOut.map, "RANK");
scmOut.extServicecode=scm.getMapField(scmOut.map, "EXT_SERVICECODE");
scmOut.usageclass=scm.getMapField(scmOut.map, "USAGECLASS");
scmOut.locarindVasevent=scm.getMapField(scmOut.map, "LOCARIND_VASEVENT");
scmOut.qosRequested=scm.getMapField(scmOut.map, "QOS_REQUESTED");
scmOut.qosUsed=scm.getMapField(scmOut.map, "QOS_USED");
scmOut.recordtype=scm.getMapField(scmOut.map, "RECORDTYPE");
scmOut.intServicecode=scm.getMapField(scmOut.map, "INT_SERVICECODE");
```

```
scmOut.intServiceclass=scm.getMapField(scmOut.map, "INT_SERVICECLASS");
if (scm.isEmpty(scmOut.map)==scm.TRUE()) {
scmOut.empty="search "+str+","+str2+","+str3+" map is empty";
} else {
scmOut.empty="search "+str+","+str2+","+str3+" map is NOT empty";
}
write(scmOut);
```

9. (Optional) To normalize the usage class maps, add the following Java hook and the corresponding output record configuration:

   JavaHook
   ucm=oracle.communications.brm.nm.prc.nplhook.UsageClassMapMethodHandler;

   For example:

```
JavaHook
ucm=oracle.communications.brm.nm.prc.nplhook.UsageClassMapMethodHandler;
OutputRec {
    Map map;
    String mapGroup;
    String rank;
    String extUsageclass;
    String usagetype;
    String zoneWs;
    String tariffclass;
    String tariffsubclass;
    String recordtype;
    String connecttype;
    String connectsubtype;
    String apnAddress;
    String ssPacket;
    String transitAreacode;
    String name;
    String intUsageclass;
    String empty;
    String exist;
} ucmOut;
String str;
String str2;
str="ALL_RATE";
str2="mBI";
ucmOut.exist=ucm.VALUE(ucm.exists(str));
ucmOut.map=ucm.search(str, str2, "", "", "", "", "", "", "", "", "");
ucmOut.mapGroup=ucm.getMapField(ucmOut.map, "MAP_GROUP");
ucmOut.rank=ucm.getMapField(ucmOut.map, "RANK");
ucmOut.extUsageclass=ucm.getMapField(ucmOut.map, "EXT_USAGECLASS");
ucmOut.usagetype=ucm.getMapField(ucmOut.map, "USAGETYPE");
ucmOut.zoneWs=ucm.getMapField(ucmOut.map, "ZONE_WS");
ucmOut.tariffclass=ucm.getMapField(ucmOut.map, "TARIFFCLASS");
ucmOut.tariffsubclass=ucm.getMapField(ucmOut.map, "TARIFFSUBCLASS");
ucmOut.recordtype=ucm.getMapField(ucmOut.map, "RECORDTYPE");
ucmOut.connecttype=ucm.getMapField(ucmOut.map, "CONNECTTYPE");
ucmOut.connectsubtype=ucm.getMapField(ucmOut.map, "CONNECTSUBTYPE");
ucmOut.apnAddress=ucm.getMapField(ucmOut.map, "APN_ADDRESS");
ucmOut.ssPacket=ucm.getMapField(ucmOut.map, "SS_PACKET");
ucmOut.transitAreacode=ucm.getMapField(ucmOut.map, "TRANSIT_AREACODE");
ucmOut.name=ucm.getMapField(ucmOut.map, "NAME");
ucmOut.intUsageclass=ucm.getMapField(ucmOut.map, "INT_USAGECLASS");
if (ucm.isEmpty(ucmOut.map)==ucm.TRUE()) {
ucmOut.empty="search "+str+","+str2+" map is empty";
```

```
} else {
ucmOut.empty="search "+str+","+str2+" map is NOT empty";
}
write(ucmOut);
```

10. (Optional) To normalize the access point name (APN) maps, add the following Java hook and the corresponding output record configuration:

    JavaHook
    apnm=oracle.communications.brm.nm.prc.nplhook.AccessPointNameMapMethodHandler;

    For example:

```
JavaHook
apnm=oracle.communications.brm.nm.prc.nplhook.AccessPointNameMapMethodHandler;
OutputRec {
    Map map;
    String apnGroup;
    String rank;
    String servicecode;
    String accesspointname;
    String pdpAddress;
    String empty;
    String exist;
} apnmOut;
str="MY_TEST";
str2="GPR";
str3="oracle.de";
apnmOut.exist=apnm.VALUE(apnm.exists(str));
apnmOut.map=apnm.search(str, str2, str3);
apnmOut.apnGroup=apnm.getMapField(apnmOut.map, "APN_GROUP");
apnmOut.rank=apnm.getMapField(apnmOut.map, "RANK");
apnmOut.servicecode=apnm.getMapField(apnmOut.map, "SERVICECODE");
apnmOut.accesspointname=apnm.getMapField(apnmOut.map, "ACCESSPOINTNAME");
apnmOut.pdpAddress=apnm.getMapField(apnmOut.map, "PDP_ADDRESS");
if (apnm.isEmpty(apnmOut.map)==apnm.TRUE()) {
apnmOut.empty="search "+str+","+str2+","+str3+" map is empty";
} else {
apnmOut.empty="search "+str+","+str2+","+str3+" map is NOT empty";
}
write(apnmOut);
```

11. Compile and save the file.
12. Close the NPL Editor dialog box.
13. Click **Save**.

    The configuration is saved.

3

Working with Record Enhancement Charging Java Hooks in NPL

This chapter lists and describes the Java hooks available for the Oracle Communications Offline Mediation Controller record enhancement charging Enhancement Processor (EP) cartridge.

About Record Enhancement Charging Java Hooks

Java hooks are an advanced feature of NPL (Node Programming Language) that enable Offline Mediation Controller to call a Java method from an NPL program. For more information on using Java hooks with NPL, see the discussion on Java hooks in *Offline Mediation Controller Cartridge Development Kit NPL Reference Guide*.

Table 3–1 lists the record enhancement charging Java hooks methods.

Table 3–1 Record Enhancement Charging Java Hooks Method Summary

Modifier and Type	Method and Description
void	load() throws NodeProcessingException Loads data from the database into memory.
IntField	exists(StringField *key*) throws NodeProcessingException Searches for a key in the data collection for the configuration service.
MapField	get(StringField *key*) throws NodeProcessingException Searches for the field that contains the record that matches *key*.
StringField	getLoadedInfo() throws NodeProcessingException Searches for the string that represents the cached data and time it was cached.
StringField	getMapField(MapField *map*, StringField *fieldName*) throws NodeProcessingException Searches for the value of *fieldName* in *map*.
IntField	isEmpty(DCField *field*) throws NodeProcessingException Verifies if *field* contains any values.
IntField	TRUE() throws NodeProcessingException Use this method instead of checking if the return value is **1** (true).
IntField	FALSE() throws NodeProcessingException Use this method instead of checking if the return value is **0** (false).
StringField	VALUE(IntField *val*) throws NodeProcessingException Use this method to return the string representation of *val*.

Table 3–1 (Cont.) Record Enhancement Charging Java Hooks Method Summary

Modifier and Type	Method and Description
MapField	search(StringField *areacode*) throws NodeProcessingException Searches for the longest best match in the cached data for *areacode*.
MapField	search(StringField *mapGroup*, StringField *extServicecode*, StringField *usageclass*, StringField *locarindVasevent*, StringField *qosRequested*, StringField *qosUsed*, StringField *recordtype*) throws NodeProcessingException Searches for the first ranked record matching the search criteria.
MapField	search(StringField *mapGroup*, StringField *extUsageclass*, StringField *usagetype*, StringField *zoneWs*, StringField *tariffclass*, StringField *tariffsubclass*, StringField *recordtype*, StringField *connecttype*, StringField *connectsubtype*, StringField *transitAreacode*, StringField *apnAddress*, StringField *ssPacket*) throws NodeProcessingException Searches for the first ranked record matching the search criteria.
MapField	search(StringField *apnGroup*, StringField *servicecode*, StringField *accesspointname*) throws NodeProcessingException Searches for the first ranked record matching the search criteria for the access point name (APN) group.
MapField	getByNo(IntField *no*) throws NodeProcessingException Searches for the network operator record having the internal ID *no*.

Record Enhancement Charging Java Hook Method Details

The section describes the record enhancement charging Java hook methods.

load

```
void load() throws NodeProcessingException
```

Usage

This function loads data from the database into memory.

Parameters

This method has no parameters.

Returns

This function returns nothing.

exists

```
IntField exists(StringField key) throws NodeProcessingException
```

Usage

This function searches for a key in the data collection for the configuration service.

Parameters

key is the key field in the record to search for.

Returns

1 (true) if the key is found in the record.

0 (false) if the key is not found in the record.

get

```
MapField get(StringField key) throws NodeProcessingException)
```

Usage

This function searches for the record that matches *key*. The database column name (case insensitive) is used as the field name in the MapField. For Service Code Map and Usage Class Map, which are keyed by the map_group, the first ranked record is returned if found.

Parameters

key is the key field in the record.

Returns

The record that matches the key.

getLoadedInfo

```
StringField getLoadedInfo() throws NodeProcessingException)
```

Usage

This function searches for the string that represents the cached data and time it was cached.

Parameters

This method has no parameters.

Returns

The string that represents the cached data and the time the data was cached.

getMapField

```
StringField getMapField(MapField map, StringField fieldName) throws
NodeProcessingException)
```

Usage

This function searches for the value of *fieldName* in *map*.

Parameters

map is the MapField in which the value for *fieldName* is to be found.

fieldName is the field name for which the value is to be returned.

Returns

The string associated with *fieldName*.

An empty string ("") is returned if the field is not found.

isEmpty

```
IntField isEmpty(DCField field) throws NodeProcessingException)
```

Usage
This function verifies if *field* contains any values.

Parameters
field is the DCField that contains the field type and field value.

Returns
1 (true) if the field is empty.

0 (false) if the field is not empty.

TRUE

```
IntField TRUE() throws NodeProcessingException)
```

Usage
This function can be used to verify that the return value is **1** (true).

Parameters
This method has no parameters.

Returns
1 (true) if the return value is true.

FALSE

```
IntField FALSE() throws NodeProcessingException)
```

Usage
This function can be used to verify that the return value is **0** (false).

Parameters
This method has no parameters.

Returns
0 (false) if the return value is false.

VALUE

```
StringField VALUE(IntField val) throws NodeProcessingException)
```

Usage
This function returns the string representation of *val*.

Parameters
val is the IntField to be converted from an integer to a string.

Returns
The string representation of *val*.

search

```
MapField search(StringField areacode) throws NodeProcessingException)
```

Usage

This function searches for the longest best match in the cached data for *areacode*.

Parameters

areacode is the area code to search.

Returns

Returns the longest best match for *areacode*.

search

```
MapField search(StringField mapGroup, StringField extServicecode, StringField
usageclass, StringField locarindVasevent, StringField qosRequested, StringField
qosUsed, StringField recordtype) throws NodeProcessingException
```

Usage

This function searches for the first ranked record matching the search criteria.

Parameters

mapGroup is the map group.

extServicecode is the name of the external service code map.

usageclass is the name of the usage class map.

locarindVasevent is the MSC responsible for handling the call and the location of the equipment making or receiving the call.

qosRequested is the type of QoS requested.

qosUsed the type of QoS negotiated by the network.

recordtype is the record type.

Returns

The first ranked record matching the search criteria.

search

```
MapField search(StringField mapGroup, StringField extUsageclass, StringField
usagetype, StringField zoneWs, StringField tariffclass, StringField
tariffsubclass, StringField recordtype, StringField connecttype, StringField
connectsubtype, StringField transitAreacode, StringField apnAddress, StringField
ssPacket) throws NodeProcessingException
```

Usage

This function searches for the first ranked record matching the search criteria.

Parameters

mapGroup is the map group.

extUsageclass is the external usage class.

usagetype is the customer-related usage scenario.

zoneWs is the impact category for wholesale zone.

tariffclass is the tariff class that contains the tariff information.

tariffsubclass is the detailed tariff information.

recordtype is the record type.

connecttype is the type of connection.

connectsubtype is the detailed description of the connection or call type.

transitAreacode is the area code.

apnAddress is the logical name of the connected access point to the external packet data network.

ssPacket is the number of supplementary service records.

Returns

The first record matching the search criteria.

search

```
MapField search(StringField apnGroup, StringField servicecode, StringField
accesspointname) throws NodeProcessingException
```

Usage

This function searches for the first ranked record matching the search criteria for the access point name (APN) group.

Parameters

apnGroup is the APN group.

servicecode is the service code.

accesspointname is the APN name.

Returns

The first record matching the search criteria for the APN.

getByNo

```
MapField getByNo(IntField no) throws NodeProcessingException
```

Usage

This function searches for the network operator record having the internal ID *no*.

Parameters

no is the internal ID for the network operator.

Returns

The network operator record having the internal ID *no*.