Oracle® Communications
Offline Mediation Controller

Suspending and Recycling Call Detail Records User's Guide

Release 6.0

E52524-01

May 2014

ORACLE®

Oracle Communications Offline Mediation Controller Suspending and Recycling Call Detail Records User's Guide, Release 6.0

E52524-01

Contents

6 Suspense Fields

7 Working with Suspense Java Hooks in NPL

Preface

This document describes how to use the Oracle Communications Offline Mediation Controller suspense and recycle cartridges with Oracle Communications Billing and Revenue Management (BRM) Suspense Manager to collect and manage suspended and recycled call detail records (CDRs).

Audience

This document is intended for charging solution designers who configure Offline Mediation Controller and BRM for suspense handling.

Downloading Oracle Communications Documentation

Product documentation is located on Oracle Technology Network:

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.
- *Oracle Communications Billing and Revenue Management Concepts*: For an overview of the BRM software.
- *Oracle Communications Billing and Revenue Management Configuring Pipeline Rating and Discounting*: For information about suspense handling in BRM.
- *Oracle Communications Billing and Revenue Management Installation Guide*: For an overview of Account Synchronization and Oracle AQ queueing.

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 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

Suspending and Recycling CDRs Overview

This chapter provides an overview of the Oracle Communications Offline Meditation Controller suspense handling, which collects and manages suspended and recycled call detail records (CDRs) after detection of a failed CDR or CDR file.

Before reading this chapter, you should be familiar with:

- Oracle Communications Billing and Revenue Management (BRM) concepts and Suspense Manager. For more information, see the BRM documentation.
- Offline Mediation Controller cartridge concepts. For more information, see *Offline Mediation Controller Cartridge Development Kit Developer's Guide*.

About Suspending and Recycling Call Detail Records

Offline Mediation Controller suspense handling is the process of collecting and recycling suspended CDRs with the Offline Mediation Controller suspense and recycle cartridges and managing suspended and recycled CDRs with Suspense Manager, a component of BRM. Suspense handling collects, recycles, fixes, write-offs, and audits your suspended CDRs.

Suspense handling is used for failed event or batch CDRs, where:

- An event action processes individual CDRs.
- A batch action processes a file that contains a collection of CDRs.

Failed event or batch CDRs can occur when:

- There is an issue with the CDR, such as, missing or incorrect fields.
- There is a problem with the CDR file due to a bad policy or configuration.
- There is an issue in your system configuration, such as, it contains the wrong pricing information or the account information is not loaded into the system.

Before you can use the Offline Mediation Controller suspense handling, do the following:

- Integrate Offline Mediation Controller with BRM. For more information, see "Integrating Offline Mediation Controller with BRM".
- Create and configure cartridges to detect either event or batch CDR errors. For more information, see "Configuring Cartridges to Detect and Suspend Failed CDRs".
- Create and configure the Offline Mediation Controller suspense and recycle cartridges for either event or batch suspense handling. For more information, see "Creating and Configuring the Suspense and Recycle Cartridges".

- If you use Oracle Communications Billing and Revenue Management Elastic Charging Engine (ECE) for rating, create and configure the Offline Mediation Controller ECE Distribution Cartridge (DC). For more information, see "Implementing Suspense Handling for the ECE Cartridge Pack".

Suspense Handling Workflow for Event CDRs

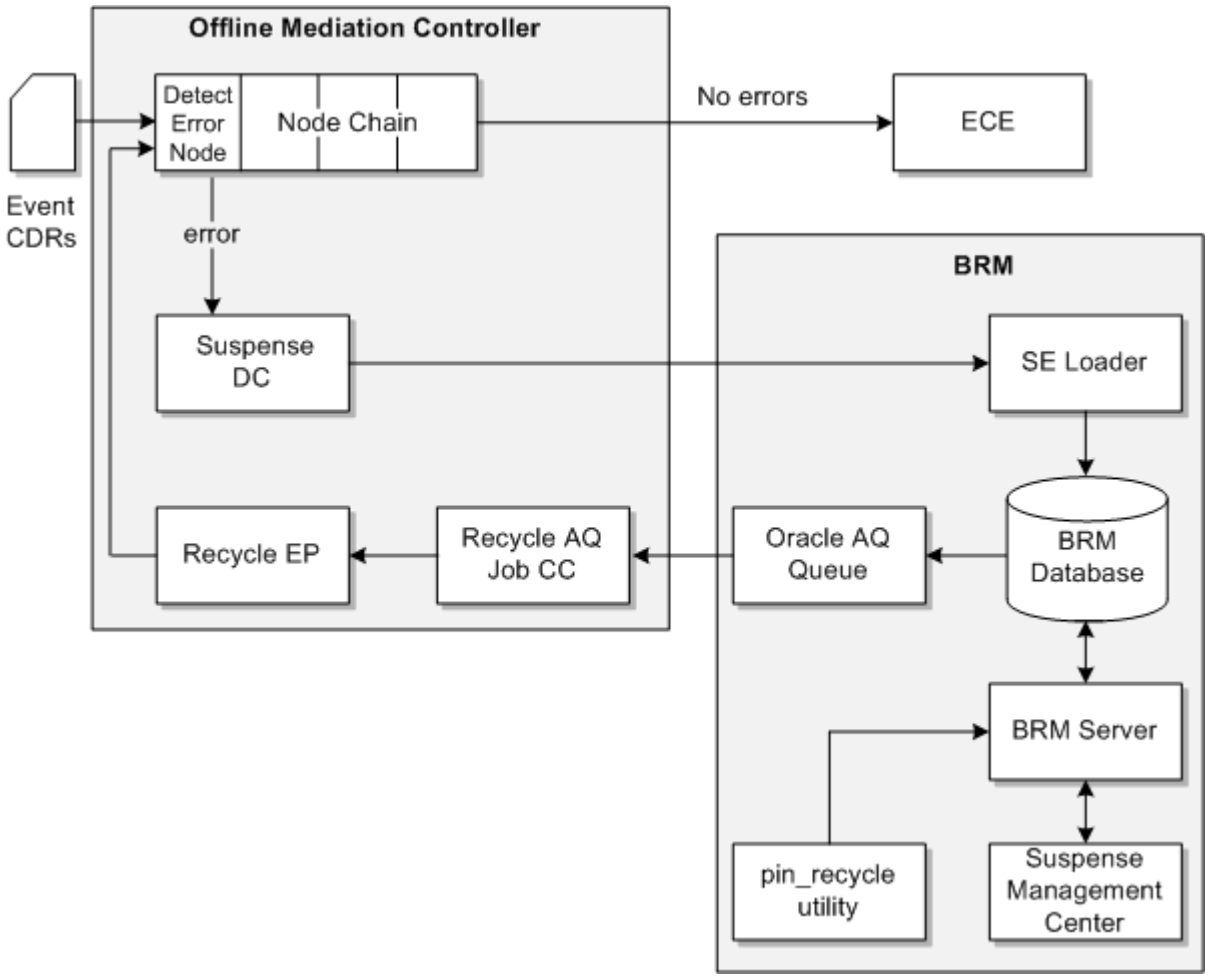
The suspense handling workflow for event CDRs is as follows:

1. Nodes in the node chain, such as, the ECE DC node, are configured to detect errors. For more information, see "Configuring Cartridges to Detect and Suspend Failed CDRs".
2. The CDRs enter the node chain, where:
   - If no errors are detected, they pass through the node chain's processing flow and are distributed to the target system. For example the ECE DC node will distribute output to ECE for rating.
   - If an error is detected, the CDR input stream, error code, cartridge name, cartridge category, and (if the CDR is recycled) the suspense ID are made available for the Suspense Distribution Cartridge (DC) node.
3. The Suspense DC node receives the suspended CDRs and generates **Create** and **Update** files in a format that is understood by Suspended Event (SE) Loader, which is a component of BRM. For more information, see "About the Suspense Distribution Cartridge".
4. The **Create** or **Update** files are received by the SE Loader application, which is used to load suspended events into **/suspended_usage** objects in the BRM database.
5. The suspended CDR, which is in a suspended state, is fixed, submitted for recycling, and its status updated to **Recycling** in Suspense Management Center. For more information on the Suspense Management Center statuses, see the BRM documentation.
6. The recycle request is sent to the Oracle AQ queue, by Suspense Management Center or the **pin_recycle** utility, which are components of BRM.
7. The Recycle AQ Job Collection Cartridge (CC) node receives and parses the job ID message of the recycle request. For more information, see "About the Recycle AQ Job Collection Cartridge".
8. The Recycle Enhancement Processor Cartridge (EP) node uses the job ID message from the Recycle AQ Job CC node to restore the recycled CDR for reprocessing in Offline Mediation Controller. For more information, see "About the Recycle Enhancement Processor Cartridge".
9. The recycled CDR passes back to the node that detected the error and on through the node chain's processing flow, where:
   - If no more errors are detected, the CDR is distributed to the target system, such as, ECE for rating. Also, the recycled CDR goes back to the Suspense DC node, which changes the status to a succeeded state and generates an **Update** file for SE Loader.
   - If more errors are detected, the recycled CDR is sent back to the Suspense DC node, which changes the status to a suspended state and generates an **Update** file for SE Loader.

The recycled CDR continues through the suspense handling flow until either it is removed (**Written off**) by Suspense Management Center, or it is successful (**Succeeded**).

Figure 1–1 shows the suspense handling flow of suspended and recycled event CDRs.

Figure 1–1 Event Suspense Handling Flow



Suspense Handling Workflow for Batch CDRs

The suspense handling workflow for batch CDRs is as follows:

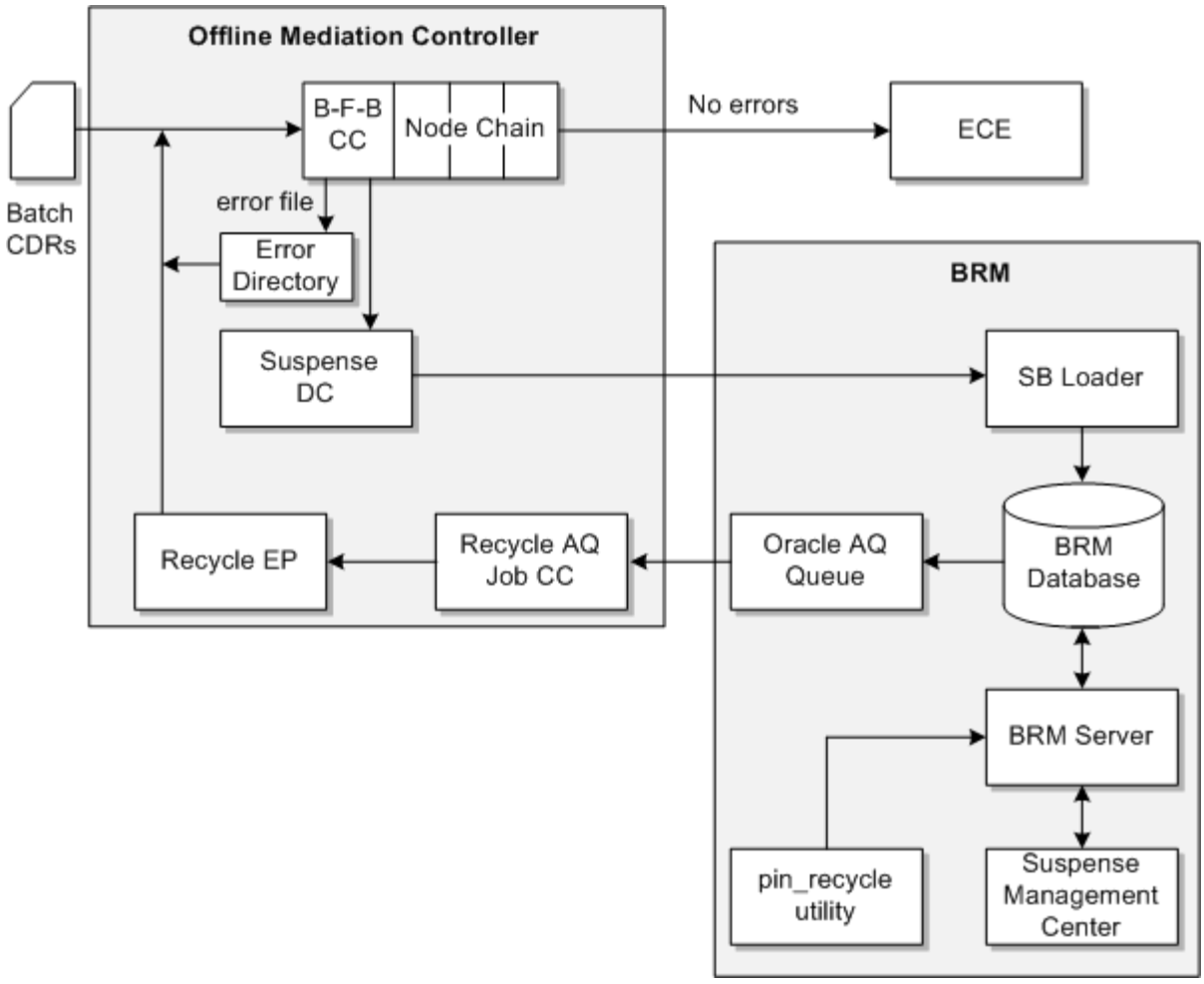
1. A batch-file-based CC node is configured to detect errors. For more information, see "Configuring Cartridges to Detect and Suspend Failed CDRs".
2. The batch CDR file enters the batch-file-based CC node, where:
   - If no errors are detected, processing continues.
   - If an error is detected, the suspended batch CDR file is put into an error directory and information on the batch CDR file's suspension is made available.
3. The Suspense DC node receives a file containing information on the suspended batch CDR file and generates **Create** or **Update** files in a format that is understood by SB Loader, which is a component of BRM. For more information, see "About the Suspense DC for Batch Mode".

4. The **Create** or **Update** files are received by the SB Loader application, which is used to load information on failed batch files into **/suspended_batch** objects in the BRM database.
5. The suspended batch, which is in a suspended state, is resubmitted to Offline Mediation Controller and its status updated to **Resubmitting** in Suspense Management Center. For more information on the Suspense Management statuses, see the BRM documentation.
6. A resubmitted batch request is sent to the Oracle AQ queue, by Suspense Management Center or the **pin_recycle** utility.
7. The Recycle AQ Job CC node receives and parses the job ID message of the resubmitted batch request. For more information, see "About the Recycle AQ Job CC for Batch Mode".
8. The Recycle EP node uses the job ID message from the Recycle AQ Job CC node to retrieve the resubmitted batch and returns the suspended batch CDR file back to the input stream of the batch-file-based CC. For more information, see "About the Recycle EP for Batch Mode".
9. The batch-file-based CC node continues the processing of the suspended batch CDR file:
   - If no more errors are detected, the CDRs from the suspended batch CDR file are distributed to the target system, such as, ECE for rating. Also, the resubmitted batch goes back to the Suspense DC node, which changes the status to a succeeded state and generates an **Update** file for SB Loader.
   - If more errors are detected, the resubmitted batch is sent back to the Suspense DC node, which changes the status to a suspended state and generates an **Update** file for SB Loader.

     The resubmitted batch continues through the suspense handling flow until either it is removed (**Written off**) by Suspense Management Center, or it is successful (**Succeeded**).

Figure 1–2 shows the suspense handling flow of suspended and resubmitted batch CDRs.

Figure 1–2 Batch Suspense Handling Flow



2

Integrating Offline Mediation Controller with BRM

This chapter describes how to integrate Oracle Communications Offline Mediation Controller with Oracle Communications Billing and Revenue Management (BRM) for suspense handling.

Before reading this chapter, you should be familiar with BRM concepts and architecture related to suspense handling and the Account Synchronization Data Manager (DM), which is a component of BRM.

About Suspense Manager

Suspense Manager, which is a component of BRM, is an optional service integration component that you purchase separately. You use Suspense Manager to:

- Analyze, edit, recycle, write off, archive, restore, resubmit, and delete individual (event) CDRs that have failed processing.
- Analyze, resubmit, write off, and delete CDR files (batch) containing any number of individual CDRs that have failed processing. CDR files cannot be edited or archived.

Suspense Manager includes the Suspense Management Center client that allows you to perform these tasks using a graphical user interface (GUI).

Integrating Offline Mediation Controller with Suspense Manager

The following steps summarize what is required to integrate Offline Mediation Controller with Suspense Manager to manage suspended record-level (event) and file-level (batch) CDRs:

Note: Before integrating Offline Mediation Controller with Suspense Manager, verify the following:

- Offline Mediation Controller is installed.
- Suspense Manager is installed. For more information on how to install Suspense Manager, see the discussion about installing Suspense Manager in the BRM documentation.

---

1. Configure and customize Suspense Management Center.

   You use the Suspense Management Center GUI to analyze, edit, recycle, write off, archive, restore, resubmit, and delete suspended event CDRs or batch CDR files.

For more information, see the discussion about configuring and customizing Suspense Management Center in the BRM documentation.

2. Configure the event notifications for Suspense Manager.

   Suspense Manager uses event notifications to perform follow-up operations.

   For more information, see the discussion about configuring event notification for Suspense Manager in the BRM documentation.

3. Create a list of queryable fields.

   A queryable field list is a list of event data record (EDR) fields that you use to search and analyze suspended CDRs. Suspense Management Center allows you to search for suspended CDRs based on values in these queryable fields, and displays these values in your search results.

   For more information, see the discussion about creating a list of editable fields based on your **/suspended_usage** subclasses in the BRM documentation.

4. Create a list of editable fields.

   An editable fields list, is a list of EDR fields that you use to edit and correct failed CDRs in Suspense Management Center.

   For more information, see the discussion about creating a list of editable fields based on your **/suspended_usage** subclasses in the BRM documentation.

5. Load editable fields into the database.

   For more information, see the discussion about loading editable fields into the database in the BRM documentation.

6. Configure the Suspended Event (SE) Loader and the Suspended Batch (SB) Loader applications, which are components of BRM.

   For Offline Mediation Controller suspense handling:

   - The SE Loader application loads suspended events into **/suspended_usage** objects in the BRM database.
   - The SB Loader application loads information about suspended batch files into **/suspended_batch** objects in the BRM database.

   For more information, see the discussions about setting up SE Loader and SB Loader for Suspense Manager in the BRM documentation.

7. Create indexes for search templates.

   You can improve database performance by creating indexes for your most common searches.

   For more information, see the discussion about creating indexes for search templates in the BRM documentation.

8. (Optional) Change suspense reasons and subreasons.

   If the default error messages or error message mappings do not meet your business needs, you can change them.

   You use the **load_pin_suspense_reason_code** (event) or the **load_pin_batch_suspense_reason_code** (batch) utilities, which are components of BRM, to load your suspense reason code mapping into the BRM database.

   For more information, see the discussion on changing the list of suspense reasons and subreasons in the BRM documentation.

9. (Optional) Configure debugging.

   Suspense Management Center provides several ways to capture and display debugging information.

   For more information, see the discussion about configuring debugging in the BRM documentation.

About Oracle AQ Messaging for Suspense Handling

In Offline Mediation Controller suspense handling, the Oracle AQ database queue publishes business events from the Account Synchronization Data Manager (DM), which are components of BRM, to the Oracle AQ database queue. The Recycle AQ Job Collection Cartridge (CC) receives the business event and dequeues the job ID message from the Oracle AQ database queue.

For more information on the Account Synchronization DM, see the BRM documentation.

For more information on the Recycle AQ Job CC, see "About the Recycle AQ Job Collection Cartridge".

Integrating Offline Mediation Controller with Oracle AQ Messaging

The following steps summarize what is required to integrate Offline Mediation Controller with Oracle AQ messaging for suspense handling:

1. Verify that your database is configured for advanced queuing.

   For more information, see the discussion on configuring database machines for advance queuing in the BRM documentation.

2. Verify that Account Synchronization DM is installed and configured for Oracle AQ queues.

   For more information, see the discussion on installing and configuring the Account Synchronization DM in the BRM documentation.

3. Configure the BRM event notification and the Enterprise Application Integration (EAI) payload.

   The Account Synchronization EAI framework notifies the Account Synchronization DM when an event occurs and requires some action, and sends the business event payload, which comprises of all the BRM events that belong to a specific event, to the Account Synchronization DM.

   For more information, see the discussion about the EIA framework in the BRM documentation.

4. Verify that the **acct_sync** stored procedure package exists.

   To send events to multiple queues in different schemas in a multischema system, you must grant running permission for each source schema user from the target schema, for the **acct_sync** package.

   For more information, see the discussion on installing and configuring account synchronization in the BRM documentation.

5. Configure the published format for the payload definition (**payloadconfig_ifw_sync.xml**) file as XML by doing the following:

   a. Open the *BRM_Home*/**sys/eai_js/payloadconfig_ifw_sync.xml** file, where *BRM_Home* is the directory in which BRM is installed.

b. Search for the following line:

```
<PublisherDefs>
```

c. Add the following:

```
<Publisher DB="0.0.9.9" Format="XML">
   RecycleRequest,
   ResubmitBatchRequest
</Publisher>
```

d. Save and close the file.

6. Enable the **RecycleRequest** and **ResubmitBatchRequest** business events in the (**payloadconfig_ifw_sync.xml**) file by doing the following:

   a. Open the *BRM_Home*/**sys/eai_js/payloadconfig_ifw_sync.xml** file.

   b. Search for the following lines and adjust accordingly:

```
<!-- For Suspended Events -->
<RecycleRequest Source="EVENT"
   Tag="RecycleRequest"
StartEvent="/event/notification/suspense/recycle" >
   <Attribute Tag="Version" Value="1.0" />
   <Field PinFld="PIN_FLD_ACCOUNT_OBJ" Tag="AccountObj"/>
   <SubElement Name="JobActions"
OnEvent="/event/notification/suspense/recycle" />
</RecycleRequest>

<!-- For Suspended Batch -->
<ResubmitBatchRequest Source="EVENT"
   Tag="ResubmitBatchRequest"
   StartEvent="/event/notification/suspense/batch_resubmit" >
   <Attribute Tag="Version" Value="1.0" />
   <Field PinFld="PIN_FLD_ACCOUNT_OBJ" Tag="AccountObj"/>
   <SubElement Name="JobActions"
    OnEvent="/event/notification/suspense/batch_resubmit" />
</ResubmitBatchRequest>

<!-- Action Obj -->
<JobActions Source="EVENT" PinFld="PIN_FLD_ACTIONS"
   DataFrom="PIN_FLD_ACTIONS" Tag="Actions" >
   <Field PinFld="PIN_FLD_ACTION_OBJ" Tag="ActionObj" />
</JobActions>
```

   c. Save and close the file.

For more information, see the discussion about configuring the account synchronization in the BRM documentation.

Integrating EDR Field Mapping

The Suspense Distribution Cartridge (DC) node serializes the CDR based on the output field names from the NPL mapping. Shorter designations may be used in place of the output field names by using EDR field mapping, which reduces the length of field names by replacing the field name with an ID.

For suspense handling, EDR field mapping is used by the Suspense DC and the Recycle Enhancement Processor Cartridge (EP) nodes for suspended event CDRs.

EDR field mapping uses the following:

- The *BRM_Home*/**xsd/edr_field_mapping.xsd** XML schema file, contains valid XML rules and values.
- The *BRM_Home*/**sys/data/config/edr_field_mapping.xml** configuration file, maps the EDR field name to an ID number. The **edr_field_mapping.xml** file must conform to the XML schema rules as defined in the **edr_field_mapping.xsd** file.
- The *BRM_Home*/**bin/load_edr_field_mapping** utility, loads the EDR field mapping configuration file into the BRM database.

Important: The sample **edr_field_mapping.xml** file is loaded into the BRM database during installation of the Suspense Manager Server.

After installation, changes can be made to the sample **edr_field_mapping.xml** file using the **load_edr_field_mapping** utility.

Updating the EDR Field Mapping File

After installation of the Suspense Manager Server, changes can be made to the **edr_field_mapping.xml** file.

To update the EDR field mapping file:

1. Go to the *BRM_Home*/**sys/data/config/** directory.
2. Open the **edr_field_mapping.xml** file in a text editor.
3. Search for the following line:

   ```
   <edr_field_mapping name="Name">
   ```

   where, *Name* is the version number of the mapping.
4. Add or update an **id** field entry, using the following syntax:
   - The **id** field must contain numbers and periods.

     The parent block field container must end with a **.b** value. The childs **id** field must be sequentially numbered and prefixed with the parents **id** block field's value, without the **.b** value.
   - A **type** field, which contains the numbers **1, 2, 4, 8**, or **32**.

     where:
     - **1** is used for **string** data types.
     - **2** is used for **integer** data types.
     - **4** is used for **date** data types.
     - **8** is used for **decimal** data types.
     - **32** is used for **block** data types.

   For example:

   ```
   <f id="1.5.b" name="DETAIL.ASS_DUMMY_EXT" type="32"/>
   <f id="1.5.0.1" name="DETAIL.ASS_DUMMY_EXT.RECORD_TYPE" type="1"/>
   <f id="1.5.1.2" name=DETAIL.ASS_DUMMY_EXT.RECORD_NUMBER" type="2"/>
   ```

Important: Before new entries are loaded in the database, all existing entries that contain the same version number as the new entries are deleted. To prevent overwriting of existing entries when the **edr_field_mapping.xml** file is loaded into the database, configure the **name** attribute to increment the version number to the next number.

5. Save and close the file.

Loading EDR Field Mapping into the BRM Database

To load the EDR field mapping into the BRM Database:

1. Go to the *BRM_Home*/**sys/data/config/** directory.
2. Run the following command, which loads the EDR field mapping file into the BRM database:

   ```
   load_edr_field_mapping XML_file
   ```

   where *XML_file* is the name of the XML file that contains the configuration data.

3

Configuring Cartridges to Detect and Suspend Failed CDRs

This chapter describes how to configure Oracle Communications Offline Mediation Controller cartridges to detect and suspend failed call detail records (CDRs).

Before reading this chapter, you should be familiar with:

- Oracle Communications Billing and Revenue Management (BRM) suspense handling and the BRM suspense components outlined in "Integrating Offline Mediation Controller with BRM".
- Offline Mediation Controller cartridge concepts and Node Programming Language (NPL).
- The Offline Mediation Controller ECE Cartridge Pack. For more information, see *Offline Mediation Controller Elastic Charging Engine Cartridge Pack User Guide*.

Deciding Your Suspense Handling Output Flow

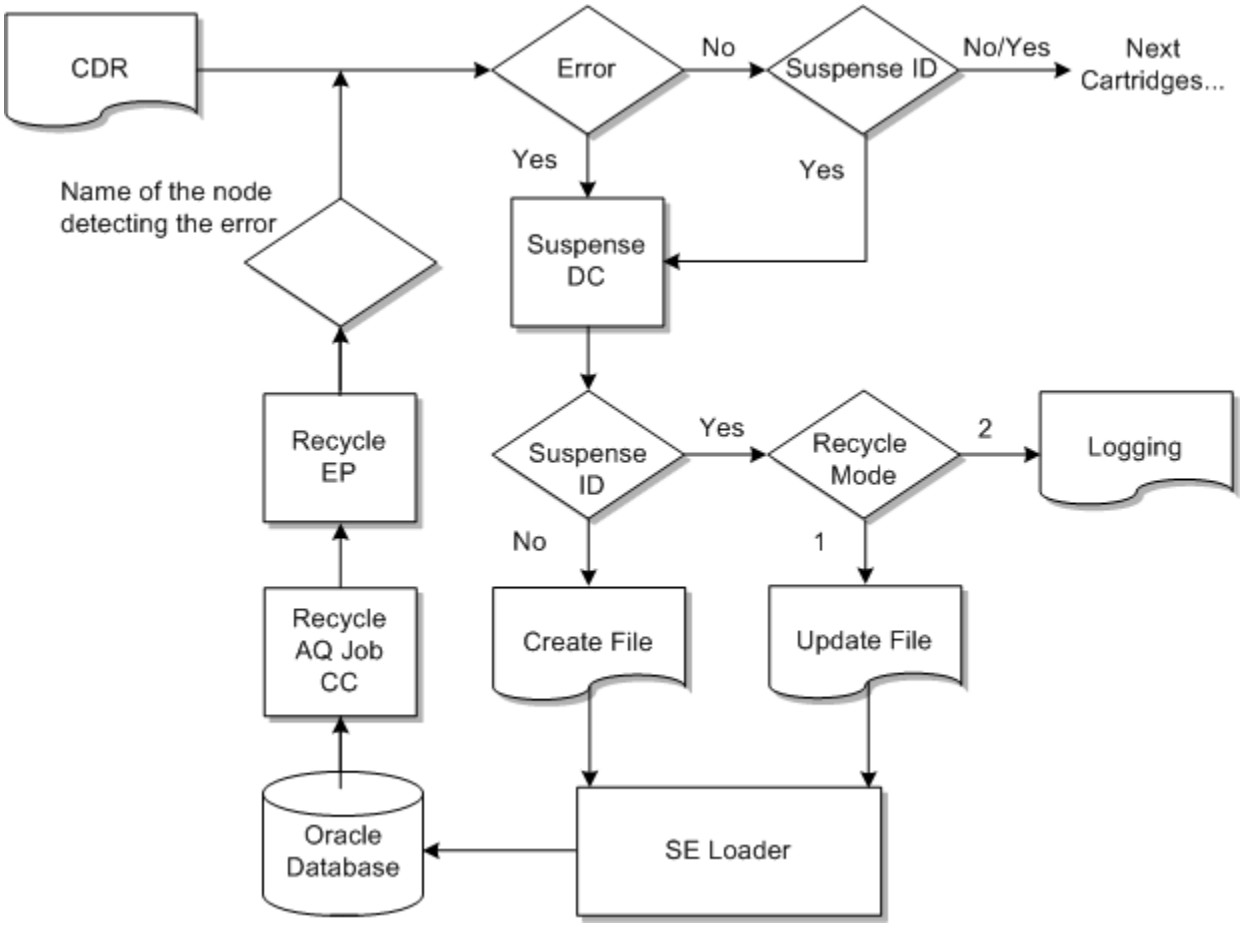
As nodes in the node chain process individual CDRs and batch CDR files, any CDRs or batch CDR files that contain errors should be made available for suspense handling. For more information on Offline Mediation Controller suspense handling, see "About Suspending and Recycling Call Detail Records".

Decision Points

The Offline Mediation Controller suspense decision points need to be considered in order to implement a cartridge that detects errors.

Figure 3–1 shows the event CDR flow with suspense handling decision points.

Figure 3–1 Event Suspense Handling Error Decision Points



Event Flow Decision Points

The decision points for event suspense handling are as follows:

1. The CDR enters the cartridge for processing, where:
   - If no errors are detected, the CDR passes through to the next cartridge in the processing flow of the node chain or is outputted to the target system.
   - If no errors are detected and the CDR contains the suspense ID field (previously passed through the suspense flow):
     a. The CDR passes through to the next cartridge in the processing flow of the node chain or is outputted to the target system.
     b. The CDR's input stream, error code, cartridge name, cartridge category, and the suspense ID are made available for the Suspense Distribution Cartridge (DC) node.
   - If an error is detected, the CDR's input stream, error code, cartridge name, and cartridge category are made available for the Suspense DC node.
   - If an error is detected and the CDR contains a suspense ID field (previously passed through the suspense flow), the CDR's input stream, error code, cartridge name, cartridge category, and the suspense ID are made available for the Suspense DC node.

   For a list of field names used in suspense handling, see "Suspense Fields".

2. The Suspense DC node receives the suspended or recycled CDR, where:
   - If the suspended CDR does not contain a suspense ID field (a suspended CDR), the Suspense DC node:
     a. Changes the status to a suspended state, which displays as **Suspended** in Suspense Management Center, a component of BRM.
     b. Generates a **Create** file, which is in a format understood by Suspended Event (SE) Loader, which is a component of BRM.
   - If the suspended CDR contains a suspense ID field (a recycled CDR), the Suspense DC node:
     a. Changes the status to a suspended state, which displays as **Suspended** in Suspense Management Center.
     b. Generates an **Update** file, which is in a format understood by SE Loader.
   - If the suspended CDR contains a suspense ID field (a recycled CDR) and no longer contains an error, the Suspense DC node:
     a. Changes the status to a succeeded state, which displays as **Succeeded** in Suspense Management Center.
     b. Generates an **Update** file, which is in a format understood by SE Loader.

Batch Flow Decision Points

The decision points for batch suspense handling are as follows:

1. The batch CDR file enters the batch-file-based collection cartridge (CC) for processing, where:
   - If no errors are detected, the CDRs in the batch CDR file pass through to the next cartridge in the processing flow of the node chain.
   - If no errors are detected and the batch CDR file was resubmitted (previously passed through the suspense flow):
     a. The CDRs in the batch CDR file pass through to the next cartridge in the processing flow of the node chain.
     b. Information on the resubmitted batch CDR file is made available for the Suspense DC node.
   - If an error is detected, the batch CDR file is put into an error directory and information on the batch CDR file is made available for the Suspense DC.
   - If an error is detected and the batch CDR file was resubmitted (previously passed through the suspense flow), information on the resubmitted batch CDR file is made available for the Suspense DC.
2. The Suspense DC node receives information on the suspended or resubmitted batch CDR file and queries the SUSPENDED_BATCH_T table with the batch key for existing information on the batch CDR file, where:
   - If no suspended batch CDR file information exists, the Suspense DC node:
     a. Changes the status to a suspended state, which displays as **Suspended** in Suspense Management Center.
     b. Generates a **Create** file, which is in a format understood by Suspended Batch (SB) Loader, which is a component of BRM.

- If suspended batch CDR file information exists (resubmitted batch CDR), the Suspense DC node:
  a. Changes the status to a suspended state, which displays as **Suspended** in Suspense Management Center.
  b. Generates an **Update** file, which is in a format understood by SB Loader.
- If suspended batch CDR file information exists and the batch CDR file no longer contains an error, the Suspense DC node:
  a. Changes the status to a succeeded state, which displays as **Succeeded** in Suspense Management Center.
  b. Generates an **Update** file, which is in a format understood by SB Loader.

Note: Each suspended batch CDR file is identified by the composite value of the following fields:

- DETAIL:ASS_SUSPENSE_EXT:SOURCE_FILENAME, which is the input file.

  This field refers to the SUSPENDED_BATCH_T.INPUT_FILE column.
- DETAIL:ASS_SUSPENSE_EXT:PIPELINE_NAME, which is the cartridge name that detected the error.

  This field refers to the SUSPENDED_BATCH_T.PIPELINE_NAME column.

For a list of field names used in suspense handling, see "Suspense Fields".

Determining Your CDR Errors and Error Codes

Before implementing a cartridge that will detect CDRs or batch CDR files with errors, you need to decide what those errors are and which error codes to use from the BRM error code list.

The BRM error codes and corresponding reason codes can be found in the *BRM_Home*/**sys/data/config/pin_suspense_reason_code** file and the *BRM_Home*/**sys/data/config/pin_batch_suspense_reason** file, where *BRM_Home* is the directory in which BRM is installed.

The reason descriptions for the error codes are listed in the **suspense_reason_code.en_US** file and the **batch_suspense_reason_code.en_US** file, which can be found in the *BRM_Home*/**sys/msgs/suspense_reason_code** directory.

Note: Localized versions of these files are available.

The Suspense DC maps error codes to suspense reasons, which can be divided into more specific suspense subreasons:

- For event suspense handling, suspense reasons and subreasons are mapped to the error codes in the **Create** and **Update** output files.
- For batch suspense handling, only suspense reasons are mapped to the error codes in the **Create** and **Update** output files.

You search for suspended event and batch CDRs using the error, reason, or subreason codes in Suspense Management Center.

To determine your CDR errors and error codes:

1. Decide what errors your cartridge needs to detect.
2. Check the *BRM_Home*/**sys/data/config/pin_suspense_reason_code** file and the *BRM_Home*/**sys/data/config/pin_batch_suspense_reason** file for the suspense handling error and reason codes that correspond to the errors you decided on in step 1.
3. Incorporate the BRM error code in the cartridge that you have created to detect errors. For NPL examples, see "Error Detecting NPL Examples".

For specific business requirements, you can create new reasons and subreasons. New suspense reasons and subreasons must contain a valid suspense error code. For more information, see the discussion on changing reasons and subreasons in the BRM documentation.

Error Detecting NPL Examples

The following examples describe the NPL rule files for cartridges that detect event and batch CDR errors. Both examples use Java hook methods. For more information on Java hooks, see "Working with Suspense Java Hooks in NPL".

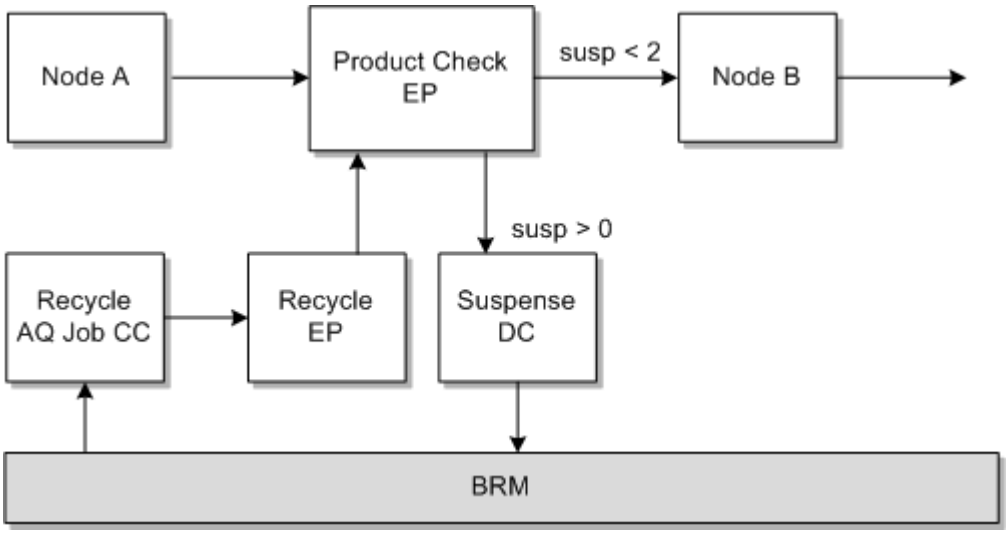
Product Type Error Event NPL Example

This example describes an NPL file for a node that detects and suspends event CDRs if the product type is incorrect.

Figure 3–2 shows the processing flow of a CDR in a node chain that contains three nodes:

1. Node A, whose output stream is received by the Product Check EP node.
2. Product Check EP node, which checks to see if the product type is TEL or SMS.
3. Node B, which if no errors are detected, receives the input stream from the Product Check EP node.

Figure 3–2 Product Type Error Detection Flow



In the Product Check EP node, configure the NPL with a flag value that outputs the CDR to one of the following:

- If the product type is not valid, the CDR is made available for the Suspense DC node, which changes the status to a suspended state and generates a **Create** file for the SE Loader.
- If the product type is valid, the CDR outputs to Node B.
- If the product type is valid and the CDR is recycled, the CDR is made available for the Suspense DC node, which changes the status to a succeeded state and generates an **Update** file for SE Loader.

Table 3–1 describes the NPL configuration for the Product Check EP node:

Table 3–1 Product Check EP NPL

NPL	Description
`JavaHook suspHandler=oracle.communications.brm.nm.nplhook.SuspenseMethodHandlerImpl;`	Java hook declaration.
`InputRec { String product_type; String calling_number; } in;`	Input stream from Node A.
`OutputRec { String cdr_service; String eventType; String A_NUMBER;`	Output stream to Node B.
`// internal flag Integer pSUSPENDED; } out;`	The routing Flag.
`Expose for Routing { out.pSUSPENDED "susp"; }`	Expose the routing flag and set the display name.
`// initialize routing flag to 0 out.pSUSPENDED=0;`	Initialize the routing flag to **0**, which if no errors are found sends the CDR to Node B.
`if (in.product_type != "TEL" && in.product_type!="SMS") { // bad prodcut type detected // 1) prepare output stream // 2) set suspended routing flag logInfo("inside wrong product type in.product_type= "+in.product_type); suspHandler.assemble(in, out, 5002, "ProductCheck", "ASCII");`	If the product type is incorrect, call the **assemble** Java hook method to clone the input stream to the output stream and add suspense fields for the given error code of **5002**, cartridge name **ProductCheck**, and cartridge category **ASCII**.
`out.pSUSPENDED=2;`	Set the routing flag to **2**, which directs the suspended CDR to the Suspense DC for suspense handling.

Table 3–1 (Cont.) Product Check EP NPL

NPL	Description

`} else {`
`  // good product type`
`  logInfo("good product type in.product_type=`
`  "+in.product_type);`
 	
`  // prepare output stream by assigning input values`
`  if(in.product_type=="TEL"){`
`  out.cdr_service="VOICE";`
`  out.eventType="USAGE";`
`  }else if(in.product_type=="SMS"){`
`  out.cdr_service="SMS";`
`  out.eventType="SMS_USAGE";`
`  }`
`  out.A_NUMBER=in.calling_number;`
	If the product type is correct, prepare the output stream based on the input stream.

`  // if this NAR was previously suspended (from`
`  recycle)`
`  //  1) add suspense fields to output stream`
`  //  2) set suspended routing flag`
`  if (suspHandler.isRecycled(in)>0) {`
`  logInfo("key suspense fields found, copying..." );`
`  suspHandler.append(in, out);`
	If the product type is correct and any suspense fields exist, such as, SUSPENSE_ ID, append the suspense fields to the output stream.

`  out.pSUSPENDED=1;` `  }`	Set the routing flag to **1**, which directs the CDR to: **1.** Node B to continue processing. **2.** Suspense DC, which updates the status to Successful.
`  }` `write(out);`	Writes out the stream.

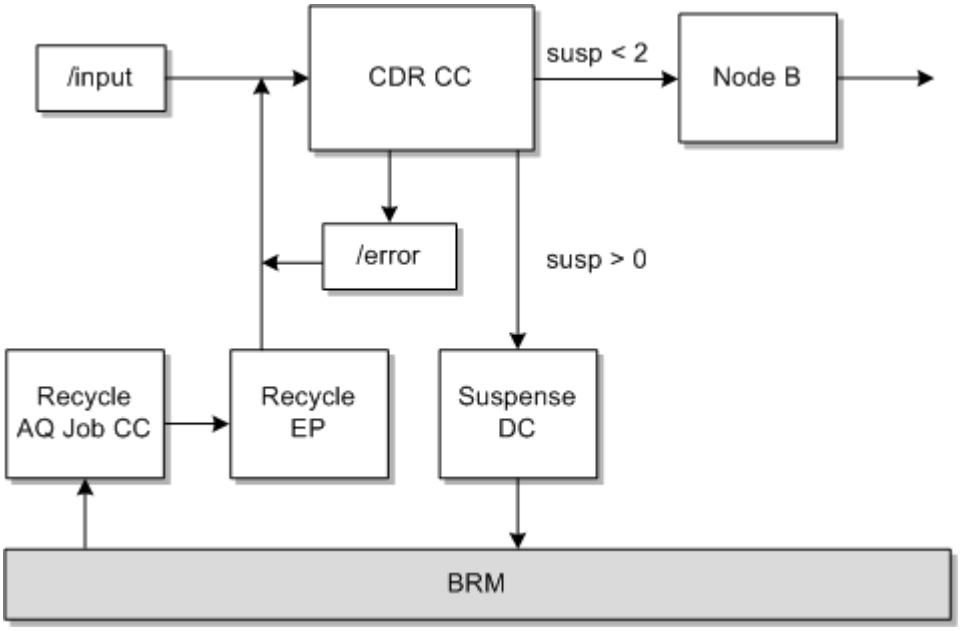
Batch CDR File Error NPL Example

This example describes an NPL file for a node that detects and suspends a batch CDR file if it contains an error.

Figure 3–3 shows the processing flow of a batch CDR file in a node chain that contains two nodes:

1. CDR CC node, which is responsible for processing input files received from the **input** directory.
2. Node B, which if no errors are detected, receives the input stream from the CDR CC node.

Figure 3–3 Batch CDR File Error Detection Flow



In the CDR CC node you configure the NPL with a flag value that outputs the CDR to one of the following:

- If no suspended batch CDR file information exists, the CDR CC node moves the batch CDR file to an **error** directory and makes information on the suspended batch CDR file available for the Suspense DC node. The Suspense DC node changes the status to a suspended state and generates a **Create** file for SB Loader.
- If the batch CDR file is valid, data processing continues to Node B.
- If suspended batch CDR file information exists, the Suspense DC node changes the status to a succeeded state and generates an **Update** file for SB Loader.

Table 3–2 describes the NPL configuration for the CDR CC node:

Table 3–2 CDR CC NPL

NPL	Description
`JavaHook suspHandler=oracle.communications.brm.nm.nplhook.SuspenseMethodHandlerImpl;`	Java hook declaration.
`InputRec { String sourceFileName; Byte rejected; Byte resubmitted; } in;`	Input stream from the **input** directory. **Note:** Not all the fields in the input stream are declared.
`OutputRec { Integer pSUSPENDED; } out;`	Declares the routing Flag. **Note:** Not all the fields in the output stream are declared.

Table 3–2 (Cont.) CDR CC NPL

NPL	Description

`Expose for Routing {`
`  out.pSUSPENDED "susp";`
`}`
	Expose the routing flag and set the display name.

`/// local variables`
`String pipelineName;`
`String pipelineCategory;`
`String sourceDir;`
`String errorDir;`
`String sourcePrefix;`
`String sourceSuffix;`
`String targetPrefix;`
`String targetSuffix;`
`Integer count;`
 	
`pipelineName="CDR Pipeline";`
`pipelineCategory="Wireless";`
`sourceDir="/input";`
`errorDir="batch/error";`
`sourcePrefix="test_";`
`sourceSuffix="edr";`
`targetPrefix="err_";`
`targetSuffix="bad";`
	Declare and initialize local variables.

`// initialize routing flag to 0` `out.pSUSPENDED=0;`	Initialize the routing flag to **0**.

`if (in.rejected > 0) {`
`  // file was rejected`
`  logInfo("in rejected");`
`  // 1) move files from input directory to error`
`  directory`
`  count=SuspHandler.move(sourceDir, sourcePrefix,`
`  sourceSuffix, errorDir, targetPrefix,`
`  targetSuffix);`
`  // 2) create batch nar`
`  SuspHandler.assemble(out,`
`  sourcePrefix+in.sourceFileName+sourceSuffix,`
`  pipelineName,`
`  127,`
`  errorDir,`
`  targetPrefix+in.sourceFileName+targetSuffix,`
`  "Acme Wireless",`
`  "control10001",`
`  "01",`
`  "tap processing info",`
`  pipelineCategory);`
	If the batch CDR file is invalid, move the batch CDR file to an error directory and call the **assemble** Java hook method to add the suspended batch file information to the output stream.

`  // 3) set suspended flag` `  out.pSUSPENDED=2;`	Sets the suspended Flag to **2**.

Table 3–2 (Cont.) CDR CC NPL

NPL	Description

`} else {`
`  // file is good, output input stream, and set flag`
`  logInfo("in good");`
`  out=clone(in);`
	If the input file is valid, prepare the output stream based on the input stream.

`  // if this file was previously suspended`
`  (from resubmit/recycle),`
`  // 1) add suspense fields to output stream`
`  // 2) set suspended routing flag`
`  if (in.resubmitted > 0) {`
	If the input file is valid and was previously suspended, append the suspense fields to the output stream.

`  // send 0 error code to flag success`
`  SuspHandler.assemble(out,`
`  sourcePrefix+in.sourceFileName+sourceSuffix,`
`  pipelineName,`
`  0);`
`  out.pSUSPENDED=1;`
`  }`
	Set the suspended flag to **1**.

`  }` `write(out);`	Writes out the stream.

4

Creating and Configuring the Suspense and Recycle Cartridges

This chapter describes how to create and configure Oracle Communications Offline Mediation Controller suspense and recycle cartridges for suspense handling of call detail records (CDRs).

Before reading this chapter, you should be familiar with:

- Oracle Communications Billing and Revenue Management (BRM) suspense handling and the BRM suspense components outlined in "Integrating Offline Mediation Controller with BRM".
- Offline Mediation Controller cartridge concepts and Node Programming Language (NPL).

About the Suspense and Recycle Cartridges

Offline Mediation Controller suspense handling is used for failed event or batch CDRs, where:

- An event action processes individual CDRs.
- A batch action processes a file that contains a collection of CDRs.

The following cartridges are used for both event and batch suspense handling:

- Suspense Distribution Cartridge (DC).

  The Suspense DC receives suspended CDRs or information on suspended batch CDR files, and generates **Create** and **Update** files in a format that is understood by Suspended Event (SE) Loader, or Suspended Batch (SB) Loader, which are components of BRM. For more information on how to create and configure the Suspense DC, see "About the Suspense Distribution Cartridge".

- Recycle AQ Job Collection Cartridge (CC).

  The Recycle AQ Job CC receives and parses the job ID message of the recycle request of the suspended event or batch CDRs that are submitted for reprocessing from Suspense Management Center or the **pin_recycle** utility, which are components of BRM. For more information on how to create and configure the Recycle AQ Job CC, see "About the Recycle AQ Job Collection Cartridge".

- Recycle Enhancement Processor Cartridge (EP).

  The Recycle EP uses the job ID message from the Recycle AQ Job CC to restore the recycled CDR or the resubmitted batch CDR for reprocessing or resubmission in Offline Mediation Controller. For more information on how to create and

configure the Recycle EP, see "About the Recycle Enhancement Processor Cartridge".

About the Suspense Distribution Cartridge

The Suspense DC does the following for event suspended CDRs:

1. Receives the suspended CDR input stream, error code, cartridge name, cartridge category, and (if the CDR is recycled) the suspense ID.

   Where:

   - suspended CDR input streams refers to the CDR input sent to the node that detected the error.
   - error code refers to the DETAIL:ASS_SUSPENSE_EXT:ERROR_CODE field.
   - cartridge name refers to the DETAIL:ASS_SUSPENSE_EXT:PIPELINE_NAME field.
   - cartridge category refers to the DETAIL:ASS_SUSPENSE_EXT:PIPELINE_CATEGORY field.
   - suspense ID refers to DETAIL:ASS_SUSPENSE_EXT:SUSPENSE_ID field.

   For a full list of Offline Mediation Controller suspense fields, see "Suspense Fields".

2. Serializes the suspended CDR with event data record (EDR) field mapping where applicable, into XML format.

   Note: When EDR field mapping is configured, serialization reduces the length of the EDR fields by replacing the EDR name with an ID. For more information on EDR field mapping, see "Integrating EDR Field Mapping".

3. Maps reason and subreason codes, which are based on error codes.

4. Generates **Create** (new suspended CDR) or **Update** (recycled suspended CDR) files in a format that is understood by SE Loader. For a full list of the **Create** and **Update** record type fields, see "Suspense DC Event Output".

   The format of the **Create** file consists of four record types:

   - **010** record type, which contains information such as, the CDR's record number, sender, and the time of creation.
   - **020** record type, which contains information such as, the suspense reason code, suspense subreason code, error code, cartridge name and cartridge category.
   - **030** record type, which contains the suspended CDR in a specialized BRM XML format, which is used by Suspense Management Center to correct and recycle the suspended CDR.
   - **040** record type, which contains the values of queryable fields. For more information on queryable fields, see "About Queryable Fields".

   The format of the **Update** file consists of two record types:

   - **010** record type, which contains information such as, the time of creation and the event type.

- **020** record type, which contains information such as, the suspense reason code, suspense subreason code, error code, suspense ID, and status.

For more information on batch suspense handling by the Suspense DC, see "About the Suspense DC for Batch Mode".

Creating a Suspense Distribution Cartridge Node

To create a Suspense DC node:

1. Log on to Administration Client.

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

2. In the **Mediation Hosts** table, select a host on which to create the Suspense DC node.

3. In the **Nodes on Mediation Host** section, click **New**.

   The Create a Node dialog box appears.

4. Select **Cartridge Kit** and click **Next**.

5. Select **Distribution Cartridge (DC)** and click **Next**.

6. Select **Suspense Distribution 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, select one of the following node programming language (NPL) rule files that matches your input data and apply changes where applicable:

   - For a NPL rule file that contains no field mapping and that can be used as a template, select **Generic**.
   - For a NPL rule file that contains ASCII and EDR field mapping, select **ASCIIEdr**.
   - For a NPL rule file that contains ASN and EDR field mapping, select **ASNEdr**.
   - For a NPL rule file that contains IMS input to output and selected IMS and EDR field mapping, select **IMS**.
   - For a NPL rule file that contains IMS and EDR field mapping, select **IMSEdr**.
   - For specific business requirements, select **New**.

   For instructions on how to configure a NPL rule file for the Suspense DC node, see "Configuring the NPL Rule File for the Suspense DC Node".

9. In the **Node Configuration** section, select each tab and configure the node's settings.

   For more information on the Suspense DC node tabs and their fields, see "About the Suspense DC Node Configuration Tabs".

10. Click **Save**, which saves the node.

About the Suspense DC Node Configuration Tabs

The Suspense DC node contains the following tabs:

- **General**, which contains general information for log files, reports, and data input and output for the node. For more information, see "Configuring General Information Settings for the Suspense DC Node".

- **Create File Output**, which contains the **Create** output file settings for the node. For more information, see "Configuring the Create File Output Settings for the Suspense DC Node".
- **Update File Output**, which contains the **Update** output file settings for the node. For more information, see "Configuring the Update File Output Settings for the Suspense DC Node".
- **Database Info**, which contains settings for Java database connectivity (JDBC) information that the node uses to access error reasons and EDR field mappings from BRM. For more information, see "Configuring JDBC Settings for the Suspense DC Node".
- **Mode**, which contains batch and individual CDR settings for the node. For more information, see "Configuring Batch and Event Settings for the Suspense DC Node".

Configuring General Information Settings for the Suspense DC Node

To configure general information settings for the Suspense DC node:

1. Verify the node is open.
2. In the **Node Configuration** section, click the **General** tab.
3. From the **Debug** list, select the level of debugging detail for the node's log file as follows:
   - To generate all Critical, Major, Minor, Warning, Informational, and Trace logs, select **OFF**.
   - To generate all Critical, Major, Minor, Warning, Informational, Trace logs, and if a Java exception occurs, generates a Java Stack Trace for the exception object, select **ON**.
4. In the **Max Log File Size** field, enter the maximum size, in bytes, for the log file. The minimum value is 50000 bytes and the maximum value is 2 gigabytes.

   This field allows you to control the size of the log file, before the node closes the file and opens a new one.
5. If you require the statistics reporting feature, select the **Enable Statistics** check box.
6. If you require bulk reading and writing of network accounting record (NAR) files, select the **Enable bulk read/write** check box.

   The bulk reading and writing function:
   - Reads and stores each intermediate NAR file as a whole into memory for processing by the respective node.
   - Writes the NAR output file as a whole into memory, until either the maximum number of NARs for a file is reached or the idle write time has expired, before writing the output data to a cache file.
7. In the **Read Timer** field, enter the amount of time, in seconds, that the node waits before checking for incoming data. The minimum value is 1 and the maximum value is 3600.
8. If you require the node to backup each processed NAR file, select the **Backup NAR Files** check box.

   Offline Mediation Controller keeps the NAR files in the backup directory for the period of time you specify in the **NAR File Retention Period** field.

9. In the **NAR File Retention Period** field, enter the number of days to retain the backup NAR files.
10. If you require the node to monitor the input stream of records and trigger an alarm if there is no input for the set interval specified in the **Interval** field, select the **Input Stream Monitoring** check box.

    The node will clear the alarm when it begins to receive records again.
11. In the **Interval** field, enter a time, in days, hours, or minutes that the node waits for new records before the node raises an alarm.
12. Click **Save**.

Configuring the Create File Output Settings for the Suspense DC Node

To configure the **Create** file output settings for the Suspense DC node:

1. Verify the node is open.
2. In the **Node Configuration** section, click the **Create File Output** tab.
3. In the **Output Directory** field, enter the directory name where Offline Mediation Controller stores the **Create** files produced by the Suspense DC node.
4. In the **File Name** field, enter the prefix for the **Create** output file.
5. In the **Current File Extension** field, enter the file extension of the preprocessed file.
6. In the **Processed File Extension** field, enter the file extension of the processed file.
7. From the **Output Push Time Unit** list:
   - To close the file when the size reaches the number of records entered in the **Max Records Per File** field, select **NONE**.
   - To close the file after an amount of time, select a time unit.
8. In the **Output Push Time Period** field, enter the duration of time, that the node waits before closing a file and creating a new file.

   The time unit you selected in the **Output Push Time Unit** list dictates the minimum and maximum values. For example, if you selected minutes, the minimum is 1 and the maximum is 60.
9. In the **Max Records Per File** field, enter the maximum number of records that can be entered into an output file before it is closed. The minimum value is 1 and the maximum value is 200000.
10. Click **Save**.

Configuring the Update File Output Settings for the Suspense DC Node

To configure the **Update** file output settings for the Suspense DC node:

1. Verify the node is open.
2. In the **Node Configuration** section, click the **Update File Output** tab.
3. In the **Output Directory** field, enter the directory name where Offline Mediation Controller stores the **Update** files produced by the Suspense DC node.
4. In the **File Name** field, enter the prefix for the **Update** output file.
5. In the **Current File Extension** field, enter the file extension of the preprocessed file.
6. In the **Processed File Extension** field, enter the file extension of the processed file.

7. From the **Output Push Time Unit** list:
   - To close the file when the size reaches the number of records entered in the **Max Records Per File** field, select **NONE**.
   - To close the file after an amount of time, select a time unit.
8. In the **Output Push Time Period** field, enter the duration of time, that the node waits before closing a file and creating a new file.

   The time unit you selected in the **Output Push Time Unit** list dictates the minimum and maximum values. For example, if you selected minutes, the minimum is 1 and the maximum is 60.
9. In the **Max Records Per File** field, enter the maximum number of records that can be entered into an updated output file before it is closed. The minimum value is 1 and the maximum value is 200000.
10. Click **Save**.

Configuring JDBC Settings for the Suspense DC Node

To configure JDBC settings for the Suspense DC node:

1. Verify the node is open.
2. In the **Node Configuration** section, click the **Database Info** tab.
3. In the **Batch Size** field, enter the number of records to retrieve from the Oracle database for each read operation.
4. In the **User** field, enter a valid username for accessing the Oracle database.
5. In the **Password** field, enter the password for the username.
6. In the **Verify Password** field, re-enter the password used in the **Password** field, which verifies that the password value was entered correctly.
7. In the **Host** field, enter the Oracle database server host name or IP address.
8. In the **Port** field, enter the listener port number used to communicate with the database server. The number must be a positive integer.
9. In the **SID** field, enter the system ID for the database server.
10. Click **Save**.

Configuring Batch and Event Settings for the Suspense DC Node

To configure batch and event settings for the Suspense DC node:

1. Verify the node is open.
2. In the **Node Configuration** section, click the **Mode** tab.
3. In the **Default Source Node** field, enter a designated name for the cartridge that suspended the CDR. If the Suspense DC node cannot find the name of the cartridge in the suspended CDR, this name is used.
4. In the **Default Category** field, enter a designated category name. This can be the name for a group of cartridges that suspend CDRs. If the Suspense DC node cannot find a category name in the suspended CDR, this name is used.
5. If your Suspense DC node is configured for suspended batch records, select the **Suspended Batch Mode** check box. For more information on suspended batch records, see "Working with Suspense Handling in Batch Mode".

6. (Optional) In the **EDR Field Mapping Name** field, enter a valid EDR field mapping name. The value refers to the EDR_FIELD_MAPPING_T.NAME. For more information on EDR field mapping and locating the EDR field mapping name, see "About EDR Field Mapping".
7. If your Suspense DC node is configured for suspended event records, in the **Create Event Type** field, enter the storable class name for the suspended CDR.
8. If your Suspense DC node is configured for suspended event records, in the **Update Event Type** field, enter the storable class name for the recycled CDR.
9. If your Suspense DC node is configured for suspended batch records, in the **Storable Class** field, enter the storable class name for the suspended batch CDR.
10. Click **Save**.

Configuring the NPL Rule File for the Suspense DC Node

The Suspense DC node uses the NPL rule file primarily to set the configuration values for JDBC and queryable tables, and to map the fields for the queryable tables.

To configure the NPL rule file for the Suspense DC node:

1. Verify the node is open.
2. From the **Rule File** list, select one of the sample NPL rule files, or select **New**.

   For a list of sample NPL rule files available, see "Creating a Suspense Distribution Cartridge Node".
3. Click **Edit**.

   The NPL Editor dialog box appears.
4. Enter a configuration clause and configuration values to set JDBC and your queryable tables:

   ```
   Config {
   JDBCDriver "JDBC_Driver";
   JDBCUrl "JDBC_URL";
   QueryableTables "Table_name1,Table_name2,..";
   }
   ```

   where:

   - *JDBC_Driver*, defines the JDBC driver.
   - *JDBC_URL*, defines the JDBC URL and includes the JDBC host, port and SID tokens, whose values are configured in the **Database Info** configuration tab of the Suspense DC node.
   - *Table_name* is a list of tables (separated by a comma) that contain the columns for queryable fields. These tables must exist in the BRM database.

   For example:

   ```
   Config {
   JDBCDriver "oracle.jdbc.driver.OracleDriver";
   JDBCUrl "jdbc:oracle:thin:@%DBHOST%:%DBPORT%:%DBSID%";
   QueryableTables "SUSP_USAGE_TELCO_INFO_T,SUSP_USAGE_TELCO_GSM_INFO_T";
   }
   ```

5. Enter an **Expose for** clause and mapping for the queryable table fields:

   ```
   Expose for Table_name {
   ```

```
  out.CDR_field_name "Column_name";
}
```

where:

- *Table_name* is the name of a table that you provided in the **QueryableTables** configuration clause.
- *CDR_field_name* is the name of a CDR field, which must be declared in the NPL rule file.
- *Column_name* is the name of a column that must exist in the *Table_name* table.

  For example:

```
Expose for SUSP_USAGE_TELCO_INFO_T
{
  out.DETAIL:VOLUME_RECEIVED "BYTES_IN";
  out.DETAIL:VOLUME_SENT "BYTES_OUT";
  out.DETAIL:B_NUMBER "CALLED_TO";
  out.DETAIL:A_NUMBER "CALLING_FROM";
  out.DETAIL:DURATION "CALL_DURATION";
  out.DETAIL:A_NUMBER "PRIMARY_MSID";
  out.DETAIL:BASIC_SERVICE "SERVICE_TYPE";
  out.DETAIL:CHARGING_START_TIMESTAMP "START_TIME";
  out.DETAIL:USAGE_TYPE "USAGE_TYPE";
}
```

  For more information on queryable fields, see "About Queryable Fields".

6. According to your business requirements, continue to apply edits to the NPL rule file.
7. When you have finished making changes, from the **File** menu, select **Save**.

   The Save As... dialog box appears.
8. In the **Display Name** field, enter a new GUI display name for the NPL rule file.
9. In the **Rules File Name** field, enter a new NPL rule file name.
10. Click **Save**.
11. To close the NPL Editor dialog box, from the **File** menu, select **Exit**.
12. From the **Rule File** list, verify your NPL rule file is in the list, and select it.
13. Click **Save**.

For specific business requirements other rules can be set. For more information about NPL rule files, see *Offline Mediation Controller Cartridge Development Kit NPL Reference Guide*.

Java hooks for suspense handling allow you to call Java methods from the NPL rule file. For more information on the Java hooks available for suspense handling, see "Working with Suspense Java Hooks in NPL". For more information on using Java hooks with NPL rule files, see the discussion on Java hooks in *Offline Mediation Controller Cartridge Development Kit NPL Reference Guide*.

About Queryable Fields

A queryable field corresponds to a column in an existing table in the BRM database. Suspense Management Center allows you to:

- Search for suspended CDRs based on values in queryable fields and displays those values in your search results.
- Analyze and correct queryable fields in suspended CDRs.

For more information on queryable fields, see the discussion on selecting a list of queryable EDR fields in the BRM documentation.

Once you have decided on what queryable fields to use in your implementation, in the Suspense DC NPL rule file, set the configuration values and map the queryable fields for your queryable tables. For more information, see "Configuring the NPL Rule File for the Suspense DC Node".

The output mapping for queryable fields are in the record type **010** of the **Create** output file. For more information, see "Suspense DC Event Output".

The **Create** format consists of a list of editable table names and corresponding column names as follows:

```
Table_name{Column_name:Data_type;Column_name:Data_type;....}
Table_name{Column_name:Data_type;....}
```

where:

- *Table_name* is the name of a table that must exist in the BRM database.
- *Column_name* is the name of a column that must exist in the *Table_name* table.
- *Data_type* is determined by the EDR field mapping for the CDR field. If EDR field mapping is not used then the CDR field type is used.

For example:

```
SUSP_USAGE_TELCO_GSM_INFO_T{CELL_ID:STRING;DESTINATION_SID:STRING;
DIALED_NUMBER:STRING;ORIGIN_SID:STRING;SECONDARY_MSID:STRING}
SUSP_USAGE_TELCO_INFO_T{BYTES_IN:DECIMAL;BYTES_OUT:DECIMAL;CALLED_TO:STRING;
CALLING_FROM:STRING;CALL_DURATION:DECIMAL;PRIMARY_MSID:STRING;
SERVICE_TYPE:STRING;START_TIME:DATE;USAGE_TYPE:STRING}
```

For more information on queryable fields, see the BRM documentation.

About EDR Field Mapping

Note: Oracle recommends EDR field mapping if you use Suspense Manager.

In Offline Mediation Controller, the NPL rules for a cartridge allows values from the fields in the input stream to be assigned to the fields in the output stream, which are sent to the downstream cartridges or external medium for processing.

The Suspense DC node serializes the CDR based on the output field names from the NPL mapping. Shorter designations may be used in place of the output field names by using EDR field mapping, which reduces the length of field names by replacing the field name with an ID. Whatever mapping is defined in the Suspense DC node, the reverse is required for the Recycle EP node.

The **EDR Field Mapping Name** field in the **Mode** tab of the Suspense DC node, refers to the database EDR_FIELD_MAPPING_T.NAME and the EDR field mapping is stored in the EDR_FLD_MAP_BUF_T table. Both tables are installed during a BRM

installation or when you run the **load_edr_field_mapping** utility. Before using EDR field mapping, verify that these tables exists.

An EDR field mapping XML file is loaded into the database during installation of Suspense Manager. You use the **load_edr_field_mapping** utility, a BRM component, to add or modify existing EDR field names and load new versions of the EDR field mapping XML file.

For more information on the **load_edr_field_mapping** utility and EDR field mapping, see "Integrating EDR Field Mapping" and the BRM documentation.

About the Recycle AQ Job Collection Cartridge

The Recycle AQ Job CC does the following:

1. Polls and checks the Oracle AQ queue for recycle request notifications.
2. Retrieves the job ID message information contained within the Oracle AQ event message.
3. Parses the job ID message.
4. Outputs the job ID message.

For more information on batch suspense handling by the Recycle AQ Job CC, see "About the Recycle AQ Job CC for Batch Mode".

For more information on working with the Oracle AQ queue message processing, see the discussion on the account synchronization DM in the BRM documentation.

Creating a Recycle AQ Job Collection Cartridge Node

To create a Recycle AQ Job CC node:

1. Log on to Administration Client.

   The Node Hosts & Nodes (logical view) screen appears.
2. In the **Mediation Hosts** table, select a host on which to create the Recycle AQ Job CC node.
3. In the **Nodes on Mediation Host** section, click **New**.

   The Create a Node dialog box appears.
4. Select **Cartridge Kit** and click **Next**.
5. Select **Collection Cartridge (CC)** and click **Next**.
6. Select **Recycle AQ Job Collection 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, select the **RecycleAQJob** rule file.

   For instructions on how to configure a NPL rule file for the Recycle AQ Job CC node, see "Configuring the NPL Rule File for the Recycle AQ Job CC Node".
9. In the **Node Configuration** section, select each tab and configure the node's settings.

   For more information on the Recycle AQ Job CC node tabs and their fields, see "About the Recycle AQ Job CC Node Configuration Tabs".

10. Click **Save**, which saves the node.

About the Recycle AQ Job CC Node Configuration Tabs

The Recycle AQ Job CC node contains the following tabs:

- **General**, which contains general information for log files, reports, and data input and output for the node. For more information, see "Configuring General Information Settings for the Recycle AQ Job CC Node".
- **Database Info**, which contains settings for JDBC information that the node uses to access the BRM database. For more information, see "Configuring JDBC Settings for the Recycle AQ Job CC Node".
- **DeQueue**, which contains dequeuing settings for the node. For more information, see "Configuring Dequeuing Settings for the Recycle AQ Job CC Node".

Configuring General Information Settings for the Recycle AQ Job CC Node

To configure general information settings for the Recycle AQ Job CC node:

1. Verify the node is open.
2. In the **Node Configuration** section, click the **General** tab.
3. From the **Debug** list, select the level of debugging detail for the node's log file as follows:
   - To generate all Critical, Major, Minor, Warning, Informational, and Trace logs, select **OFF**.
   - To generate all Critical, Major, Minor, Warning, Informational, Trace logs, and if a Java exception occurs, generates a Java Stack Trace for the exception object, select **ON.**
4. In the **Max Log File Size** field, enter the maximum size, in bytes, for the log file. The minimum value is 50000 and the maximum value is 2 gigabytes.

   This field, allows you to control the size of the log file, before the node closes the file and opens a new one.
5. If you require the statistics reporting feature, select the **Enable Statistics** check box.
6. If you require bulk reading and writing of NAR files, select the **Enable bulk read/write** check box.

   The bulk reading and writing function:
   - Reads and stores each intermediate NAR file as a whole into memory for processing by the respective node.
   - Writes the NAR output file as a whole into memory, until either the maximum number of NARs for a file is reached or the idle write time has expired, before writing the output data to a cache file.
7. In the **NARs Per File** field, enter the maximum number of NARs you require in an output file. The minimum value is 1 and the maximum value is 10000.
8. In the **Idle Write Time** field, enter the number of seconds the node waits before transferring the NAR file to the input directory of the destination node, whether or not it has reached its maximum size. The minimum value is 1 and the maximum value is 3600.
9. Click **Save**.

Configuring JDBC Settings for the Recycle AQ Job CC Node

To configure JDBC settings for the Recycle AQ Job CC node:

1. Verify the node is open.
2. In the **Node Configuration** section, click the **Database Info** tab.
3. In the **Batch Size** field, enter the number of records to retrieve from the Oracle database for each read operation.
4. In the **User** field, enter a valid username for accessing the Oracle database.
5. In the **Password** field, enter the password for the username.
6. In the **Verify Password** field, re-enter the password used in the **Password** field, which verifies that the password value was entered correctly.
7. In the **Host** field, enter the Oracle database server host name or IP address.
8. In the **Port** field, enter the listener port number used to communicate with the database server. The number must be a positive integer.
9. In the **SID** field, enter the system ID for the database server.
10. Click **Save**.

Configuring Dequeuing Settings for the Recycle AQ Job CC Node

To configure dequeuing settings for the Recycle AQ Job CC node:

1. Verify the node is open.
2. In the **Node Configuration** section, click the **DeQueue** tab.
3. In the **Queue** field, enter the name of the queue, from which the Recycle AQ Job CC node dequeues the job ID message. For more information on Oracle AQ queues see "About Oracle AQ Messaging for Suspense Handling" and the BRM documentation.
4. In the **Event** field, enter the name of the event that is used to dequeue messages as follows:
   - For suspended event CDRs, enter **RecycleRequest**.
   - For suspended batch CDRs, enter **ResubmitBatchRequest**.
5. If you require polling of the dequeue operation, select the **Enable Polling** option. By default, the **enqueue** notification starts the dequeue operation of the message if it matches the configured event entered in the **Event** field.
6. In the **Polling Interval** field, enter the time, in days, hours, or minutes, that the node waits before checking for the message to dequeue.
7. Click **Save**.

Configuring the NPL Rule File for the Recycle AQ Job CC Node

The Recycle AQ Job CC node uses the NPL rule file primarily to set the configuration values for JDBC.

To configure the NPL rule file for the Recycle AQ Job CC node:

1. Verify the node is open.
2. From the **Rule File** list, select **New.**

For a list of sample NPL rule files available, see "Creating a Recycle AQ Job Collection Cartridge Node".

3. Click **Edit**.

   The NPL Editor dialog box appears.

4. Enter a configuration clause and configuration values to set JDBC:

```
Config {
JDBCDriver "JDBC_Driver";
JDBCUrl "JDBC_URL";
}
```

   where:

   - *JDBC_Driver*, defines the JDBC driver.
   - *JDBC_URL*, defines the JDBC URL and includes the JDBC host, port and SID tokens, whose values are configured in the **Database Info** configuration tab of the Recycle AQ Job CC node.

   For example:

```
Config {
JDBCDriver "oracle.jdbc.driver.OracleDriver";
JDBCUrl "jdbc:oracle:thin:@%DBHOST%:%DBPORT%:%DBSID%";
}
```

5. According to your business requirements, continue to apply edits to the NPL rule file.
6. When you have finished making changes, from the **File** menu, select **Save**.

   The Save As... dialog box appears.

7. In the **Display Name** field, enter a new GUI display name for the NPL rule file.
8. In the **Rules File Name** field, enter a new NPL rule file name.
9. Click **Save**.
10. To close the NPL Editor dialog box, from the **File** menu, select **Exit**.
11. From the **Rule File** list, verify your NPL rule file is in the list, and select it.
12. Click **Save**.

For more information about NPL rule files, see *Offline Mediation Controller Cartridge Development Kit NPL Reference Guide*.

Java hooks for suspense handling allow you to call Java methods from the NPL rule file. For more information on the Java hooks available for suspense handling, see "Working with Suspense Java Hooks in NPL". For more information on using Java hooks with NPL rule files, see the discussion on Java hooks in *Offline Mediation Controller Cartridge Development Kit NPL Reference Guide*.

About the Recycle Enhancement Processor Cartridge

The Recycle EP does the following for event suspended CDRs:

1. Retrieves the recycled CDR based on the SQL statements stored in the detail SQL and header SQL files.
2. De-serializes the BRM XML for recycled CDRs with EDR field mapping, where applicable.

3. Applies edit changes to the restored CDR.

For more information on batch suspense handling by the Recycled EP, see "About the Recycle EP for Batch Mode".

Creating a Recycle Enhancement Processor Cartridge Node

To create a Recycle EP node:

1. Log on to Administration Client.

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

2. In the **Mediation Hosts** table, select a host on which to create the Recycle EP node.
3. In the **Nodes on Mediation Host** section, click **New**.

   The Create Node dialog box appears.

4. Select **Cartridge Kit** and click **Next**.
5. Select **Enhancement Processor (EP)** and click **Next**.
6. Select **Recycle 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, select one of the following NPL rule files that matches your input data and apply changes where applicable:
   - For a NPL rule file that contains no field mapping and that can be used as a template, select **Generic**.
   - For a NPL rule file that contains ASCII and EDR reverse mapping, select **ASCIIEdr**.
   - For a NPL rule file that contains ASN and EDR reverse mapping, select **ASNEdr**.
   - For a NPL rule file that contains the restored IMS and selected IMS and EDR reverse mapping, select **IMS**.
   - For a NPL rule file that contains IMS and EDR reverse mapping, select **IMSEdr**.
   - For a NPL rule file to restore batch CDR files, select **Batch**.
   - For specific business requirements, select **New**

   For instructions on how to configure a NPL rule file for the Recycle EP node, see "Configuring the NPL Rule File for the Recycle EP Node".
9. In the **Node Configuration** section, select each tab and configure the node's settings.

   For more information on the Recycle EP node tabs and their fields, see "About the Recycle Enhancement Processor Cartridge Node Configuration Tabs".
10. Click **Save**, which saves the node.

About the Recycle Enhancement Processor Cartridge Node Configuration Tabs

The Recycle EP node contains the following tabs:

- **General**, which contains general information for log files, reports, and data input and output for the node. For more information, see "Configuring General Information Settings for the Recycle EP Node".
- **Database Info**, which contains settings for JDBC information that the node uses to access recycled CDRs. For more information, see "Configuring JDBC Settings for the Recycle EP Node".
- **SQL Files**, which contains settings for the cartridge category value and for the files containing SQL statements, which are used by the Recycle EP node to retrieve recycled CDRs. For more information, see "Configuring SQL Settings for the Recycle EP Node".

Configuring General Information Settings for the Recycle EP Node

To configure general information settings for the Recycle EP node:

1. Verify the node is open.
2. In the **Node Configuration** section, click the **General** tab.
3. From the **Debug** list, select the level of debugging detail for the node's log file as follows:
   - To generate all Critical, Major, Minor, Warning, Informational, and Trace logs, select **OFF**.
   - To generate all Critical, Major, Minor, Warning, Informational, Trace logs, and if a Java exception occurs, generates a Java Stack Trace for the exception object, select **ON**.
4. In the **Max Log File Size** field, enter the maximum size, in bytes, for the log file. The minimum value is 50000 and the maximum value is 2 gigabytes.

   This field, allows you to control the size of the log file, before the node closes the file and opens a new one.
5. If you require the statistics reporting feature, select the **Enable Statistics** check box.
6. If you require bulk reading and writing of NAR files, select the **Enable bulk read/write** check box.

   The bulk reading and writing function:
   - Reads and stores each intermediate NAR file as a whole into memory for processing by the respective node.
   - Writes the NAR output file as a whole into memory, until either the maximum number of NARs for a file is reached or the idle write time has expired, before writing the output data to a cache file.
7. In the **Read Timer** field, enter the amount of time, in seconds, that the node waits before checking for incoming data. The minimum value is 1 and the maximum value is 3600.
8. In the **NARs Per File** field, enter the maximum number of NARs you require in an output file. The minimum value is 1 and the maximum value is 10000.
9. In the **Idle Write Time** field, enter the number of seconds the node waits before transferring the NAR file to the input directory of the destination node, whether or not it has reached its maximum size. The minimum value is 1 and the maximum value is 3600.

10. If you require the node to backup each processed NAR file, select the **Backup NAR Files** check box.

    Offline Mediation Controller keeps the NAR files in the backup directory for the period of time you specify in the **NAR File Retention Period** field.

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

12. If you require the node to monitor the input stream of records and trigger an alarm if there is no input for the set interval specified in the **Interval** field, select the **Input Stream Monitoring** check box.

    The node will clear the alarm when it begins to receive records again.

13. In the **Interval** field, enter a time, in days, hours, or minutes that the node waits for new records before the node raises an alarm.

14. Click **Save**.

Configuring JDBC Settings for the Recycle EP Node

To configure JDBC settings for the Recycle EP node:

1. Verify the node is open.
2. In the **Node Configuration** section, click the **Database Info** tab.
3. In the **Batch Size** field, enter the number of records to retrieve from the Oracle database for each read operation.
4. In the **User** field, enter a valid username for accessing the Oracle database.
5. In the **Password** field, enter the password for the username.
6. In the **Verify Password** field, re-enter the password used in the **Password** field, which verifies that the password value was entered correctly.
7. In the **Host** field, enter the Oracle database server host name or IP address.
8. In the **Port** field, enter the listener port number used to communicate with the database server. The number must be a positive integer.
9. In the **SID** field, enter the system ID for the database server.
10. Click **Save**.

Configuring SQL Settings for the Recycle EP Node

To configure SQL settings for the Recycle EP node:

1. Verify the node is open.
2. In the **Node Configuration** section, click the **SQL Files** tab.
3. (Optional) In the **Category** field, enter the category name. This can be the ID name for a group of cartridges. This name is used to retrieve recycled CDRs.
4. (Optional) In the **SQL Header File** field, enter the full path and filename to the SQL header file. For more information, see "Working with SQL Files to Restore a Recycled CDR".
5. In the **SQL Detail File** field, enter the full path and filename to the SQL detail file. For more information, see "Working with SQL Files to Restore a Recycled CDR".
6. Click **Save**.

Configuring the NPL Rule File for the Recycle EP Node

The Recycle EP node uses the NPL rule file primarily to set the configuration values for JDBC.

To configure the NPL rule file for the Recycle EP Node:

1. Verify the node is open.
2. From the **Rule File** list, select one of the sample NPL rule files, or select **New.**

   For a list of sample NPL rule files available, see "Creating a Recycle Enhancement Processor Cartridge Node".
3. Click **Edit**.

   The NPL Editor dialog box appears.
4. Enter a configuration clause and configuration values to set JDBC:

   ```
   Config {
   JDBCDriver "JDBC_Driver";
   JDBCUrl "JDBC_URL";
   }
   ```

   where:

   - *JDBC_Driver*, defines the JDBC driver.
   - *JDBC_URL*, defines the JDBC URL and includes the JDBC host, port and SID tokens, whose values are configured in the **Database Info** configuration tab of the Recycle EP node.

   For example:

   ```
   Config {
   JDBCDriver "oracle.jdbc.driver.OracleDriver";
   JDBCUrl "jdbc:oracle:thin:@%DBHOST%:%DBPORT%:%DBSID%";
   }
   ```
5. According to your business requirements, continue to apply edits to the NPL rule file.
6. When you have finished making changes, from the **File** menu, select **Save**.

   The Save As... dialog box appears.
7. In the **Display Name** field, enter a new GUI display name for the NPL rule file.
8. In the **Rules File Name** field, enter a new NPL rule file name.
9. Click **Save**.
10. To close the NPL Editor dialog box, from the **File** menu, select **Exit**.
11. From the **Rule File** list, verify your NPL rule file is in the list, and select it.
12. Click **Save**.

For more information about NPL rule files, see *Offline Mediation Controller Cartridge Development Kit NPL Reference Guide*.

Java hooks for suspense handling allow you to call Java methods from the NPL rule file. For more information on the Java hooks available for suspense handling, see "Working with Suspense Java Hooks in NPL". For more information on using Java hooks with NPL rule files, see the discussion on Java hooks in *Offline Mediation Controller Cartridge Development Kit NPL Reference Guide*.

Working with SQL Files to Restore a Recycled CDR

The Recycle EP node restores the recycled CDRs based on structured query language (SQL) statements.

Two types of SQL statements are used by the Recycle EP node:

- Detail SQL, which is required.
- Header SQL, which is optional.

The following sample SQL statement files are provided with the Recycle EP jar file:

- To retrieve serialized suspended CDRs in ASCII format, use the **RecycleDetail.sql** file.
- To retrieve serialized suspended CDRs in EDR, use the **RecycleEdr.sql** file.
- To retrieve serialized suspended CDRs in IMS format, which includes selected EDR common fields, use the **RecycleIMS.sql** file.
- To retrieve header information, use the **RecycleHeader.sql** file, which is a generic SQL statement.

  The data from the Header SQL is set in every recycled CDR retrieved by the detail SQL statement.

Installing the Sample Detail and Header SQL Files

To install the sample detail and header SQL files:

1. Go to *OMC_Home*/**web/htdocs** directory, where *OMC_Home* is the directory in which Offline Mediation Controller is installed.
2. Extract the SQL files from the jar file by running the following command:

   ```
   jar xvf Recycle.jar sql/RecycleDetail.sql sql/RecycleEdr.sql sql/RecyleIMS.sql
   sql/RecycleHeader.sql
   ```

   The SQL files are placed in an **sql** directory.

Updating and Applying a Sample SQL File in the Recycle EP Node

To update and apply a sample SQL file in the Recycle EP node:

1. Choose one of the sample SQL statement files that best represents the format of the CDR before suspension and update according to your business requirements. For more information, see "SQL Statement Examples".

   For more information on configuring SQL statements, see the SQL documentation.

   Note: The following variables may be used in the SQL statement:

   - ${RECYCLE_JOB_ID}, which is replaced by the job ID message sent by the AQ listener.
   - ${PIPELINE_CATEGORY}, which is replaced by the name entered into the **Category** field in the **SQL** configuration tab of the Recycle EP node. A NULL value is given if the **Category** field is left blank.

2. In the **SQL Header File** and **SQL Detail File** fields in the **SQL Files** tab of the Recycle EP node, enter the full path and filename to the SQL header and SQL detail files. For more information, see "Configuring SQL Settings for the Recycle EP Node".

For more information on working with SQL statements, see the SQL documentation.

SQL Statement Examples

The following example shows a detail SQL statement without editable fields:

```
SELECT   b.poid_id0, b.pipeline_name, r.recycle_mode, '', b.batch_id,
         b.suspended_from_batch_id, b.recycle_key, e.edr_buf
FROM     suspended_usage_t b,
         susp_usage_edr_buf e,
         susp_usage_recycle_t r
WHERE    b.poid_id0 = e.obj_id0
  AND    b.recycle_obj_id0 = r.obj_id0
  AND    b.recycle_obj_id0 = ${RECYCLE_JOB_ID}
  AND    NVL(b.pipeline_category,'NULL') = ${PIPELINE_CATEGORY}
ORDER BY b.poid_id0
```

The following example shows a detail SQL statement with editable fields:

```
SELECT  a.poid_id0, a.pipeline_name, a.recycle_mode, a.override_reasons,
        a.batch_id, a.suspended_from_batch_id, a.recycle_key, a.edr_buf,
        a.editable_flds
FROM
 ( SELECT b.poid_id0 , b.pipeline_name, r.recycle_mode,
          r.override_reasons,b.batch_id, b.suspended_from_batch_id,
          b.recycle_key, e.edr_buf,
          'CalledId='|| t.calling_from|| CHR (9)||
          'calling_number='||t.called_to|| CHR (9)||
          'duration='|| t.call_duration|| CHR (9)||
          'start_time='|| t.start_time|| CHR (9)||
          'product_type='|| t.usage_type|| CHR (9)||
          'requestedInputVolume='|| t.bytes_in|| CHR (9)||
          'requestedOutputVolume='|| t.bytes_out|| CHR (9)||
          'cell_id='|| g.cell_id as editable_flds
FROM      suspended_usage_t b,
          susp_usage_telco_info_t t,
          susp_usage_telco_gsm_info_t g,
          susp_usage_edr_buf e,
          susp_usage_recycle_t r
WHERE     b.poid_id0 = t.obj_id0
     AND  b.poid_id0 = e.obj_id0
     AND  b.recycle_obj_id0 = r.obj_id0
     AND  b.poid_id0 = g.obj_id0
     AND  b.recycle_obj_id0 = ${RECYCLE_JOB_ID}
     AND  NVL(b.pipeline_category,'NULL') = ${PIPELINE_CATEGORY}) a
ORDER BY a.poid_id0
```

Working with Suspense Handling in Batch Mode

Offline Mediation Controller supports batch suspense handling for suspended and resubmitted batch CDR files.

The following cartridges are used for batch suspense handling:

- Suspense DC.

  The Suspense DC receives information on the suspended batch CDR file and generates **Update** and **Create** files in a format that is understood by Suspended Batch (SB) Loader.

- Recycle AQ Job CC.

The Recycle AQ Job CC receives and parses the job ID message for the suspended batch that was resubmitted from Suspense Management Center or the **pin_recycle** utility.

- Recycle EP.

  The Recycle EP uses the job ID message from the Recycle AQ Job CC to retrieve the resubmitted batch CDR file information and to return the suspended batch CDR file, from the error directory, back to the input stream of the batch-file-based CC for reprocessing in Offline Mediation Controller.

About the Suspense DC for Batch Mode

After receiving the suspended batch CDR file information from the batch-file-based CC node the Suspense DC does the following for suspended batch CDR files:

1. Maps the suspense reason code, which is based on the error code.
2. Generates **Create** (suspended batch) or **Update** (resubmitted batch) files in a format that are understood by SB Loader. For a full list of the **Create** and **Update** record type fields, see "Suspense DC Batch Output".

   The format of the **Create** file consists of two record types:

   - **010** record type, which contains information on the storable class.
   - **020** record type, which contains information such as, the suspense reason code (based on the error code), error code, cartridge name and cartridge category.

   The format of the **Update** file consists of two record types:

   - **010** record type, which contains information on the storable class.
   - **030** record type, which contains information on the batch CDR file name, cartridge name, suspense reason (based on the error code), error code, and suspense status.

Creating a Suspense DC Node for Batch Mode

To create a Suspense DC node for batch mode, follow the steps outlined in "Creating a Suspense Distribution Cartridge Node".

Configuring the Suspense DC Node Configuration Tabs for Batch Mode

The Suspense DC node contains the following tab to configure specifically for batch mode:

- **Mode**, which contains batch and event settings for the node. For more information, see "Configuring Batch Mode Settings for the Suspense DC Node".

For the **General**, **Create File Output**, **Update File Output** and **Database Info** tabs, follow the steps outlined in:

- Configuring General Information Settings for the Suspense DC Node
- Configuring the Create File Output Settings for the Suspense DC Node
- Configuring the Update File Output Settings for the Suspense DC Node
- Configuring JDBC Settings for the Suspense DC Node

Configuring Batch Mode Settings for the Suspense DC Node

To configure batch mode settings for the Suspense DC node:

1. Verify the node is open.
2. In the **Node Configuration** section, click the **Mode** tab.
3. In the **Default Source Node** field, enter a designated name for the cartridge that suspended the batch CDR file. If the Suspense DC node cannot find the name of the cartridge that suspended the batch CDR file, in the information on the suspended batch CDR file, this name is used.
4. In the **Default Category** field, enter a designated category name. This can be the name for a group of cartridges that suspend CDRs. If the Suspense DC node cannot find a category name in the information on the suspended batch CDR file, this name is used.
5. Select the **Suspended Batch Mode** check box.
6. In the **Storable Class** field, enter the storable class name for the suspended batch CDR file.
7. Click **Save**.

Configuring the Suspense DC Node NPL Rule File for Batch Mode

For batch mode, the Suspense DC node uses the NPL rule file primarily to set the configuration values for JDBC.

To configure the NPL rule file for the Recycle EP node:

1. Verify the node is open.
2. From the **Rule File** list, select one of the sample NPL rule files, or select **New.**

   For a list of sample NPL rule files available, see "Creating a Suspense Distribution Cartridge Node".
3. Click **Edit**.

   The NPL Editor dialog box appears.
4. Enter a configuration clause and configuration values to set JDBC:

   ```
   Config {
   JDBCDriver "JDBC_Driver";
   JDBCUrl "JDBC_URL";
   }
   ```

   where:

   - *JDBC_Driver*, defines the JDBC driver.
   - *JDBC_URL*, defines the JDBC URL and includes the JDBC host, port and SID tokens, whose values are configured in the **Database Info** configuration tab of the Suspense DC node.

   For example:

   ```
   Config {
   JDBCDriver "oracle.jdbc.driver.OracleDriver";
   JDBCUrl "jdbc:oracle:thin:@%DBHOST%:%DBPORT%:%DBSID%";
   }
   ```

5. According to your business requirements, continue to apply edits to the NPL rule file.
6. When you have finished making changes, from the **File** menu, select **Save**.

   The Save As... dialog box appears.
7. In the **Display Name** field, enter a new GUI display name for the NPL rule file.
8. In the **Rules File Name** field, enter a new NPL rule file name.
9. Click **Save**.
10. To close the NPL Editor dialog box, from the **File** menu, select **Exit**.
11. From the **Rule File** list, verify your NPL rule file is in the list, and select it.
12. Click **Save**.

For more information about NPL rule files, see *Offline Mediation Controller Cartridge Development Kit NPL Reference Guide*.

Java hooks for suspense handling allow you to call Java methods from the NPL rule file. For more information on the Java hooks available for suspense handling, see "Working with Suspense Java Hooks in NPL". For more information on using Java hooks with NPL rule files, see the discussion on Java hooks in *Offline Mediation Controller Cartridge Development Kit NPL Reference Guide*.

About the Recycle AQ Job CC for Batch Mode

The Recycle AQ job CC does the following:

1. Polls and checks the Oracle AQ queue for recycle request notifications.
2. Retrieves the job ID message information contained within the Oracle AQ event message.
3. Parses the job ID message.
4. Outputs the job ID message.

Creating a Recycle AQ Job CC Node for Batch Mode

To create a Recycle AQ Job CC node for batch mode, follow the steps outlined in "Creating a Recycle AQ Job Collection Cartridge Node".

Configuring the Recycle AQ Job CC Node Configuration Tabs for Batch Mode

The Recycle AQ Job CC node contains the following tab to configure specifically for batch mode:

- **DeQueue**, which contains dequeuing settings for the node. For more information, see "Configuring Batch Dequeuing Settings for the Recycle AQ Job CC Node".

For the **General** and **Database Info** tabs, follow the steps outlined in:

- Configuring General Information Settings for the Recycle AQ Job CC Node
- Configuring JDBC Settings for the Recycle AQ Job CC Node

Configuring Batch Dequeuing Settings for the Recycle AQ Job CC Node

To configure batch dequeuing settings for the Recycle AQ Job CC node:

1. Verify the node is open.

2. In the **Node Configuration** section, click the **DeQueue** tab.
3. In the **Queue** field, enter the name of the queue, from which the Recycle AQ Job CC node dequeues the job ID message. For more information on Oracle AQ queues see "About Oracle AQ Messaging for Suspense Handling" and the BRM documentation.
4. In the **Event** field, enter **ResubmitBatchRequest**, which is used to dequeue suspended batch messages.
5. If you require polling of the dequeue operation, select the **Enable Polling** option. By default, the **enqueue** notification starts the dequeue operation of the message if it matches the configured event entered in the **Event** field.
6. In the **Polling Interval** field, enter the time, in days, hours, or minutes, that the node waits before checking for the message to dequeue.
7. Click **Save**.

Configuring the Recycle AQ Job CC NPL Rule File for Batch Mode

For batch mode, the Recycle AQ Job CC node uses the NPL rule file primarily to set the configuration values for JDBC.

To configure the NPL rule file for the Recycle AQ Job CC node:

1. Verify the node is open.
2. From the **Rule File** list, select **New.**

   For a list of sample NPL rule files available, see "Creating a Recycle AQ Job Collection Cartridge Node".
3. Click **Edit**.

   The NPL Editor dialog box appears.
4. Enter a configuration clause and configuration values to set JDBC:

   ```
   Config {
   JDBCDriver "JDBC_Driver";
   JDBCUrl "JDBC_URL";
   }
   ```

   where:

   - *JDBC_Driver*, defines the JDBC driver.
   - *JDBC_URL*, defines the JDBC URL and includes the JDBC host, port and SID tokens, whose values are configured in the **Database Info** configuration tab of the Recycle AQ Job CC node.

   For example:

   ```
   Config {
   JDBCDriver "oracle.jdbc.driver.OracleDriver";
   JDBCUrl "jdbc:oracle:thin:@%DBHOST%:%DBPORT%:%DBSID%";
   }
   ```
5. According to your business requirements, continue to apply edits to the NPL rule file.
6. When you have finished making changes, from the **File** menu, select **Save**.

   The Save As... dialog box appears.

7. In the **Display Name** field, enter a new GUI display name for the NPL rule file.
8. In the **Rules File Name** field, enter a new NPL rule file name.
9. Click **Save**.
10. To close the NPL Editor dialog box, from the **File** menu, select **Exit**.
11. From the **Rule File** list, verify your NPL rule file is in the list, and select it.
12. Click **Save**.

For more information about NPL rule files, see *Offline Mediation Controller Cartridge Development Kit NPL Reference Guide*.

Java hooks for suspense handling allow you to call Java methods from the NPL rule file. For more information on the Java hooks available for suspense handling, see "Working with Suspense Java Hooks in NPL". For more information on using Java hooks with NPL rule files, see the discussion on Java hooks in *Offline Mediation Controller Cartridge Development Kit NPL Reference Guide*.

About the Recycle EP for Batch Mode

The Recycle EP node does the following for batch mode:

1. Retrieves the resubmitted batch CDR file information based on the job ID message.
2. Moves the suspended batch CDR file from the error directory back to the input stream of the batch-file-based CC in the node chain, based on actions configured in the Recycle EP node's NPL rule file.

Creating a Recycle EP Node for Batch Mode

To create a Recycle EP node for batch mode, follow the steps outlined in"Creating a Recycle Enhancement Processor Cartridge Node".

Configuring the Recycle EP Node Configuration Tabs for Batch Mode

The Recycle EP node does not contain specific batch mode settings. For the **General**, **Database Info** and **SQL Files** tabs, follow the steps outlined in:

- Configuring General Information Settings for the Recycle EP Node
- Configuring JDBC Settings for the Recycle EP Node
- Configuring SQL Settings for the Recycle EP Node

Configuring the Recycle EP NPL Rule File for Batch Mode

For batch mode, the Recycle EP node uses the NPL primarily to set the configuration values for JDBC and to move the suspended batch CDR file back to the input stream of the batch-file-based CC in your node chain.

To configure the NPL rule file for the Recycle EP node:

1. Verify the node is open.
2. From the **Rule File** list, select one of the sample NPL rule files, or select **New.**

   For a list of sample NPL rule files available, see "Creating a Recycle Enhancement Processor Cartridge Node".
3. Click **Edit**.

The NPL Editor dialog box appears.

4. Enter a configuration clause and configuration values to set JDBC:

```
Config {
JDBCDriver "JDBC_Driver";
JDBCUrl "JDBC_URL";
}
```

where:

- *JDBC_Driver*, defines the JDBC driver.
- *JDBC_URL*, defines the JDBC URL and includes the JDBC host, port and SID tokens, whose values are configured in the **Database Info** configuration tab of the Recycle EP node.

For example:

```
Config {
JDBCDriver "oracle.jdbc.driver.OracleDriver";
JDBCUrl "jdbc:oracle:thin:@%DBHOST%:%DBPORT%:%DBSID%";
}
```

5. Configure the NPL rule file to move the suspended batch CDR file from the error directory to the input stream of the batch-file-based CC in your node chain.

For example:

a. The input record variable fields include the following fields:

```
// input data
InputRec {
   Long   DETAIL:ASS_SUSPENSE_EXT:SUSPENSE_ID;
   String DETAIL:ASS_SUSPENSE_EXT:SOURCE_FILENAME;
   String DETAIL:ASS_SUSPENSE_EXT:ERROR_FILENAME;
   String DETAIL:ASS_SUSPENSE_EXT:ERROR_PATH;
   String DETAIL:ASS_SUSPENSE_EXT:PIPELINE_NAME;
   String HEADER:TAP_PROCESSING_INFO;
   String DETAIL:ASS_SUSPENSE_EXT:OVERRIDE_REASONS;
} in;
```

where:

– The DETAIL:ASS_SUSPENSE_EXT:ERROR_FILENAME field, is the name of the batch CDR file.

– The DETAIL:ASS_SUSPENSE_EXT:ERROR_PATH field, is the path to the error directory.

– The DETAIL:ASS_SUSPENSE_EXT:PIPELINE_NAME field, is the name of the cartridge that detected the error and suspended the batch CDR file.

b. Declare and initialize the local variables:

```
// local variables
String pipelineName;
String pipelineCategory;
String sourceFileName;
String sourceDir;
String errorDir;
String sourcePrefix;
String sourceSuffix;
String targetPrefix;
String targetSuffix;
```

```
Integer count;

pipelineName=in.DETAIL:ASS_SUSPENSE_EXT:PIPELINE_NAME;
pipelineCategory="Wireless";
sourceFileName=in.DETAIL:ASS_SUSPENSE_EXT:SOURCE_FILENAME;
sourceDir="batch/repo";
errorDir="batch/error";
sourcePrefix="err_";
sourceSuffix="bad";
targetPrefix="test_";
targetSuffix="edr";
```

c. Move the batch CDR file from the error directory to the input directory of the batch-file-based CC. The following example uses the Java hook **move** to move files. For more information on the move java hook method, see "move".

```
//move files from error directory to input directory
count=SuspHandler.move(sourceDir, sourcePrefix, sourceSuffix,
                       errorDir, targetPrefix, targetSuffix);
logInfo("moved from "+sourceDir+"("+sourcePrefix+","+sourceSuffix+") to
        "+errorDir+"("+targetPrefix+","+targetSuffix+").
                        result="+int2str(count));
```

For an NPL example of the CC node that suspended the batch CDR file, which this Recycle EP example restores, see "Batch CDR File Error NPL Example".

6. According to your business requirements, continue to apply edits to the NPL rule file.
7. When you have finished making changes, from the **File** menu, select **Save**.

   The Save As... dialog box appears.
8. In the **Display Name** field, enter a new GUI display name for the NPL rule file.
9. In the **Rules File Name** field, enter a new NPL rule file name.
10. Click **Save**.
11. To close the NPL Editor dialog box, from the **File** menu, select **Exit**.
12. From the **Rule File** list, verify your NPL rule file is in the list, and select it.
13. Click **Save**.

For more information about NPL rule files, see *Offline Mediation Controller Cartridge Development Kit NPL Reference Guide*.

Java hooks for suspense handling allow you to call Java methods from the NPL rule file. For more information on the Java hooks available for suspense handling, see "Working with Suspense Java Hooks in NPL". For more information on using Java hooks with NPL rule files, see the discussion on Java hooks in *Offline Mediation Controller Cartridge Development Kit NPL Reference Guide*.

5

Implementing Suspense Handling for the ECE Cartridge Pack

This chapter describes how to implement the Oracle Communications Offline Mediation Controller suspense handling for the Oracle Communications Offline Mediation Controller Oracle Communications Elastic Charging Engine (ECE) Cartridge Pack.

Before reading this chapter, you should be familiar with:

- Oracle Communications Billing and Revenue Management (BRM) suspense handling and the BRM suspense components outlined in "Integrating Offline Mediation Controller with BRM".
- Offline Mediation Controller cartridge concepts and Node Programming Language (NPL).
- The Offline Mediation Controller ECE Cartridge Pack. For more information, see *Offline Mediation Controller Elastic Charging Engine Cartridge Pack User Guide*.

About the Cartridges Used for Suspense Handling with the ECE Cartridge Pack

The following cartridges are used to implement Offline Mediation Controller suspense handling with the ECE cartridge pack:

- ECE Distribution Cartridge (DC).

  The ECE DC receives the CDR input stream, which includes the ECE response record from Oracle Communications Billing and Revenue Management Elastic Charging Engine, and checks for CDR errors. If an error is detected, it is sent to a **suspense** directory. For more information, see "About the ECE Distribution Cartridge and Suspense Handling".

  For more information on the ECE DC, see *Offline Mediation Controller Elastic Charging Engine Cartridge Pack User Guide*.

- Network Accounting Record (NAR) Collection Cartridge (CC).

  The NAR CC reads the suspended CDR input stream, error code, cartridge name, cartridge category, and (if the CDR is recycled) the suspense ID, from the **suspense** directory. The output is made available for the Suspense DC. For more information, see "About the Network Accounting Record Collection Cartridge and Suspense Handling".

- Suspense Distribution Cartridge (DC).

The Suspense DC receives suspended CDRs and generates **Create** or **Update** files in a format that is understood by Suspended Event (SE) Loader, which is a component of BRM. For more information on how to create and configure the Suspense DC, see "About the Suspense Distribution Cartridge".

- Recycle AQ Job Collection Cartridge (CC).

  The Recycle AQ Job CC receives and parses the job ID message of the recycle request of suspended CDRs that are submitted for reprocessing from Suspense Management Center or the **pin_recycle** utility, which are components of BRM. For more information on how to create and configure the Recycle AQ Job CC, see "About the Recycle AQ Job Collection Cartridge".

- Recycle Enhancement Processor Cartridge (EP).

  The Recycle EP uses the job ID message from the Recycle AQ Job CC to restore the recycled CDR for reprocessing by the ECE DC. For more information on how to create and configure the Recycle EP, see "About the Recycle Enhancement Processor Cartridge".

Suspense Handling Flow for the ECE Cartridge Pack

The following steps describe the Offline Mediation Controller suspense handling flow for the ECE cartridge pack:

1. The CDR input stream enters the ECE DC node, where:
   - If no errors are detected, the ECE DC creates usage requests based on the input stream, which are submitted to ECE for rating.
   - If an error is detected, the CDR input stream, error code, cartridge name, cartridge category, and (if the CDR is recycled) the suspense ID are written to a suspended CDR file in the *OMC_Home*/**suspense**/*node_ID* directory, where *OMC_Home* is the directory in which Offline Mediation Controller is installed, and *node_ID* is the name of the ECE DC node that suspended the CDR.
2. The NAR CC node receives the suspended CDR file from the **suspense** directory and its output is made available for the Suspense DC node.
3. The Suspense DC node receives the suspended CDRs and generates **Create** or **Update** files in a format that is understood by SE Loader.
4. The **Create** or **Update** files are received by the SE Loader application, which is used to load suspended events into **/suspended_usage** objects in the BRM database.
5. The suspended CDR, which is in a suspended state, is fixed, submitted for recycling, and its status updated to **Recycling** in Suspense Management Center. For more information on the Suspense Management Center statuses, see the BRM documentation.
6. The recycle request is sent to the Oracle AQ queue, by Suspense Management Center or the **pin_recycle** utility, which is a component of BRM.
7. The Recycle AQ Job CC node receives and parses the job ID message of the recycle request.
8. The Recycle EP node uses the job ID message from the Recycle AQ Job CC node to restore the recycled CDR for reprocessing in the ECE DC node.
9. The ECE DC node reprocesses the recycled CDR, where:
   - If no more errors are detected:

a. The recycled CDR is distributed to ECE for rating.

b. The recycled CDR goes back to the Suspense DC node, which changes the status to a succeeded state, and generates an **Update** file for the SE Loader.

- If more errors are detected:

a. The recycled CDR is sent back to the **suspense** directory.

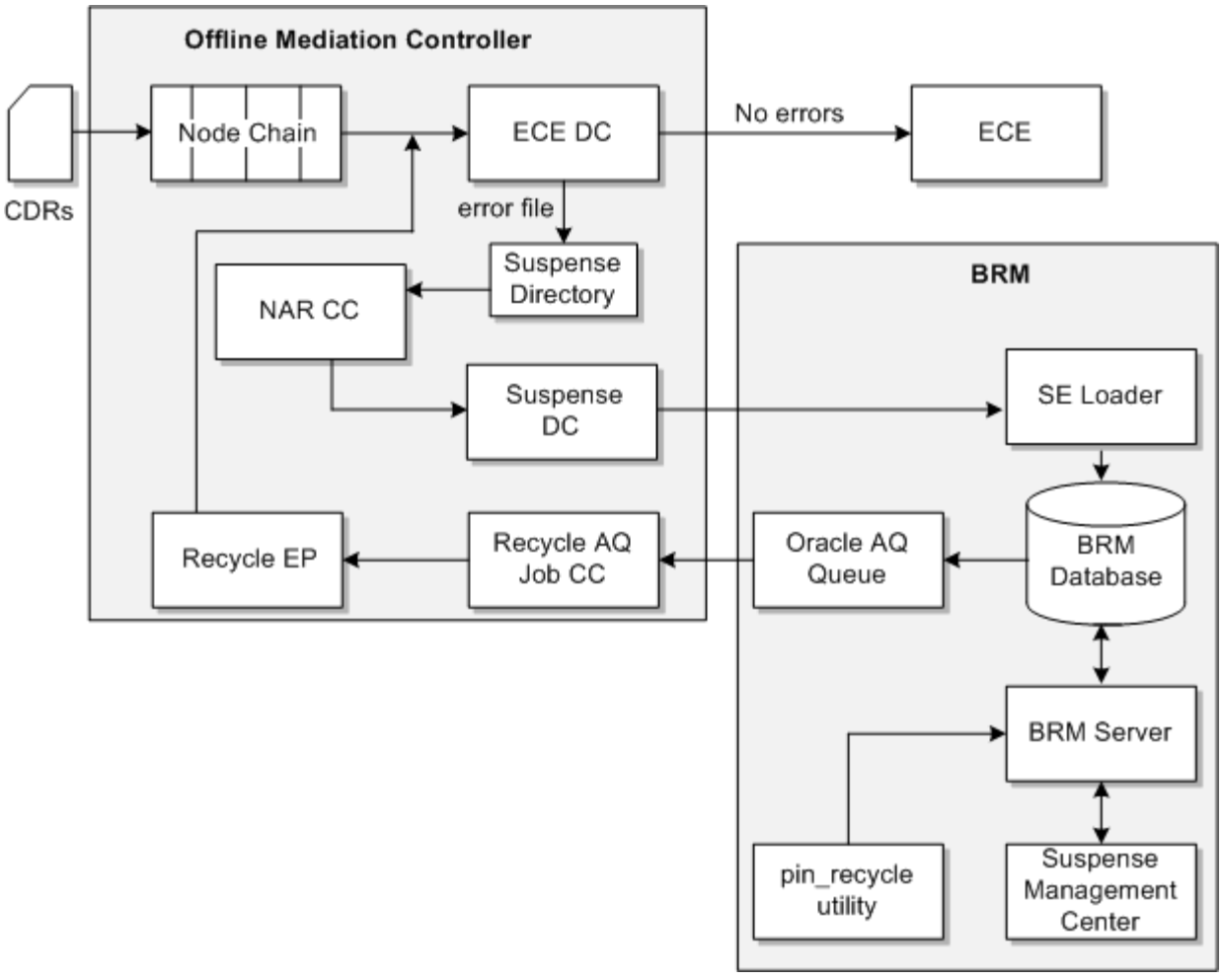
b. The NAR CC node receives the recycled CDR file from the **suspense** directory and its output is made available for the Suspense DC node.

c. The Suspense DC node receives the recycled CDR, which changes the status to a suspended state and generates an **Update** file for the SE Loader.

The recycled CDR continues through the Offline Mediation Controller suspense handling flow, until either it is removed (**Written off**) by Suspense Management Center, or it is successful (**Succeeded**).

Figure 5–1 shows the Offline Mediation Controller suspense handling flow for the ECE cartridge pack.

Figure 5–1 ECE Cartridge Pack Suspense Handling Flow



About the ECE Distribution Cartridge and Suspense Handling

The ECE DC receives the CDR input stream and reports CDR errors using the Offline Mediation Controller error codes. If an error is detected the suspended CDR is sent to the **suspense** directory.

Offline Mediation Controller error codes can be found in the *OMC_Home*/**config/nodemgr/ocomc_errorcodes.xml** file, where *OMC_Home* is the directory in which Offline Mediation Controller is installed. For more information, see "About Offline Mediation Controller Error Codes".

For more information on how to configure the ECE DC node to detect errors, see "Configuring Cartridges to Detect and Suspend Failed CDRs".

Creating an ECE DC Node for Suspense Handling

To create an ECE DC node for suspense handling:

1. Log on to Administration Client.

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

2. In the **Mediation Hosts** table, select a host on which to create an ECE DC node.
3. In the **Nodes on Mediation Host** section, click **New**.

   The Create a Node dialog box appears.

4. Select the **Wireless** service solution option and click **Next**.
5. Select **Distribution Cartridge (DC)** and click **Next**.
6. Select **OCECE** 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, select a rule file that matches your input data and apply changes where applicable.
9. In the **Node Configuration** section, select each tab and configure your nodes settings.

   For more information on the ECE DC node tabs and their fields, see *Offline Mediation Controller Elastic Charging Engine Cartridge Pack User Guide*.

10. Click **Save**, which saves the node.

Configuring the ECE DC Node Configuration Tabs for Suspense Handling

The ECE DC node contains the following tabs to configure for suspense handling:

- **General**, which contains general information for log files, reports, and data input and output for the node. For more information on configuring the **General** tab for a distribution cartridge, see "Configuring General Information Settings for the Suspense DC Node", and the discussion on creating and configuring the node in the *Offline Mediation Controller Elastic Charging Engine Cartridge Pack User Guide*.
- **OCECE Configuration**, which contains suspense file settings for the node. For more information, see "Configuring Suspense Settings for the ECE DC Node".

Configuring Suspense Settings for the ECE DC Node

To configure suspense settings for the ECE DC node:

1. Verify the node is open.
2. In the **Node Configuration** section, click the **OCECE Configuration** tab.
3. In the **Max records per suspense file** field, enter the number of records that can be added to the suspended CDR file before the file is closed. When the file contains the maximum number of records, it is moved into the *OMC_Home*/**suspense/***node_ID* directory for suspense handling.
4. Select the **Output bad records as NAR** check box to generate suspended CDRs in NAR format.
5. Click **Save**.

About the Network Accounting Record Collection Cartridge and Suspense Handling

The NAR CC reads the suspended CDR input stream, error code, cartridge name, cartridge category, and (if the CDR is recycled) the suspense ID from the **suspense** directory. The output is made available for the Suspense DC.

Creating a NAR CC Node for Suspense Handling

To create a NAR CC node for suspense handling:

1. Log on to Administration Client.

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

   The Create a Node dialog box appears.
4. Select the **Cartridge Kit** service solution option and click **Next**.
5. Select **Collection Cartridge (CC)** and click **Next**.
6. Select **Network Accounting Record Collection 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, select one of the following node programming language (NPL) rule files that matches your input data and apply changes where applicable:
   - For an NPL rule file that reads ECE DC suspended records for ASCII CDRs, select **ECEDC suspended ascii NAR input**.
   - For an NPL rule file that reads ECE DC suspended records for IMS CDRs, select **ECEDC suspended IMS NAR input**.
   - For an NPL rule file that reads ECE DC suspended records for SGSN CDRs, select **ECEDC suspended SGSN NAR input**.
   - For specific business requirements, select **New**.

   For more information on the NPL rule file mappings for the NAR CC node, see "NPL Rule File Mappings for the NAR CC Node".
9. In the **Node Configuration** section, select each tab and configure your node's settings.

10. Click **Save**, which saves the node.

Configuring the NAR CC Node Configuration Tabs for Suspense Handling

The NAR CC node contains the following tabs to configure for suspense handling:

- **General**, which contains general information for log files, reports, and data input and output for the node. For more information on configuring the **General** tab for a collection cartridge, see "Configuring General Information Settings for the Recycle AQ Job CC Node".
- **File Location**, which contains suspended CDR location settings. For more information, see "Configuring Suspended CDR Location Settings for the NAR CC Node".

Configuring Suspended CDR Location Settings for the NAR CC Node

To configure suspended CDR location settings for the NAR CC node:

1. Verify the node is open.
2. In the **Node Configuration** section, click the **File Location** tab.
3. Select the **Files are pushed to this node** option, which enables the data protocol to push data to the NAR CC node.
4. In the **File Parse Delay** field, enter the time in seconds, minutes, or hours that the node waits before checking for new files to process.
5. In the **Local Directory** field, enter the path and name of the directory that contains the suspended CDR file as follows, *OMC_Home*/**suspense/***node_ID*.
6. In the **Local File suffix** field, enter **.archive**, which is the suffix name used for suspense handling.
7. Select one of the following options:
   - To keep CDR files after processing, select **Keep Track of Processed Files?**.
   - To delete CDR files after processing, select **Delete Files After Processing?**.
8. Click **Save**.

NPL Rule File Mappings for the NAR CC Node

The NAR CC node uses the NPL rule file primarily to map the input stream, error code, cartridge name, cartridge category, and (if the CDR is recycled) the suspense ID to the output stream.

For specific business requirements other rules can be set. For more information about NPL rule files, see *Offline Mediation Controller Cartridge Development Kit NPL Reference Guide*.

Java hooks for suspense handling allow you to call Java methods from the NPL rule file. For more information on the Java hooks available for suspense handling, see "Working with Suspense Java Hooks in NPL". For more information on using Java hooks with NPL rule files, see the discussion on Java hooks in *Offline Mediation Controller Cartridge Development Kit NPL Reference Guide*.

About Offline Mediation Controller Error Codes

You incorporate error codes for CDR errors that you want to detect into your node. Error codes are mapped to suspense reasons and subreasons in the **Create** and **Update** output files by the Suspense DC. You can search for suspended event CDRs and batch CDR files using the error, reason and subreason codes in Suspense Management Center.

The Offline Mediation Controller error codes are specific to Offline Mediation Controller and inform Suspense Manager, which is a component of BRM, that the suspended CDR comes from Offline Mediation Controller. The Offline Mediation Controller error codes are stored in the *OMC_Home***/config/nodemgr/ocomc_errorcodes.xml** file.

For more information, see "Determining Your CDR Errors and Error Codes".

Configuring New Offline Mediation Controller Error Codes for Suspended CDRs

To configure new Offline Mediation Controller error codes for suspended CDRs:

1. Stop Node Manager.
2. Go to the *OMC_Home***/config/nodemgr** directory, where *OMC_Home* is the directory in which Offline Mediation Controller is installed.
3. Open the **ocomc_errorcodes.xml** file in a text editor.
4. Add suspended error codes according to your specific business requirements by using the following XML format:

```
<OCOMCErrorCodes>
<error id="1">
<code>60001</code>
<message>CANCEL_NOT_ALLOWED</message>
<ReasonCode>13</ReasonCode>
<SubReasonCode>0</SubReasonCode>
</error>
</OCOMCErrorCodes>
```

5. Save and close the file.
6. Restart Node Manager.

Important: Any new Offline Mediation Controller error codes must be:

1. Added in the *BRM_Home***/sys/data/config/pin_suspense_reason_code** and the *BRM_Home***/sys/msgs/suspense_reason_code/suspense_reason_code.en_US** files.
2. Loaded into the BRM database using the **load_pin_suspense_reason_code** and **load_localized_strings** utilities, which are components of BRM.

For more information, see the discussion on changing reasons and subreasons in the BRM documentation.

Working with the Offline Mediation Controller Error Code API

The **OCOMCErrorCode** Java API is used to access the errors defined in the **ocomc_errorcodes.xml** file. The following methods can be used:

- **OCOMCErrorCode.get(String errorMsg)**, this method will return the error code associated with the error message.
- **public static synchronized OCOMCError getError(String errorMessage)**, this method will return an instance of the **OCOMCError** object, which contains all the fields associated with the error.

Note: You do not need to create an instance of the **OCOMCErrorCode** class. Node Manager will add the class when it has been restarted.

6

Suspense Fields

This chapter provides reference information for the Oracle Communications Offline Mediation Controller suspense handling.

Suspense Fields

Table 6–1 lists the event and batch CDR fields that are used for Offline Mediation Controller suspense handling.

Table 6–1 Suspense CDR Fields

Field Name	Data Type	Suspense Type	Description
DETAIL:ASS_SUSPENSE_EXT:ACCOUNT_POID	String	event	The field is provided by the calling cartridge or the Oracle Communications Billing and Revenue Management (BRM) default root poid **0.0.0.1 /account 1 0**.
DETAIL:ASS_SUSPENSE_EXT:ERROR_CODE	Long	both	The error code. This is a required field and is provided by the calling cartridge.
DETAIL:ASS_SUSPENSE_EXT:ERROR_FILENAME	String	batch	(Optional) The field is provided by the calling cartridge.
DETAIL:ASS_SUSPENSE_EXT:ERROR_PATH	String	batch	The field is provided by the calling cartridge.
DETAIL:ASS_SUSPENSE_EXT:OVERRIDE_REASONS	String	both	(Optional) The field is provided by the calling cartridge.
DETAIL:ASS_SUSPENSE_EXT:PIPELINE_CATEGORY	String	both	(Optional) Identifies that the cartridge is a member of a group of cartridges. If the Suspense DC cannot find the category name, the name entered in the **Default Category** field of the **Mode** configuration tab, is used. The field is provided by the calling cartridge.

Table 6–1 (Cont.) Suspense CDR Fields

Field Name	Data Type	Suspense Type	Description
DETAIL:ASS_SUSPENSE_EXT:PIPELINE_NAME	String	both	The name of the cartridge that suspended the CDR. If the Suspense DC node cannot find the cartridge name, the name entered in the **Default Source Node** field of the **Mode** configuration tab is used. The field is provided by the calling cartridge.
DETAIL:ASS_SUSPENSE_EXT:RECYCLE_KEY	String	event	(Optional) The field is provided by the calling cartridge.
DETAIL:ASS_SUSPENSE_EXT:RECYCLE_MODE	Integer	event	System generated based on the recycling mode selected by the user.
DETAIL:ASS_SUSPENSE_EXT:SERVICE_CODE	String	event	(Optional) The field is provided by the calling cartridge.
DETAIL:ASS_SUSPENSE_EXT:SOURCE_FILENAME	String	both	This is a required field for batch CDRs. This field is part of the batch key, which queries the SUSPENDED_BATCH_T table for information on the batch CDR file. This is an optional field for event CDRs.
DETAIL:ASS_SUSPENSE_EXT:SUSPENDED_FROM_BATCH_ID	String	event	(Optional) The field is provided by the calling cartridge
DETAIL:ASS_SUSPENSE_EXT:SUSPENSE_ID	Long	both	It is used by Suspended Event (SE) Loader for updating suspended usage records. This is a required field.
DETAIL:ASS_SUSPENSE_EXT:SUSPENSE_REASON	Long	both	Mapped from the error code.
DETAIL:ASS_SUSPENSE_EXT:SUSPENSE_SUBREASON	Long	event	Mapped from the error code.
DETAIL:ASS_SUSPENSE_EXT.:SUSPENSE_STATUS	Integer	both	Indicates whether the CDR is suspended, recycled or successful (**Succeeded**). This is a required field and is system generated.
DETAIL:ORIGINAL_BATCH_ID	String	event	(Optional) The field is provided by the calling cartridge.
HEADER:BATCH_ID	String	batch	The field is provided by the calling cartridge.
HEADER:TAP_PROCESSING_INFO	String	batch	The field is provided by the calling cartridge.
HEADER:SENDER	String	batch	Name of the service provider. The field is provided by the calling cartridge.
HEADER:SEQUENCE_NUMBER	String	batch	The field is provided by the calling cartridge.

Suspense DC Event Output

For suspended event CDRs, the Suspense Distribution Cartridge (DC) node generates **Create** or **Update** files in a format that is understood by Suspended Event (SE) Loader, which is a component of BRM.

Table 6–2 lists the **Create** event output fields.

Table 6–2 Create Event Output Fields

Field	Value
RECORD_TYPE	**"010"**
RECORD_NUMBER	"1"
SENDER	""
RECIPIENT	""
SEQUENCE_NUMBER	"1"
ORIGIN_SEQUENCE_NUMBER	"1"
CREATION_TIMESTAMP	UNIX EPOC time in seconds.
TRANSMISSION_DATE	UNIX EPOC time in seconds.
TRANSFER_CUTOFF_TIMESTAMP	UNIX EPOC time in seconds.
UTC_TIME_OFFSET	""
SPECIFICATION_VERSION_ NUMBER	"0"
RELEASE_VERSION	"0"
ORIGIN_COUNTRY_CODE	""
SENDER_COUNTRY_CODE	""
DATA_TYPE_INDICATOR	""
IAC_LIST	""
CC_LIST	""
CREATION_PROCESS	"SUSPENSE_CREATE"
SCHEMA_VERSION	"10000"
EVENT_TYPE	"/suspended_usage" The value entered in the **Create Event Type** field in the **Mode** configuration tab of the Suspense DC node. For more information, see "Configuring the Create File Output Settings for the Suspense DC Node".
QUERYABLE_FIELDS_MAPPING	List of editable table names and corresponding column names and types. For more information, see "Configuring the NPL Rule File for the Suspense DC Node".
RECORD_TYPE	**"020"**
SUSPENSE_REASON	Mapped from the error code.
SUSPENSE_SUBREASON	Mapped from the error code.
RECYCLE_KEY	DETAIL:ASS_SUSPENSE_EXT:RECYCLE_KEY
ERROR_CODE	DETAIL:ASS_SUSPENSE_EXT:ERROR_CODE If the value is not found, the value is set to "0".
PIPELINE_NAME	DETAIL:ASS_SUSPENSE_EXT:PIPELINE_NAME

Table 6–2 (Cont.) Create Event Output Fields

Field	Value
SOURCE_FILENAME	DETAIL:ASS_SUSPENSE_EXT:SOURCE_FILENAME
SERVICE_CODE	DETAIL:ASS_SUSPENSE_EXT:SERVICE_CODE
EDR_RECORD_TYPE	"20"
EDR_SIZE	Calculated. The size of the serialized CDR.
ACCOUNT_POID	DETAIL:ASS_SUSPENSE_EXT:ACCOUNT_POID If the value is not found, the value is set to "1 /account 1 0".
BATCH_ID	DETAIL:ASS_SUSPENSE_EXT:ORIGINAL_BATCH_ID
SUSPENDED_FROM_BATCH_ID	DETAIL:ASS_SUSPENSE_EXT:SUSPENDED_FROM_BATCH_ID
UTC_OFFSET_SECONDS	Calculated.
PIPELINE_CATEGORY	DETAIL:ASS_SUSPENSE_EXT:PIPELINE_CATEGORY
RECORD_TYPE	**"030"**
EDR_BUF	Contains the serialized CDR in XML format.
RECORD_TYPE	**"040"**
QUERYABLE_FIELDS	List of tab delimited CDR field values corresponding to each column listed in the QUERYABLE_FIELDS_MAPPING field. For more information, see "About Queryable Fields".

Table 6–3 lists the **Update** event output field values.

Table 6–3 Update Event Output Fields

Field	Value
RECORD_TYPE	**"010"**
RECORD_NUMBER	"1"
SENDER	""
RECIPIENT	""
SEQUENCE_NUMBER	"1"
ORIGIN_SEQUENCE_NUMBER	"1"
CREATION_TIMESTAMP	UNIX EPOC time in seconds.
TRANSMISSION_DATE	UNIX EPOC time in seconds.
TRANSFER_CUTOFF_TIMESTAMP	UNIX EPOC time in seconds.
UTC_TIME_OFFSET	""
SPECIFICATION_VERSION_ NUMBER	"0"
RELEASE_VERSION	"0"
ORIGIN_COUNTRY_CODE	""
SENDER_COUNTRY_CODE	""
DATA_TYPE_INDICATOR	""
IAC_LIST	""
CC_LIST	""

Table 6–3 (Cont.) Update Event Output Fields

Field	Value
CREATION_PROCESS	SUSPENSE_UPDATE
SCHEMA_VERSION	"10000"
EVENT_TYPE	"/tmp_suspended_usage" The value entered in the **Update Event Type** field in the **Mode** configuration tab of the Suspense DC node. For more information, see "Configuring the Update File Output Settings for the Suspense DC Node".
RECORD_TYPE	**"020"**
SUSPENSE_ID	DETAIL:ASS_SUSPENSE_EXT:SUSPENSE_ID
STATUS	If the error code is 0 or not found, the status is set to "2" (Succeeded). If the error code is not 0 or is found, the status is set to "0" (Suspended).
SUSPENSE_REASON	Mapped from the error code.
SUSPENSE_SUBREASON	Mapped from the error code.
ERROR_CODE	DETAIL:ASS_SUSPENSE_EXT:ERROR_CODE If the value is not found, the value is set to "0".
RECYCLE_MODE	DETAIL:ASS_SUSPENSE_EXT:RECYCLE_MODE
RECYCLE_KEY	DETAIL:ASS_SUSPENSE_EXT:RECYCLE_KEY

Suspense DC Batch Output

For suspended batch CDRs, the Suspense DC node generates **Create** or **Update** files in a format that is understood by Suspended Batch (SB) Loader, which is a component of BRM.

Table 6–4 lists the **Create** batch output fields.

Table 6–4 Create Batch Output Fields

Field	Value
RECORD_TYPE	**"010"**
STORABLE_CLASS	The value entered in the **Storable Class** field in the **Mode** configuration tab of the Suspense DC node.
RECORD_TYPE	**"020"**
SENDER	HEADER:SENDER
BATCH_ID	HEADER:BATCH_ID
SEQUENCE_NUMBER	HEADER:SEQUENCE_NUMBER
BATCH_NAME	DETAIL:ASS_SUSPENSE_EXT:SOURCE_FILENAME
ERROR_DIRECTORY	DETAIL:ASS_SUSPENSE_EXT:ERROR_PATH
ERROR_FILENAME	DETAIL:ASS_SUSPENSE_EXT:ERROR_FILENAME
PIPELINE_NAME	DETAIL:ASS_SUSPENSE_EXT:PIPELINE_NAME
PIPELINE_CATEGORY	DETAIL:ASS_SUSPENSE_EXT:PIPELINE_CATEGORY
ERROR_CODE	DETAIL:ASS_SUSPENSE_EXT:ERROR_CODE If the value is not found, the value is set to "0".

Table 6–4 (Cont.) Create Batch Output Fields

Field	Value
SUSPENSE_REASON	Mapped from the error code.
SUSPENSE_T	Calculated. UTC in seconds.
TAP_INFO	HEADER:TAP_PROCESSING_INFO

Table 6–5 lists the **Update** batch output fields.

Table 6–5 Update Batch Output Fields

Field	Value
RECORD_TYPE	**"010"**
STORABLE_CLASS	The value entered in the **Storable Class** field in the **Mode** configuration tab of the Suspense DC node.
RECORD_TYPE	**"030"**
BATCH_NAME	DETAIL:ASS_SUSPENSE_EXT:SOURCE_FILENAME
PIPELINE_NAME	DETAIL:ASS_SUSPENSE_EXT:PIPELINE_NAME
ERROR_CODE	DETAIL:ASS_SUSPENSE_EXT:ERROR_CODE If the value is not found, the value is set to "0".
SUSPENSE_STATUS	If the error code is 0 or not found, the status is set to "2" (**Succeeded**). If the error code is not 0 or is found, the status is set to "0" (**Suspended**).
SUSPENSE_REASON	Mapped from the error code.

7

Working with Suspense Java Hooks in NPL

This chapter lists and describes the Java hooks available for the Oracle Communications Offline Mediation Controller suspense handling.

About Suspense Java Hooks

Java hooks are an advanced feature of NPL (Node Programming Language) that make it possible 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 7–1 lists the suspense Java hooks methods.

Table 7–1 Suspense Java Hooks Method Summary

Modifier and Type	Method and Description
`void`	append(DCFieldContainer in, DCFieldContainer out) Copies to the output record all suspense fields that exist in the input record.
`void`	assemble(DCFieldContainer inContainer, DCFieldContainer outContainer, LongField errorCode, StringField pipelineName, StringField pipelineCategory) Clones the input record to the output record, adding the suspense fields.
`void`	assemble(DCFieldContainer inContainer, DCFieldContainer outContainer, LongField errorCode, StringField pipelineName, StringField pipelineCategory, StringField sourceFileName, StringField serviceCode) Clones the input record to the output record, adding the suspense fields.
`void`	assemble(DCFieldContainer out, StringField sourceFileName, StringField pipelineName, LongField errorCode) Adds to the output record the suspense fields and their values.
`void`	assemble(DCFieldContainer out, StringField sourceFileName, StringField pipelineName, LongField errorCode, StringField errorDir, StringField errorFileName) Adds to the output record the suspense fields and their values.
`void`	assemble(DCFieldContainer out, StringField sourceFileName, StringField pipelineName, LongField errorCode, StringField errorDir, StringField errorFileName, StringField sender, StringField batchId, StringField sequenceNumber, StringField tapInfo, StringField pipelineCategory) Adds to the output record the suspense fields and their values.
`StringField`	bytesAsString(BytesField inputBytes) Converts the given BytesField to StringField. The StringField return value contains a string of hex byte values.

Table 7–1 (Cont.) Suspense Java Hooks Method Summary

Modifier and Type	Method and Description
`StringField`	exists(DCFieldContainer container) If 1 or more key suspense fields exist in the given record, returns **true** (1).
`IntField`	float2int(FloatField fd) throws NDKException Converts the given FloatField to an IntField. Note that data may be truncated.
`LongField`	float2long(FloatField fd) throws NDKException Converts the given FloatField to a LongtField. Note that data may be truncated.
`StringField`	get3GPPTimeStamp(BytesField threeGPPDateField) Converts the BytesField of a byte array containing date in yyMMddhhmmss format to StringField of timestamp in seconds.
`StringField`	get3GPPTimeStamp(BytesField threeGPPDateField, StringField dateFormat) Converts the BytesField of a byte array containing date in yyMMddhhmmss format to StringField of timestamp in the given date format.
`BytesField`	get3GPPTimeStamp(LongField timestampInSecs) Converts the given LongField containing the timestamp in seconds into BytesField of a byte array containing date in yyMMddhhmmss format.
`StringField`	getDurationInSeconds(StringField startDate, StringField endDate) Returns the StringField containing the absolute long value of duration in seconds between the two given dates both in "yyyy-MM-dd HH:mm:ss" format.
`StringField`	getDurationInSeconds(StringField startDate, StringField endDate, StringField dateFormat) Returns the StringField containing the absolute long value of duration in seconds between the two given the date format.
`StringField`	getValue(ListField list, StringField mapName, StringField fieldName) Returns the value for the first fieldName found directly tied to the given ListField or in the first map found in the ListField.
`StringField`	getValue(ListField list, StringField mapName, StringField fieldName, StringField defaultValue) Returns the value for the first fieldName found directly tied to the given ListField, in the first map found in the ListField, or the given default value.
`StringField`	getValue(MapField mapField, StringField fieldName) Returns the value for the given field tied to the MapField.
`StringField`	getValue(MapField mapField, StringField fieldName, StringField defaultValue) Returns the value for the given field tied to the MapField or the given default value.
LongField	getValueSum(ListField listField, StringField mapName, StringField fieldname) Returns the sum in long of the values found for all the named field under the named map in the ListField. The sum is returned as a long, **0** is the default.
`FloatField`	int2float(IntField intd) throws NDKException Converts the given IntField to a FloatField.
`IntField`	isRecycled(DCFieldContainer container) If the DETAIL:ASS_SUSPENSE_EXT:SUSPENSE_ID field is found in the given record and its value is greater than **0**, returns **true** (1).
FloatField	long2float(DCFieldContainer container) Converts the given LongField to a FloatField.

Table 7–1 (Cont.) Suspense Java Hooks Method Summary

Modifier and Type	Method and Description
`ListField`	makeField(StringField listName, StringField mapName, StringField fieldName, DCField field) Creates the named ListField and the new field to a new MapField created for it.
`MapField`	makeField(StringField mapName, StringField fieldName, DCField field) Creates the named MapField and adds the new field to it.
`intField`	move(StringField sourceDir, StringField sourcePrefix, StringField sourceSuffix, StringField targetDir, StringField targetPrefix, StringField targetSuffix) throws Exception. Moves file(s) matching specified patterns from source directory to the target directory.
`intField`	move(StringField sourceFileWithFullPath, StringField targetFileWithFullPath) throws Exception. Moves source file with full path to the target file with full path.
`ListField`	setValue(ListField list, StringField mapName, StringField fieldName, DCField field) throws NDKException Adds or updates a field directly tied to the given ListField or to a map of the given ListField. The ListField must exist, use **Boolean fieldExists(OutputRec rec, String fieldID)** to check for existence.
`MapField`	setValue(MapField mapField, StringField fieldName, DCField field) throws NDKException Adds or updates the field in the MapField. The MapField must exist, use **Boolean fieldExists(OutputRec rec, String fieldID)** to check for existence.
`BytesField`	stringAsBytes(StringField inputString) Converts the given StringField containing a string of hex byte values to a BytesField, which is returned.

Suspense Java Hook Method Details

The section describes the suspense Java hook methods.

append

```
public void append(DCFieldContainer in, DCFieldContainer out)
```

Usage

Copies all existing suspense fields from the input field to the output record. Fields that already exist in the output record are not overwritten.

Copying all the existing suspense fields, proceeds only if the **exists(in)** method returns a true value, which indicates that one or more key suspense fields exist.

Parameters

in - the input record.

out - the output record.

assemble

```
public void assemble(DCFieldContainer inContainer,
  DCFieldContainer outContainer,
  LongField errorCode,
  StringField pipelineName,
```

```
  StringField pipelineCategory)
throws Exception
```

Usage

Clones input record to output record and adds to the output record the following fields:

* DETAIL:ASS_SUSPENSE_EXT:ERROR_CODE

* DETAIL:ASS_SUSPENSE_EXT:PIPELINE_NAME

* DETAIL:ASS_SUSPENSE_EXT:PIPELINE_CATEGORY

This method is equivalent to writing the following statements in NPL:

out=clone(in);

out.DETAIL:ASS_SUSPENSE_EXT:ERROR_CODE=<error code>;

out.DETAIL:ASS_SUSPENSE_EXT:PIPELINE_NAME=<pipeline name>;

out.DETAIL:ASS_SUSPENSE_EXT:PIPELINE_CATEGORY=<pipeline category>;

Note: Fields existing in the outContainer are replaced or removed as a result of the clone operation.

Both input record and output record must be in NAR format, otherwise an exception is thrown, the statement that contains the function call is skipped, and execution will continue with the next statement in the NPL program.

A warning is written to the node's log file if the node's configured debug level is set at the logging warnings level.

Parameters

inContainer - the input record, which must be in NAR format.

outContainer - the output record, which must be in the NAR format.

errorCode - the error code to be added to the output record.

pipelineName - the cartridge name/ID that detects the error and suspends the input record.

pipelineCategory - the group name in which the cartridge that initiates the suspense belongs.

Throws

Exception - if the input record or the output record is not in NAR format.

assemble

```
public void assemble(DCFieldContainer inContainer,
  DCFieldContainer outContainer,
  LongField errorCode,
  StringField pipelineName,
  StringField pipelineCategory,
  StringField sourceFileName,
  StringField serviceCode)
throws Exception
```

Usage

Clones input a record to an output record, and adds to the output record the following fields:

* DETAIL:ASS_SUSPENSE_EXT:ERROR_CODE

* DETAIL:ASS_SUSPENSE_EXT:PIPELINE_NAME

* DETAIL:ASS_SUSPENSE_EXT:PIPELINE_CATEGORY

* DETAIL:ASS_SUSPENSE_EXT:SOURCE_FILENAME

* DETAIL:ASS_SUSPENSE_EXT:SERVICE_CODE

This method is equivalent to writing the following statements in NPL:

out=clone(in);

out.DETAIL:ASS_SUSPENSE_EXT:ERROR_CODE=<error code>;

out.DETAIL:ASS_SUSPENSE_EXT:PIPELINE_NAME=<pipeline name>;

out.DETAIL:ASS_SUSPENSE_EXT:PIPELINE_CATEGORY=<pipeline category>;

out.DETAIL:ASS_SUSPENSE_EXT:SOURCE_FILENAME=<source_filename>;

out.DETAIL:ASS_SUSPENSE_EXT:SERVICE_CODE=<service_code>;

> **Note:** Fields existing in the outContainer are replaced or removed as a result of the clone operation.

Both the input record and output record must be in NAR format, or an exception is thrown, the statement that contains the function call is skipped, and processing will continue with the next statement in the NPL program.

A warning is written to the node's log file if the node's configured debug level is set at the logging warnings level.

Parameters

inContainer - the input record, which must be in NAR format.

outContainer - the output record, which must be in the NAR format.

errorCode - the error code, which is added to the output record.

pipelineName - the cartridge names ID that detects the error and suspends the input record.

pipelineCategory - the group name in which the cartridge that initiates the suspense belongs.

sourceFileName - the input source file name for the suspended CDR.

serviceCode - the service code for the suspended record CDR.

Throws

Exception - if the input record or the output record is not in NAR format.

assemble

```
public void assemble(DCFieldContainer out,
  StringField sourceFileName,
  StringField pipelineName,
```

```
  LongField errorCode)
```

Usage

Adds to output record the following fields with the given values:

* DETAIL:ASS_SUSPENSE_EXT:SOURCE_FILENAME

* DETAIL:ASS_SUSPENSE_EXT:PIPELINE_NAME

* DETAIL:ASS_SUSPENSE_EXT:ERROR_CODE

Existing values are overwritten if the field already exists in the output record.

Parameters

out - the output record.

sourceFileName - the input source file that is suspended.

pipelineName - the cartridge name/ID that detects the error and suspends the input record.

errorCode - the error code to be added to the output record.

assemble

```
public void assemble(DCFieldContainer out,
  String sourceFileName,
  String pipelineName,
  long errorCode,
  String errorDir,
  String errorFileName)
```

Usage

Adds to output record the following fields with the given values:

* DETAIL:ASS_SUSPENSE_EXT:SOURCE_FILENAME

* DETAIL:ASS_SUSPENSE_EXT:PIPELINE_NAME

* DETAIL:ASS_SUSPENSE_EXT:ERROR_CODE

* DETAIL:ASS_SUSPENSE_EXT:ERROR_PATH

* DETAIL:ASS_SUSPENSE_EXT:ERROR_FILENAME

Existing values are overwritten if the field already exists in the output record.

Parameters

out - the output record.

sourceFileName - the input source file that is suspended.

pipelineName - the cartridge name/ID that detects the error and suspends the input record.

errorCode - the error code to be added to the output record.

errorDir - the directory path where the suspended files are located.

errorFileName - the error filename.

assemble

```
public void assemble(DCFieldContainer out,
```

```
  StringField sourceFileName,
  StringField pipelineName,
  LongField errorCode,
  StringField errorDir,
  StringField errorFileName,
  StringField sender,
  StringField batchId,
  StringField sequenceNumber,
  StringField tapInfo,
  StringField pipelineCategory)
```

Usage

Adds to output record the following fields with the given values:

* DETAIL:ASS_SUSPENSE_EXT:SOURCE_FILENAME

* DETAIL:ASS_SUSPENSE_EXT:PIPELINE_NAME

* DETAIL:ASS_SUSPENSE_EXT:ERROR_CODE

* DETAIL:ASS_SUSPENSE_EXT:ERROR_PATH

* DETAIL:ASS_SUSPENSE_EXT:ERROR_FILENAME

*HEADER:SENDER

*HEADER:BATCH_ID

*HEADER:SEQUENCE_NUMBER

*HEADER:TAP_PROCESSING_INFO

*DETAIL:ASS_SUSPENSE_EXT:PIPELINE_CATEGORY

Existing values are overwritten if the field already exists in the output record.

Parameters

out - the output record.

sourceFileName - the suspended input source file.

pipelineName - the cartridge name/ID that detects the error and suspends the input record.

errorCode - the error code, which is added to the output record.

errorDir - the directory path where the suspended files are located.

errorFileName - the error filename.

sender - the sender of the input source file.

batchId - the batch ID assigned with this suspension.

sequenceNumber - the sequence number associated with the suspended file.

tapInfo - the TAP processing information.

pipelineCategory - the group name in which the cartridge that initiates the suspense belongs.

bytesAsString

```
public StringField bytesAsString(BytesField inputBytes)
```

Usage

Converts the given BytesField to StringField, containing a string of hex byte values, which is returned.

Parameters

inputBytes - the BytesField containing the byte array, which is converted to a string.

Returns

The conversion resulted string in StringField.

exists

```
public IntField exists(DCFieldContainer container)
```

Usage

Returns true (1) if 1 or more key suspense fields exist in the given record.

* DETAIL:ASS_SUSPENSE_EXT:SUSPENSE_ID

* DETAIL:ASS_SUSPENSE_EXT:ERROR_CODE

* DETAIL:ASS_SUSPENSE_EXT:SUSPENSE_STATUS

Parameters

container - the record to check.

Returns

true if one of the above suspense fields is found in the given record.

float2int

```
public IntField float2int(FloatField fd) throws NDKException
```

Usage

Converts the given FloatField to an IntField. Note that data may be truncated.

Parameters

fd - The FloatField to be converted from a float to an integer.

Returns

The conversion resulted integer in IntField.

Throws

NDKException.

float2long

```
public LongField float2long(FloatField fd) throws NDKException
```

Usage

Converts the given FloatField to a LongField. Note that data may be truncated.

Parameters

fd - the FloatField to be converted from a float to a long.

Returns

The conversion resulted long in LongField.

Throws

NDKException.

get3GPPTimeStamp

```
public StringField get3GPPTimeStamp(BytesField threeGPPDateField)
```

Usage

Converts the BytesField of a byte array containing the date in yyMMddhhmmss format to StringField of timestamp in seconds.

The following example shows the DETAIL:CHARGING_START_TIMESTAMP is set to the timestamp in seconds converted from ServiceDeliveryStartTimeStamp:

```
out.DETAIL:CHARGING_START_
TIMESTAMP=str2long(suspHandler.get3GPPTimeStamp(in.ServiceDeliveryStartTimeStamp))
;
```

Parameters

threeGPPDateField - the BytesField containing date in yyMMddhhmmss format.

Returns

StringField containing time in seconds, which can be converted into a long.

get3GPPTimeStamp

```
public StringField get3GPPTimeStamp(BytesField threeGPPDateField, StringField
dateFormat)
```

Usage

Converts the BytesField of a byte array containing the date in yyMMddhhmmss format to StringField of timestamp in the given date format.

The following example shows how to log an entry by printing ServiceDeliveryStartTimeStamp as a string in yyyy-MM-dd HH:mm:ss date format:

```
logInfo("StartDate="+suspHandler.get3GPPTimeStamp(in.ServiceDeliveryStartTimeStamp
, "yyyy-MM-dd HH:mm:ss"));
```

Parameters

threeGPPDateField - the BytesField containing the date in yyMMddhhmmss format.

dateFormat - a date format according to the Java documentation of the time pattern format syntax for the **SimpleDateFormat** class.

For example, yyyy-MM-dd HH:mm:ss.

Use **""** (blank) to get the timestamp in seconds without the date format, which is equivalent to **get3GPPTimeStamp(BytesFieldthreeGPPDateField)**

Returns

StringField containing date in the given format.

get3GPPTimeStamp

```
public BytesField get3GPPTimeStamp(LongField timestampInSecs)
```

Usage

Converts the given LongField containing the timestamp in seconds into BytesField of a byte array containing the date in yyMMddhhmmss format.

The following example shows the BytesField named ServiceDeliveryStartTimeStamp is set to the LongField DETAIL:CHARGING_START_TIMESTAMP, converted to a byte array containing the date in yyMMddhhmmss format:

```
out.ServiceDeliveryStartTimeStamp=suspHandler.get3GPPTimeStamp(in.DETAIL:CHARGING_
START_TIMESTAMP);
```

Parameters

timestampInSecs - the given LongField, which is the timestamp in seconds.

Returns

BytesField of a byte array containing date in yyMMddhhmmss format.

getDurationInSeconds

```
public StringField getDurationInSeconds(StringField startDate, StringField
endDate)
```

Usage

Returns the StringField containing the absolute long value of duration in seconds between the two given dates, both in "yyyy-MM-dd HH:mm:ss" format.

The following example shows how to log an entry for the duration between ServiceDeliveryStartTimeStamp and ServiceDeliveryEndTimeStamp:

```
logInfo("Duration="+suspHandler.getDurationInSeconds(
suspHandler.get3GPPTimeStamp(in.ServiceDeliveryStartTimeStamp),
suspHandler.get3GPPTimeStamp(in.ServiceDeliveryEndTimeStamp)));
```

Parameters

startDate - the given start date.

endDate - the given end date.

StringField containing the long value of duration between startDate and endDate, both in "yyyy-MM-dd HH:mm:ss" format.

Returns

StringField containing the long value of duration between startDate and endDate in the given format.

getDurationInSeconds

```
public StringField getDurationInSeconds(StringField startDate, StringField
endDate, StringField dateFormat)
```

Usage

Returns the StringField containing the absolute long value of duration in seconds between the two given the date format.

The following example shows how to log an entry for the duration between ServiceDeliveryStartTimeStamp and ServiceDeliveryEndTimeStamp:

```
logInfo("Duration="+suspHandler.getDurationInSeconds(
suspHandler.get3GPPTimeStamp(in.ServiceDeliveryEndTimeStamp, "yyyy-MM-dd
HH:mm:ss"), suspHandler.get3GPPTimeStamp(in.ServiceDeliveryStartTimeStamp,
"yyyy-MM-dd HH:mm:ss"), "yyyy-MM-dd HH:mm:ss"));
```

Parameters

startDate - the given start date.

endDate - the given end date.

dateFormat - the date format for the two dates.

Returns

StringField containing the long value of duration between startDate and endDate in the given format.

getValue

```
public StringField getValue(ListField list, StringField mapName, StringField
fieldName)
```

Usage

Returns the value for the first fieldName found directly tied to the given ListField or in the first map found in the ListField.

The following example shows the DETAIL:A_NUMBER is set to the value of SIP_URI field or TEL_URI fields in any MapField under ListField named List_Of_Calling_ Party_Address:

```
out.DETAIL:A_NUMBER=suspHandler.getValue(in.List_Of_Calling_Party_
Address,"*","SIP_URI,TEL_URI");
```

Parameters

list - the ListField in which fieldName is searched and a value returned.

mapName - the map name directly under the list.

Multiple map names may be entered, delimited by a comma. The first matched map is used to find the matching fieldName.

- Use **"*"** for the first map found.
- Use **""** for the field directly tied to the list without a map.

fieldName - the field name where the value is returned.

Multiple field names may be entered, delimited by comma. The first value for the fieldName found is returned.

Returns

The String value associated with the field name.

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

getValue

```
public StringField getValue(ListField list, StringField mapName, StringField
fieldName, StringField defaultValue)
```

Usage

Returns the value for the first fieldName found directly tied to the given ListField, in the first map found in the ListField, or the given default value

The following example shows the DETAIL:VOLUME_SENT is set to the value of Content_Length field in any MapField under ListField named List_Of_Message_ Bodies, given the default value of **0**:

```
out.DETAIL:VOLUME_SENT=str2float(suspHandler.getValue(in.List_Of_Message_Bodies,
"*", "Content_Length","0"));
```

Parameters

list - the ListField in which fieldName is searched and a value returned.

mapName - the map name directly under the list.

- Multiple map names may be entered, delimited by a comma. The first matched map is used to find the matching fieldName.
- Use **"*"** for the first map found.
- Use **""** for the field directly tied to the list without a map.

fieldName - the field name in which the value is returned.

Multiple field names may be entered, delimited by comma. The first value for the **fieldName** found is returned.

defaultValue - the value to return if **fieldName** is not found.

Returns

The String value associated with the field name or the given default value if **fieldName** is not found.

getValue

```
public StringField getValue(MapField mapField, StringField fieldName)
```

Usage

Returns the value for the given field tied to the MapField.

The following example shows DETAIL:B_NUMBER is set to the value of SIP_URI or TEL_URL fields in the MapField named Called_Party_Address:

```
out.DETAIL:B_NUMBER=suspHandler.getValue(in.Called_Party_Address,"SIP_URI,TEL_
URI");
```

Parameters

mapField - the MapField in which the value for the given fieldName is to be found.

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

Multiple field names may be entered, delimited by comma. The first value for the **fieldName** found is returned.

Returns

The String value associated with the field name.

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

getValue

```
public StringField getValue(MapField mapField, StringField fieldName, StringField
defaultValue)
```

Usage

Returns the value for the given field tied to the MapField or the given value.

The following example shows DETAIL:B_NUMBER is set to the value of TEL_URL fields in the MapField named Called_Party_Address, given the default value of 9990000:

```
out.DETAIL:B_NUMBER=suspHandler.getValue(in.Called_Party_Address,"TEL_URI",
"9990000");
```

Parameters

mapField - the MapField in which the value for the given fieldName is to be found.

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

Multiple field names may be entered, delimited by a comma. The first value for the **fieldName** found is returned.

defaultValue - the value to return if fieldName is not found.

Returns

The String value associated with the field name or the default value if **fieldName** is not found.

getValueSum

```
public LongField getValueSum(ListField list, StringField mapName, StringField
fieldName)
```

Usage

Returns the sum in long of the values found for all the named field under the named map in the ListField. The sum is returned as a long, 0 is the default.

The following example shows DETAIL:VOLUME_SENT is set to the sum of the value of all DataVolumeGPRSUplink fields of any MapField in ListField 20209:

```
out.DETAIL:VOLUME_SENT = suspHandler.long2float(suspHandler.getValueSum(in.20209,
"*", "DataVolumeGPRSUplink"));
```

Parameters

list - the ListField in which **fieldName** is searched and value returned.

mapName - The name of the map to which the field is tied.

Use **"*"** for the first map found in the ListField.

Use **""** for the field directly tied to the list without a map.

fieldName - the named field. The type of the field for the given fieldName must be of long, int, or short to be summed.

Returns
The sum in long of the values found for all the named field.

int2float

```
public FloatField int2float(IntField intd) throws NDKException
```

Usage
Converts the given IntField to a FloatField.

Parameters
intd - The IntField to be converted from an integer to a float.

Returns
The conversion resulted float in FloatField.

Throws
NDKException.

isRecycled

```
public IntField isRecycled(DCFieldContainer container)
```

Usage
Returns true (1) if DETAIL:ASS_SUSPENSE_EXT:SUSPENSE_ID field is found in the given record and its value is greater than 0.

Parameters
container - the record to check.

Returns
True (1) if DETAIL:ASS_SUSPENSE_EXT:SUSPENSE_ID field exists and its value is greater than 0. Else, **false**.

long2float

```
public FloatField long2float(LongField longfd) throws NDKException
```

Usage
Converts the given LongField to a FloatField.

Parameters
longfd - The LongField to be converted from an long to a float.

Returns
The conversion resulted float in FloatField.

Throws
NDKException.

makeField

```
public ListField makeField(StringField listName, StringField mapName, StringField
fieldName, DCField field)
```

Usage

Creates the named ListField and the new field to a new MapField created for it.

The following example shows that a ListField named "List_Of_Calling_Party_ Address" is to be created with a MapField with no name to which a new field TEL_ URI is created with data type and value from TEL_URI:

```
out.List_Of_Calling_Party_Address=suspHandler.makeField("List_Of_Calling_Party_
Address", "*", "TEL_URI", TEL_URI);
```

Parameters

listName - the name of the ListField to create.

mapName - the map name for the new map. Use "*" or "" for map of no name.

fieldName - the field name of the new field to create and add to the map.

field - the DCField that contains the field type and field value for the named field.

Returns

The ListField created.

makeField

```
public MapField makeField(StringField mapName, StringField fieldName, DCField
field)
```

Usage

Creates the named MapField and adds the new field to it.

The following example shows that the MapField Called_Party_Address is created and the field TEL_URI is added to it. The data type and value is given in TEL_URL:

```
out.Called_Party_Address=suspHandler.makeField("Called_Party_Address", "TEL_URI",
TEL_URI);
```

Parameters

mapName - the map name for the new map. Use **"*"** or **""** for map of no name.

fieldName - the field name of the new field to create and add to the map.

field - the DCField that contains the field type and field value for the named field.

Returns

The MapField created.

move

```
public IntField move(String sourceDir,
  String sourcePrefix,
  String sourceSuffix,
  String targetDir,
  String targetPrefix,
  String targetSuffix)
```

```
throws Exception
```

Usage

Moves files in sourceDir having sourcePrefix and sourceSuffix to targetDir replacing file names with targetPrefix and targetSuffix.

Ordinary files are moved, but not directories. The number of files moved (and renamed) is returned.

If the target file exists, it is replaced with a delete before the file move.

> **Note:** This method uses Java File.renameTo(File) so its limitation is inherent.

In the example below, n=move("/home/pin", "my_", "txt", "/tmp", "err_", "bad"), where n=2

```
ls /home/pin
my_file101.txt my_file102.txt web/

(after the move)

ls /home/pin
web/

ls /tmp
err_file101.bad err_file102.bad
```

Parameters

sourceDir - the source directory from where files are to be moved.

sourcePrefix - the beginning string to match the source file name for move.

sourceSuffix - the ending string to match the source file name for move.

targetDir - the destination directory to where files are moved.

targetPrefix - the string to add at the beginning of the target file name.

targetSuffix - the string to append at the end of the target file name.

Returns

The number of files moved.

Throws

Exception - if the specified directories are not valid, the source directory is not readable, target directory is not writable, or when the existing target file cannot be replaced.

move

```
public IntField move(String sourceFileWithFullPath,
   String targetFileWithFullPath)
throws Exception
```

Usage

Moves the source file with full path to the target file with full path.

If the target file exists, it is deleted before the file is moved. The number of files moved is returned, that is, **1** if successful, else **0**.

> **Note:** This method uses Java File.renameTo(File) so its limitation is inherent.

Parameters

sourceFileWithFullPath - the full path of the source file to be moved (renamed).

targetFileWithFullPath - the full path of the target file.

Returns

1 if move was successful, else returns **0**.

Throws

Exception - if either file is invalid, the source file is not found, or when the existing target file cannot be replaced.

setValue

```
public ListField setValue(ListField list, StringField mapName, StringField
fieldName, DCField field) throws NDKException
```

Usage

Adds or updates a field directly tied to the given ListField or to a map of the given ListField.

The ListField must exist. Use **Boolean fieldExists(OutputRec rec, String fieldID)** to check for existence.

The following example shows that ListField InterOperatorIdentifiers is updated where OriginatingIOI field in MapField InterOperatorIdentifiers is set to the value of OperatorID, if OriginatingIOI field does not exist, it is added:

```
out.InterOperatorIdentifiers=suspHandler.setValue(out.InterOperatorIdentifiers,
"InterOperatorIdentifiers", "OriginatingIOI", OperatorID);
```

Example below shows how to update or create ListField:

```
if (fieldExists(out, "InterOperatorIdentifiers")) {
  out.InterOperatorIdentifiers=suspHandler.setValue(out.InterOperatorIdentifiers,
"InterOperatorIdentifiers", "OriginatingIOI", OperatorID);
  } else {
    out.InterOperatorIdentifiers=suspHandler.makeField("InterOperatorIdentifiers",
"InterOperatorIdentifiers", "OriginatingIOI", OperatorID);
}
```

Parameters

listField - the listField to which the field is directly tied or the map of which the field is a member.

mapName - the name of the map to which the field is tied.

Use **"*"** for the first map found.

Use **""** for the field directly tied to the list without a map.

fieldName - the name of the field to add or update.

field - the DCField that contains the field type and field value for the named field.

Returns

The updated ListField.

Throws

NDKException

setValue

```
public MapField setValue(MapField mapField, StringField fieldName, DCField field)
throws NDKException
```

Usage

Adds or updates the field in the MapField.

The MapField must exist, use **Boolean fieldExists(OutputRec rec, String fieldID)** to check for existence.

In the following example, MapField Called_Party_Address is updated, and the TEL_URI field is set to the value of DETAIL:A_NUMBER. If the TEL_URI field does not exist, it is added:

```
out.Called_Party_Address=suspHandler.setValue(out.Called_Party_Address, "TEL_URI",
in.DETAIL:A_NUMBER);
```

Parameters

mapField - the MapField to which field will be added or updated with the field and value given.

fieldName - the name of the field to add or update.

field - the DCField that contains the field type and field value for the named field.

Returns

The updated MapField.

Throws

NDKException

stringAsBytes

```
public BytesField stringAsBytes(StringField inputString)
```

Usage

Converts the given StringField containing a string of hex byte values to a BytesField, which is returned.

Parameters

inputString - the StringField containing a string of hex byte values, which are converted to a byte array.

Returns

The conversion resulted byte array in the BytesField.