Oracle® Communications Billing and Revenue Management

Setting Up Pipeline Rating and Discounting

Release 15.2
G35844-01
January 2026

Oracle Communications Billing and Revenue Management Setting Up Pipeline Rating and Discounting, Release 15.2

G35844-01

Contents

3 Configuring EDR Input Processing

4 Configuring EDR Output Processing

5 Configuring EDR Preprocessing

6 Setting Up EDR Enrichment

7 Setting Up Pipeline Aggregation

8 Migrating Pipeline Manager Data Between Test and Production Systems

9 Transferring Data Between Pipeline Manager Databases

10 Creating iScripts and iRules

15 Setting Up Revenue Assurance Manager for Pipeline Batch Rating

16 Setting Up Pipeline Manager Taxation

17 Credit Limit and Threshold Checking During Batch Rating

About This Content

This book describes how to set up Oracle Communications Billing and Revenue Management Pipeline Manager to rate and discount usage events.

Audience

This book is intended for pricing administrators and charging experts.

Part I

Configuring Pipeline Rating

This part describes how to configure pipeline rating in an Oracle Communications Billing and Revenue Management (BRM) system. It contains the following chapters:

1

About Pipeline Rating

Learn how to configure pricing components for rating with Oracle Communications Billing and Revenue Management (BRM) Pipeline Manager.

Topics in this document:

About Configuring Pipeline Rating

Configuring pipeline rating involves two sets of tasks:

- Creating pricing components using Pricing Center or Pricing Design Center (PDC)
- Configuring rating function modules

Rating EDRs Split Across Time Periods

Event data records (EDRs) can span multiple time periods, necessitating careful consideration. For instance, when off-peak rates commence at 19:10, and a call starts at 19:10 and concludes at 19:35, the call straddles the boundary between peak and off-peak rates. When you define a charge, you can specify a splitting option dictating how the EDR is rated across the time periods.

You can choose from four splitting options:

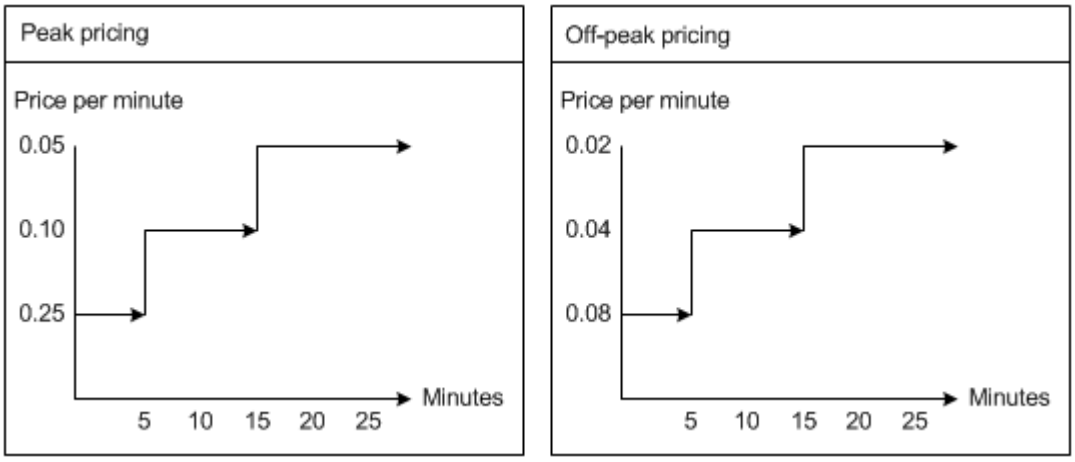
- **No splitting, based on start time**: Use the start time to rate the entire call.
- **No splitting, based on end time**: Use the end time to rate the entire call.
- **Consecutive splitting**: Divide the call into separately rated segments. When using multiple price steps for different usage levels, continue counting the call duration from the time of the split.
- **Isolated splitting**: Divide the call into separately rated parts. When using multiple price steps for different usage levels, start over at zero from the time of the split.

Pipeline Manager uses the consecutive and isolated splitting options when employing a combination of price steps and time-based rating. In such cases, an event may be divided by time zones, leading to the application of distinct pricing models, each with its own pricing steps. Consecutive and isolated splitting allows users to manage the selection of the most suitable pricing step.

Figure 1-1 shows an example of peak pricing and off-peak pricing for the following time periods:

- 06:00 - 07:30 is peak
- 07:30 - 10:00 is off-peak

Figure 1-1 Peak and Off-Peak Pricing



If Pipeline Manager rates an EDR that starts at 7:10 and finishes at 7:35, for a total of 25 minutes, it splits the event into two segments:

- 07:10 – 07:30: Peak (20 minutes)
- 07:30 – 07:35: Off-peak (5 minutes)

Figure 1-2 shows how consecutive splitting would be applied for this example. The gray areas show how the pricing steps are applied in both peak and off-peak pricing. The first 20 minutes are peak pricing, so the call is rated at 0.25 per minute, 0.10 per minute, and 0.05 per minute. The last 5 minutes are off-peak pricing, which, since it uses consecutive splitting, takes into account the previous 20 minutes, and rates the final 5 minutes at .02 per minute.

Figure 1-2 Consecutive Splitting

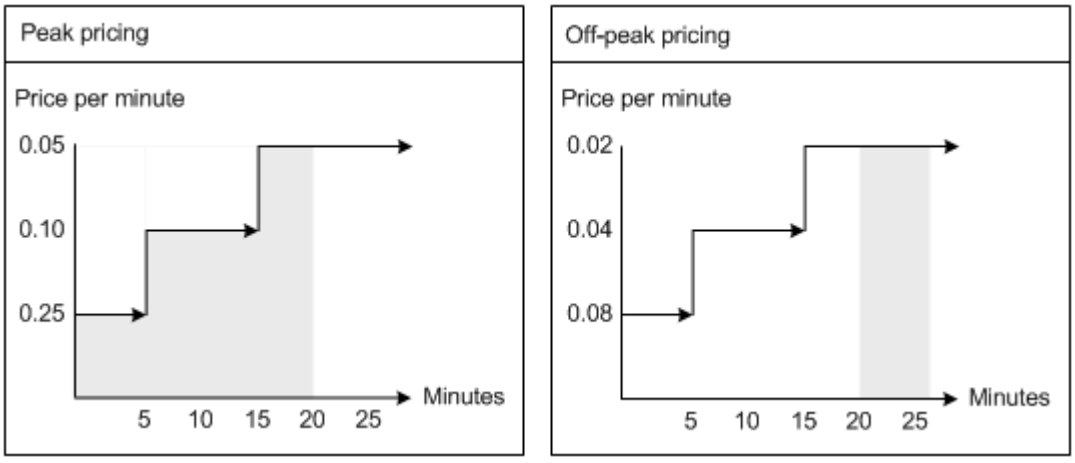
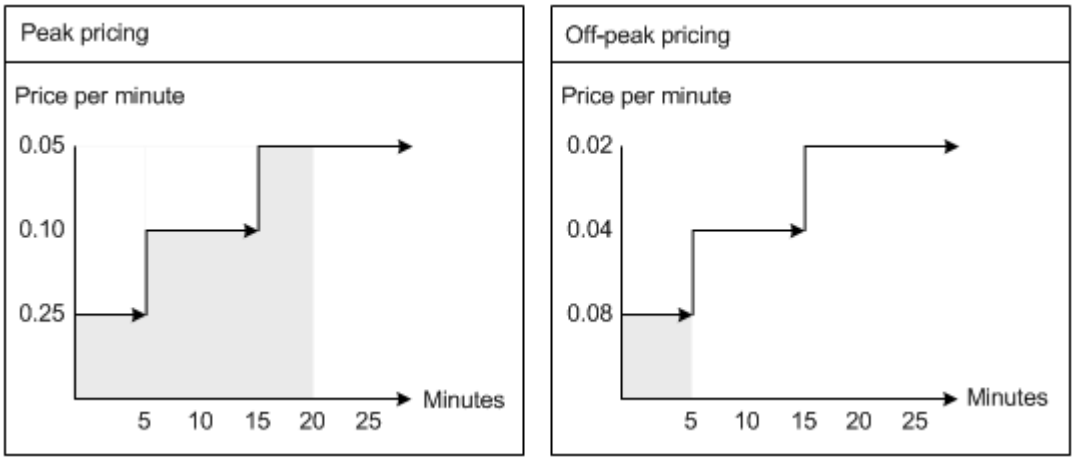


Figure 1-3 shows how isolated splitting would be applied for this example. The first 20 minutes matches the peak pricing for consecutive splitting. The last 5 minutes are off-peak pricing, which, because it uses isolated splitting, does not consider the previous 20 minutes. Instead, it begins counting at zero, so the remaining 5 minutes are rated by the first step, that is, .08 per minute.

Figure 1-3 Isolated Splitting



Using consecutive splitting and isolated splitting gives different results for the final charge for the EDR. In this example:

Table 1-1 shows the results for consecutive splitting.

Table 1-1 Results of Consecutive Splitting

Usage	Calculation	Result
Peak (20 minutes)	(5 * 0.25) + (15 * 0.10) + (5 * 0.05)	3.00
Off-peak (5 minutes)	5 * 0.02	0.10
NA	Total	3.10

Table 1-2 shows the results for isolated splitting.

Table 1-2 Results of Isolated Splitting

Usage	Calculation	Result
Peak (20 minutes)	(5 * 0.25) + (15 * 0.10) + (5 * 0.05)	3.00
Off-peak (5 minutes)	5 * 0.08	0.40
NA	Total	3.40

About Pipeline Charge Versions

Every charge needs at least one charge version, and you can create as many versions as you need. Each version is valid for a specific time period, and only one version is active during each period. The start time of an EDR determines which charge version applies.

When you create a charge version, you set a validity period and choose a zone model. The zone model categorizes calls based on their origin and destination regions for rating purposes. Each charge version can have different zone models.

Charge versions are stored in the IFW_RATEPLAN_VER table in the Pipeline Manager database. There are two types:

- **Basic Version**: This is the foundation for other versions. When creating a basic version, you must define all required attributes.
- **Delta Version**: This inherits attributes from a basic version, and you only need to specify the attributes you want to change, such as the version, validity period, and zone model.

About Pipeline Charge Configurations

The charge configuration determines which pricing or price selector is used to charge a given EDR. When you define a charge configuration, you specify a combination of criteria that an EDR must match to be rated by a particular pricing.

Each configuration maps a combination of service code, service class, and impact category to a combination of time model and time period. The configuration then maps the time model/time period combination to a pricing or price selector.

You can create any number of configurations for a charge version. The configurations in a given charge version must cumulatively cover all possible combinations of service code, service class, impact category, time model, and time period.

When you associate a pricing with a charge configuration, you can optionally specify an alternative pricing that is used in addition to the main pricing. You can compare the charges that result from the two models. For example, you may want to better understand the financial impact of a change to a different pricing before committing to the change.

Charge configurations are stored in the IFW_RATEPLAN_CNF table in the Pipeline Manager database.

Using Passthrough Prices

When you define a charge configuration, you can choose to ignore the calculated price and use a price that is passed in by the CDR instead. For example, you can ignore the calculated charge and use a passed-in charge if you receive external wholesale charges and want to use them for retail rating.

You can also modify or replace the passed-in price by specifying an add-on type and entering a charge. There are three add-on types:

- **Percentage** increases the passed-in price by a percentage that you enter.
- **Addon Value** increases the passed-in price by a fixed amount that you enter.
- **New Value** replaces the passed-in price with an amount you enter.

To use the passed-in price without modification, specify **0** as the charge.

About Pipeline Rate Adjustments

Rate adjustments are an optional way to customize a pipeline charge version. You can use rate adjustments to provide discounts based on date, time, service, and other event attributes. For example, you can provide a discount on all calls for a specific day.

An adjustment can be a percentage change to the original charge, a value to be added to the charge, or a completely new value to replace the charge.

When you define a rate adjustment, you specify dates and times during which the adjustment is valid and a maximum quantity above which the adjustment does not apply.

You also specify rules that determine whether an EDR qualifies for the adjustment. These rules filter EDRs based on values such as usage class, usage type, service code, service class, impact category, source network, and destination network. You can enter fixed values, expressions, or a wildcard (**.***) that matches all values.

When you use a rate adjustment, the original charge is overwritten by the adjusted charge. This is different from discounting, which leaves the original charge in place while calculating a discounted charge. Adjustments and discounts are also handled differently for accounting purposes. When calculating the general ledger (G/L) impact, the adjusted amount is not considered revenue. When you use discounting, however, the discounted amount is counted as revenue.

There are two ways to define rate adjustments:

- You can define rate adjustments in Pricing Center. In this case, the rate adjustment data is stored in the IFW_RATEADJUST table in the Pipeline Manager database.
- You can create a file that defines rate adjustment rules.

Rate adjustment is performed by the FCT_RateAdjust module. The module reads data from the EDR and evaluates it according to the rate adjustment rules stored in the database or in the rate adjustment file.

2

Configuring Pipeline Rating

Learn how to configure the Oracle Communications Billing and Revenue Management (BRM) pricing data and modules necessary for batch rating with Pipeline Manager.

Topics in this document:

About Configuring Function Modules for Pipeline Rating

Three groups of modules are involved in rating:

- The pre-rating modules (FCT_ GlobalRating, FCT_CustomerRating, and FCT_RSC_Map) gather information and prepare the event data record (EDR) for rating.
- The FCT_MainRating module applies charges and pricing to the EDR, creating charge breakdown data, including charge packets with charges.
- The post-rating modules (FCT_RateAdjust and FCT_BillingRecord) make changes to the EDR after the charge data is included.

Multiple Pipeline Manager modules add and delete charge packet blocks to EDRs. Therefore, the value in the NUMBER_OF_CHARGE_PACKETS field does not reflect the actual number of charge packets in the charge breakdown record. You can get the correct number of charge packets by using the edrNumDatablocks function in a custom iScript module.

Depending on how your pipelines are set up, you need to configure some or all of these modules for rating.

About the Rating Data Modules

Configure the following data modules for rating:

- DAT_AccountBatch
- DAT_BalanceBatch
- DAT_Calendar
- DAT_Currency
- DAT_Dayrate
- DAT_ExchangeRate
- DAT_ItemAssign
- DAT_ModelSelector
- DAT_NOSP
- DAT_PriceModel
- DAT_Rateplan
- DAT_TimeModel

About Using Filter Sets to Apply System Products and Discounts

Use filter sets to apply system products/charge offers and system discounts/discount offers to a select group of customers. For example, you can provide reduced international call rates for all customers with great credit.

When configured to use filter sets, the pipeline:

1. Analyzes each EDR to determine whether it meets the criteria for a filter set.
2. Adds any applicable system products/charge offers and system discounts/ discount offers, along with their priorities, to the EDR's list of purchased products/charge offers.
3. Uses the product/charge offer and/or discount/discount offer when rating the EDR.

> **Note**
>
> The actual product/charge offer or discount/discount offer the pipeline uses to rate the EDR depends on how your system is configured.

You use the following to configure your system for filter sets:

- **system_filterset_edr_field_values**. This file specifies which EDR fields and values can be used as filter criteria. You must load this file into the BRM database prior to creating your filter sets.
- **IRL_UsageType**. This iRule assigns usage types to an EDR. This signals the pipeline that the EDR qualifies for the special consideration that a filter set contains.
- **FCT_Filter_Set**. This module determines whether an EDR qualifies for any filter sets, and, if it does, adds any applicable system products/charge offers and system discounts/ discount offers to the EDR's list of purchased products/charge offers.

To configure BRM to use filter sets, perform the following tasks:

1. Specify which EDR fields and values can be used as filter criteria and then load the data into the BRM database. See "Loading Filter Set Data into BRM".
2. Define your filter sets by using a custom application. See "Defining Your Filter Sets".
3. Configure the IRL_UsageType iRule to assign usage types to EDRs.
4. Configure FCT_Filter_Set to apply system products/charge offers and discount/discount offers to specified market segments. See "FCT_Filter_Set" in *BRM Pipeline Manager Reference*.
5. Connect the FCT_DiscountAnalysis module to the FCT_Filter_Set module. Use the FCT_DiscountAnalysis module's **Filter_SetModule** registry entry. See "FCT_DiscountAnalysis" in *BRM Pipeline Manager Reference*.

Loading Filter Set Data into BRM

You can specify the EDR fields and values that you can use as filter criteria. To do so, you edit the **system_filterset_edr_field_values.en_US** sample file in the *BRM_home***/sys/msgs/system_filter_set** directory. *BRM_home* is the directory where you installed BRM components. For example, to use location as filter criteria, add these entries to the file:

```
DETAIL.ASS_CAMEL_EXT.MSC_ADDRESS="London"
DETAIL.ASS_CAMEL_EXT.MSC_ADDRESS="Paris"
```

After defining the field values, you use the load_localized_strings utility to load the contents of the **system_filterset_edr_field_values.***locale* file into the **/strings** object. To run the **load_localized_strings** utility, use this command:

```
load_localized_strings system_filterset_edr_field_values.locale
```

> **Note**
>
> If you are loading a localized version of this file, use the correct file extension for your locale.

Defining Your Filter Sets

You define your filter sets by creating a custom client application to call the filter set opcodes. BRM then stores data about each filter set in **/filter_set/product** objects in the BRM database.

> **Note**
>
> To create and manage filter sets by using a custom client application, configure your application to call the filter set opcodes.

To create a filter set, perform the following:

1. Create your system products/charge offers and system discounts/discount offers and associate them with specific service and event types. For example:
   - Create a system charge offer for **/service/telco/gsm/telephony** events that charges a flat rate of 5 cents per minute.

- Create a system discount offer for **/service/telco/gsm/sms** events that provides a 20% discount for the first 10 minutes of usage.

2. Create your filter sets by mapping filter criteria to system charge offers and system discount offers. When creating filter sets, you specify:
   - The conditions required for an EDR to qualify for a system charge offer or discount offer. That is, the list of required EDR fields and values for each filter set.
   - Applicable system products/charge offers and discounts/discount offers, along with their priority and validity dates.

   In Table 2-1, Filter_1 specifies that all GSM wireless calls made to Paris by customers with great credit qualify for a 20% discount.

Table 2-1 Example Filters

Filter name	Filter criteria	System charge offer or system discount offer	Priority
Filter_1	DETAIL.ASS_GSMW_EXT.RECORD_TYPE="Great credit" DETAIL.ASS_CAMEL_EXT.MSC_ADDRESS="Paris"	20% Discount	3
Filter_2	DETAIL.ASS_GSMW_EXT.RECORD_TYPE="Great credit"	Flat rate charge offer	6

About Global Rating

Use global rating to rate every EDR by using the same set of charges. Global rating is performed by the FCT_ GlobalRating module. This module adds an associated charge breakdown record to the EDR. Each charge, or partial charge, from the global charges adds a charge packet.

Global rating is typically used for gathering data used for business planning. For example:

- You can use global rating to calculate an average wholesale charge. You can then compare the average charge with the actual charge to find profit margins for different customer types.
- You can use global rating to calculate an average retail charge. You can use this data to monitor unusual charge amounts that can indicate an error in a rating configuration.

To configure global rating, see "FCT_CustomerRating" in *BRM Pipeline Manager Reference*. Use the **EdrRatePlans** registry entry to specify the global charges. You can use this entry in a semaphore.

About Least Cost Rating

Use least cost rating to rate EDRs by using the charge offer that produces the lowest charge to customers. In this configuration, the pipeline:

1. Rates EDRs by using all products/charge offers associated with an EDR and applies any discounts.
2. Finds the charge and discount that produces the lowest charge.
3. Applies the lowest charge to the customer's balance impact.

> **Note**
>
> An EDR can qualify for *either* least cost rating or overlay promotion. It cannot qualify for both. See "About Overlay Promotions".

You use the following modules to find the lowest possible charge for an event:

- The IRL_LeastCostPerEDR iRule screens EDRs for least cost rating. It compares each EDR against conditions that you specify. When an EDR meets the criteria, the module flags the EDR for least cost rating.
- The FCT_CustomerRating module checks whether an EDR is flagged for least cost rating, and, if it is, generates breakdown records and associated charge packets for each charge offer. All charge packets are listed in order of priority, highest priority first.
- The ISC_LeastCost iScript calculates the total charge for each charge offer and discount and flags the charge that produces the lowest charge. When the lowest charge is generated by a promotional charge offer, the module also calculates the total savings between the promotional charge offer and the lowest priority (base) charge offer.

You configure the modules in a discounting pipeline that includes the FCT_DiscountAnalysis module and the FCT_Discount module. These modules find any applicable discounts for each charge and calculate the discount for each charge packet.

Configuring Least Cost Rating

To set up least cost rating:

1. Specify your least cost rating criteria. See "Specifying the Rules to Qualify for Least Cost Rating". To do so, you edit and configure the IRL_LeastCostPerEDR iRule. See "IRL_LeastCostPerEDR" in *BRM Pipeline Manager Reference*.
2. Configure FCT_CustomerRating for least cost rating by using the **LeastCostRating** entry. See "FCT_CustomerRating" in *BRM Pipeline Manager Reference*.
3. Configure the ISC_LeastCost iScript to find the lowest charge for customers. See "ISC_LeastCost" in *BRM Pipeline Manager Reference*.

Specifying the Rules to Qualify for Least Cost Rating

To specify which EDRs are flagged for least cost rating:

1. Edit the *Pipeline_home***/iScriptLib/iScriptLib_Standard/IRL_LeastCostPerEDR.irl** file to include your rules for least cost rating. *Pipeline_home* is the directory where you installed Pipeline Manager. The variables in the file correspond to positions in the **IRL_LeastCostPerEDR.data** file.

   For example, this rule specifies that EDRs matching the conditions in position 1 will have their DETAIL.CUST_A.LEAST_COST EDR field set to the value in position 2 of the **IRL_LeastCostPerEDR.data** file:

   ```
   CONDITION:

   SetLeastCostDefault ();
   ${1};

   RESULT:
   edrLong(DETAIL.CUST_A.LEAST_COST) = ${2};
   ```

2. Edit the *Pipeline_home*/**iScriptLib/iScriptLib_Standard/IRL_LeastCostPerEDR.data** file to include the conditions required for least cost rating. Use Boolean operators to specify the combinations of market segments and charge offer priorities for testing whether EDRs qualify for least cost rating. You can create any number of entries.

   For example, this rule specifies that only EDRs that meet both of these criteria are flagged for least cost rating (DETAIL.CUST_A.LEAST_COST set to 2):

   - MARKET_SEGMENT EDR field = 1234
   - Charge offer priority greater than 4

   ```
   segmentContains("1234") and priority()>4;2
   ```

3. Configure the FCT_IRules module to point to **IRL_LeastCostPerEDR.irl**.

> **Note**
>
> If you configure a pipeline to use filter sets, make sure IRL_LeastCostPerEDR runs *after* FCT_Filter_Set.

See "IRL_LeastCostPerEDR" in *BRM Pipeline Manager Reference*.

About Calculating the Promotional Savings

You can calculate how much money your customers save when an event is rated with a promotional charge offer rather than a base charge offer by using promotional savings. *Promotional savings* allows you to calculate the difference in charges between a promotional charge offer and the base charge offer, so you can advertise the savings to your customers.

When configured for promotional savings, the pipeline:

1. Rates EDRs by using the highest priority charge offer and the lowest priority (base) charge offer.
2. Calculates the difference between the charge for the promotional charge offer and the charge for the base charge offer.
3. Applies the savings amount to the EDR.

> **Note**
>
> The pipeline applies the savings amount only when the promotional charge offer generates the lowest charge.

You use the following modules to calculate promotional savings:

- The IRL_PromotionalSavingPerEDR iRule screens EDRs for promotional savings. It compares each EDR against your specified criteria. When an EDR meets the criteria, the module flags the EDR for promotional savings.
- The FCT_CustomerRating module checks whether an EDR is flagged for promotional savings, and, if it is, generates separate charge breakdown records and associated charge packets for the highest priority (promotional) and lowest priority (base) charge offer.
- The ISC_LeastCost iScript calculates the charges for the highest and lowest priority products/charge offers and calculates the total savings amount.

See "How Pipeline Modules Process Overlay Promotions" for information about how the processing of overlay promotions and promotional savings are related.

To set up promotional savings:

1. Specify your promotional savings criteria. See "Specifying the Rules to Qualify for Promotional Savings".
2. Configure FCT_CustomerRating for promotional savings by using the **PromotionalSaving** entry. See "FCT_CustomerRating" in *BRM Pipeline Manager Reference*.
3. Configure ISC_LeastCost to calculate the promotional savings amount. See "ISC_LeastCost" in *BRM Pipeline Manager Reference*.

Specifying the Rules to Qualify for Promotional Savings

To specify which EDRs are flagged for a promotional savings calculation:

1. Edit the *Pipeline_home*/**iScriptLib/iScriptLib_Standard/IRL_PromotionalSavingPerEDR.irl** file to include your rules for qualifying for a promotional savings calculation. The variables in the file correspond to positions in the **IRL_PromotionalSavingPerEDR.data** file.

   For example, this rule specifies that EDRs matching the conditions in position 1 will have their DETAIL.CUST_A.PROMOTIONAL_SAVING EDR field set to the value in position 2:

   ```
   CONDITION:

   ${1};

   RESULT:
   edrLong( DETAIL.CUST_A.PROMOTIONAL_SAVING ) = ${2};
   ```

2. Edit the *Pipeline_home*/**iScriptLib/iScriptLib_Standard/IRL_PromotionalSavingPerEDR.data** file to include your conditions for qualifying for a promotional savings calculation. Use Boolean operators to specify the required combination of filter sets and charge offer priorities. You can create any number of entries.

   For example:

   ```
   productName("Standard GSM Telephony", "Highest") and priority("Highest") >
   4 and usageStartTimeGreaterThan("20040101000000", "Highest") and
   serviceType("/service/telco/gsm/telephony", "Highest"); 2

   productName("Standard GSM Telephony", "Base") and priority("Base") < 4 and
   usageStartTimeGreaterThan("20040101000000", "Base") and serviceType("/
   service/telco/gsm/telephony", "Base"); 2
   ```

3. Configure the FCT_IRules module to run the IRL_PromotionalSavingPerEDR iRule. Use the PromotionalSaving entry to specify the path to your IRL_PromotionalSavingPerEDR iRules file.

   For more information, see "IRL_PromotionalSavingPerEDR" in *BRM Pipeline Manager Reference*.

About Overlay Promotions

Overlay promotions allow you to quickly and easily replace your existing products/charge offers and discounts/discount offers with special products/charge offers and discounts/discount offers that take precedence. For example, you may want to offer a special 10% discount for three

months to your existing customers who have paid all their bills on time. To do this, you add a charge offer that offers the same services but costs10% less and has a higher priority than the original charge offer.

You do not need to create new zone models for promotional products/charge offers. Promotional products/charge offers can use the same zone models as standard products/charge offers. For example, you can create two promotional products/charge offers, CALL_USA and CALL_INDIA. You can give CALL_USA priority 4 and CALL_INDIA priority 2, while your base charge offer has priority 0. There is no need to create special zone models for the promotional charge offers. If a customer buys the promotional products/charge offers, BRM uses them to rate calls to the US and India. Calls to other countries will be rated using the base product/charge offer.

The capability to use overlay promotions is built into a pipeline, which:

- Recognizes and uses product/charge offer priorities.
- Allows you to associate multiple products/charge offers with the same service and event.
- Allows you to rate calls using overlay promotions only.

> **Note**
>
> The overlay promotions feature interacts with least cost rating and promotional savings. An EDR can qualify for *either* least cost rating or overlay promotion. It cannot qualify for both.

How Pipeline Modules Process Overlay Promotions

The overlay promotions feature requires no special configuration of pipeline modules. The following modules are involved in the processing of overlay promotions:

- **FCT_CustomerRating**. The module creates a list of associated charge breakdowns and corresponding charge packets ordered from highest to lowest priority. If two products/charge offers have the same priority, the charge offer with the earliest start time is given a higher priority.

  The module then determines whether an EDR qualifies for least cost rating or overlay promotions. If the EDR qualifies for overlay promotions, the module further checks whether the EDR qualifies for promotional savings.

  > **Note**
  >
  > An EDR can qualify for *either* least cost rating or overlay promotion. It cannot qualify for both. If the EDR qualifies for overlay promotion, it may also qualify for promotional savings.

  If the EDR is eligible for least cost rating, FCT_CustomerRating performs these tasks:

  – Enables least cost rating by setting the DETAIL.CUST_A.LEAST_COST field to **2**.

  – Disables promotional savings and overlay promotion by setting the DETAIL.CUST_A.PROMOTIONAL_SAVING and DETAIL.CUST_A.PROMOTION fields to **1**.

If the EDR is eligible for overlay promotions and promotional savings, FCT_CustomerRating performs these tasks:

- Enables promotional savings by setting the DETAIL.CUST_A.PROMOTIONAL_SAVING field to **2**.
- Disables overlay promotions and least cost rating by setting the DETAIL.CUST_A.PROMOTION and DETAIL.CUST_A.LEAST_COST fields to **1**.

If the EDR qualifies for overlay promotions but not promotional savings, FCT_CustomerRating performs these tasks:

- Enables overlay promotions by setting the DETAIL.CUST_A.PROMOTION field to **2.**
- Disables least cost rating and promotional savings by setting the DETAIL.CUST_A.LEAST_COST and DETAIL.CUST_A.PROMOTIONAL_SAVING fields to **1**.

- **FCT_PreRating**. The module populates the zoning information for each charge breakdown record in the EDR:
  - If least cost rating is used, the module finds the zoning information for all charge breakdown records. See "About Least Cost Rating".
  - If promotional savings or overlay promotion is used, the module finds the zoning information for at least one charge breakdown record. See "About Overlay Promotions".
- **FCT_MainRating**. The module functions differently depending on whether the EDR is rated in least cost rating, overlay promotion, or promotional savings mode:
  - **Least cost mode**. The module rates all charge breakdown records. If it fails to rate any charge breakdown record, it returns an error. See "About Least Cost Rating".
  - **Overlay promotion mode**. The module tries to rate charge breakdown records starting with the highest priority record. If rating is successful for a charge breakdown record, the module selects the charge of that record and populates the INTERN_FOUND_PP_INDEX field with the index of that charge. The module then deletes all other charge breakdown records from the container.
  - **Promotional savings mode**. The module tries to rate charge breakdown records, starting with the highest priority record. If rating is successful for a charge breakdown record, the module selects the charge of that record and populates the INTERN_FOUND_PP_INDEX field with the index of that charge. The module continues rating charge breakdown records until it reaches the last record. It then checks for and selects the last successfully rated record. The module keeps the first and last successful charge breakdown records in the EDR container but deletes all others. See "About Calculating the Promotional Savings".

About Rating with Products and Discounts Whose Validity Starts on First Usage

The effective period of an account's products/charge offers and discounts/discount offers can start when the products/charge offers and discounts/discount offers are first used to rate a subscriber's usage.

If products/charge offers or discounts/discount offers are configured to start on first usage, you set up pipeline rating to set their validity periods when first usage occurs.

> **Note**
>
> Balance Elements that are granted by products/charge offers and discounts/discount offers can also start on first usage. To set balance element validity when first usage occurs, you configure the batch discounting pipeline. See "About Setting the Validity of Balance Elements Impacted by Discounts ".

When configured to set validity on first usage, the pipeline:

1. Suspends EDRs that use products/charge offers or discounts/discount offers configured to start on first usage until the product/charge offer and discount/discount offer validity periods are set.

   EDRs are not suspended when first-usage balance element validity is set because the pipeline calculates and stores the validity period in memory. The pipeline uses the stored validity period if it needs to consume balance elements for any other events before the balance validity period has been set in the database.

2. Sends the charge offer or discount offer information to a file in the output stream.

   BRM processes the file, sets the validity periods in the database, and charges any applicable purchase and cycle fees. While their validity periods are being set, the products/charge offers and discounts/discount offers are locked.

3. If the event is discounted and the discount balance impact consumes a balance element balance that starts on first usage, the pipeline sends the balance element information to a file in the output stream.

   BRM processes the file and sets the balance element validity periods in the database. If the validity period of all first-usage balance elements are configured to be synchronized, BRM also sets the validity period of those balance elements.

4. Rates the events when the EDRs are recycled.

If the pipeline transaction is rolled back or canceled, the validity period of any charge offer or discount offer that was set in the transaction is unset.

BRM uses the EAI framework and Oracle DM to synchronize charge offer and discount offer validity periods in the BRM database and pipeline memory.

The following modules are involved in setting the effective periods of products/charge offers and discounts/discount offers:

- The FCT_FirstUsageNotify module. See "About Suspending EDRs for Products and Discounts that Start on First Usage".
- The DAT_AccountBatch module determines if an account's products/charge offers and discounts/discount offers start on first usage, determines the state of the validity period, and calculates the validity periods based on the EDR timestamp. It sends this information to the FCT_Account module.

  The state of a validity period is used for coordinating the validity-setting and EDR-suspension processes. The state can be one of the following:

  - **NOT_FirstUsage**: Indicates that the charge offer or discount offer's validity period is already set in the database.
  - **NEW_FirstUsage**: Indicates that the charge offer or discount offer is configured to start on first usage and that its validity period has not yet been initialized.

  - **INIT_FirstUsage**: Indicates that the charge offer or discount offer's validity period has been initialized but has not yet been stored in the database.

  If the validity periods of all first-usage balance elements in the bundle should be synchronized, DAT_AccountBatch retrieves a list of the balance groups that have first-usage validity and are associated with the products/charge offers or discounts/discount offers in the bundle.

- The FCT_ApplyBalance module. See "About Setting the Validity of Balance Elements Impacted by Discounts ".
- The OUT_GenericStream module and related modules. See "Configuring Pipeline Output for First-Usage Products, Discounts, and Balance Elements".

To set the validity for products/charge offers and discounts/discount offers that start on first usage, you must also perform these tasks:

- Configure BRM recycling to recycle suspended EDRs. For more information, see "Setting Up Recycling for Events whose Product or Discount Validity Starts on First Usage".
- Use Universal Event (UE) Loader to update the validity periods in the BRM database. See "About Updating Validity Period Information in the BRM Database".

About Suspending EDRs for Products and Discounts that Start on First Usage

Use the FCT_FirstUsageNotify module to suspend EDRs while BRM sets the validity periods of products/charge offers and discounts/discount offers that start on first usage. See "FCT_FirstUsageNotify" in *BRM Pipeline Manager Reference*.

> **Note**
>
> To recycle and rate EDRs that are suspended while validity is being set, you must configure BRM standard recycling. See "Configuring Standard Recycling" in *BRM Suspending and Recycling Pipeline EDRs*.

If a call details record (CDR) uses both a charge offer and a discount offer that start on first usage for the first time, the EDR is suspended and recycled twice: once to set the charge offer validity period and again to set the discount offer validity period. This is because the charge offer may grant balance elements that can be impacted by the discount offer, so the charge offer's purchase and cycle events need to be processed before the discount offer is evaluated.

The FCT_FirstUsageNotify module performs the following tasks:

1. Checks whether the products/charge offers and discounts/discount offers used to rate an EDR have validity periods that start on first usage.
2. If a product/charge offer and discount/discount offer starts on first usage, flags the EDR for suspense and recycling by setting the ERR_FIRST_USAGE_VALIDITY_NEEDS_INITIALIZING error code and the **FirstUsageValidity** recycle key in the EDR.
3. Notifies the DAT_AccountBatch module that the validity period of the charge offer or discount offer is being set.
4. Sends the charge offer and discount offer validity information to a separate output stream.

5. Continues to flag for suspense and recycling subsequent EDRs that use the products/charge offers or discounts/discount offers with first-usage validity until the validity period is set.

FCT_FirstUsageNotify predetermines whether the FCT_Reject module will reject an EDR. FCT_FirstUsageNotify does not process EDRs that will be rejected because product/charge offer, discount/discount offer, or balance element validity should not be set for EDRs that will be otherwise suspended.

Configuring Pipeline Output for First-Usage Products, Discounts, and Balance Elements

In the batch rating pipeline, you configure two output streams: one for files containing products/charge offers and discounts/discount offers that start on first usage, and another for files containing balance elements that start on first usage.

To create the first-usage output streams, configure the following:

- The OUT_GenericStream module. See "Configuring First-Usage Output Streams ".
- The DataDescription section of the registry. See "Specifying the First-Usage Format and Mapping Files in the DataDescription Registry".

Configuring First-Usage Output Streams

Configure the OUT_GenericStream module to write the first-usage products/charge offers, discounts/discount offers, and balance elements to an output file.

The default output grammar description files are:

- For products/charge offers and discounts/discount offers, **FirstUsageNotify_OutGrammar.dsc**.
- For balance elements, **FirstUsageResource_OutGrammar.dsc**.

In the EXT_OutFileManager section of the registry, specify the temporary file specifications as follows:

- For first-usage products/charge offers and discounts/discount offers:

```
OutputPath = ./data/out/firstUsage/prod_disc
OutputPrefix = test_PROD
OutputSuffix = .out_1
TempPrefix = .
TempDataPath    = ./data/out/firstUsage/prod_disc
TempDataPrefix  = prod.tmp.
TempDataSuffix  = .data_1
```

- For first-usage balance elements:

```
OutputPath = ./data/out/firstUsage/resources
OutputPrefix = test_RES
OutputSuffix = .out_1
TempPrefix = .
TempDataPath    = ./data/out/firstUsage/resources
TempDataPrefix  = res.tmp.
TempDataSuffix  = .data_1
```

See "OUT_GenericStream" in *BRM Pipeline Manager Reference*.

Specifying the First-Usage Format and Mapping Files in the DataDescription Registry

Configure the first-usage stream formats, input mapping, and output mapping in the batch pipeline DataDescription registry section as follows:

```
DataDescription
{
   Standard
   {
   ModuleName = Standard
   Module
   {
   StreamFormats
   {
      FIRST_USAGE_NOTIFY_OUTPUT=./formatDesc/Formats/FirstUsageNotify/FirstUsageNotify.dsc
      FIRST_USAGE_RESOURCES=./formatDesc/Formats/FirstUsageNotify/FirstUsageResource.dsc
   }
   . . .
   OutputMapping
   {
      FIRST_USAGE_PROD_DISC=./formatDesc/Formats/FirstUsageNotify/FirstUsageNotify_OutMap.dsc
      FIRST_USAGE_RESOURCES=./formatDesc/Formats/FirstUsageNotify/FirstUsageResource_OutMap.dsc

   }
   }
   }
}
```

For more information, see "Configuring Output for Rated Events".

About Updating Validity Period Information in the BRM Database

You use UE Loader to update validity periods in the BRM database when first usage occurs.

You use the ConfigurableValidityHandler batch handler to run the UE Loader utility (**uel**).

Perform the following tasks to update validity periods for first-usage products/charge offers, discounts/discount offers, and balance elements:

- Load the first-usage validity Universal Event Mapper templates. See "Loading the First-Usage Validity Templates".
- Configure the ConfigurableValidityHandler batch handler. See "Configuring the ConfigurableValidityHandler Batch Handler".
- Configure Batch Controller. See "Configuring Batch Controller to Start the ConfigurableValidityHandler Batch Handler".

Loading the First-Usage Validity Templates

BRM provides two Universal Event (UE) Mapper event import templates that are used by UE Loader to process the first-usage product/charge offer, discount/discount offer, and balance element files output by the OUT_GenericStream module:

- **FirstUsageProductsDiscounts.xml**: This template defines the format of the first-usage product/charge offer and discount files.
- **FirstUsageResources.xml**: This template defines the format of the first-usage balance element files.

You must load these templates into the BRM database by running the **pin_uei_deploy** utility.

For example:

```
pin_uei_deploy -c -t FirstUsageProductsDiscounts -i BRM_home/formatDesc/Formats/
FirstUsageNotify/FirstUsageProductsDiscounts.xml

pin_uei_deploy -c -t FirstUsageResources -i BRM_home/formatDesc/Formats/FirstUsageNotify/
FirstUsageResources.xml
```

Configuring the ConfigurableValidityHandler Batch Handler

BRM provides the ConfigurableValidityHandler batch handler for loading first-usage validity data. You must configure ConfigurableValidityHandler to run the following utilities:

- **pin_rel:** This utility loads data for events that are rated in batches.
- **uel**: This utility loads validity period data for products/charge offers, discounts/discount offers, and balance elements when they are used for the first time.
- **pin_load_rerate_jobs**: You must also configure to run this utility if you configured Pipeline Manager to detect and rerate events that are rated out of order. This utility creates rerate jobs for the events that were rated out of order.

For more information, see "About Using a Single Batch Handler to Run Multiple Loading Utilities" in *BRM Setting Up Pipeline Pricing*.

To configure the ConfigurableValidityHandler batch handler, you edit the handler's configuration file (*BRM_home***/apps/pin_rel/ConfigurableValidityHandler_config.values**). You specify handler, processing, and staging directories for each loading utility that this batch handler runs.

Configuring Batch Controller to Start the ConfigurableValidityHandler Batch Handler

Batch Controller polls the pipeline output directories and starts the ConfigurableValidityHandler batch handler when a data file is ready to be loaded.

> **Note**
>
> If you use the ConfigurableValidityHandler batch handler, do not configure Batch Controller to run a separate instance of the handlers that load pipeline batch-rated events, out-of-order rerating requests, or first-usage validity data. (You may configure a separate instance of the Rated Event (RE) Loader handler that loads suspended events into **/suspended_usage** objects.) If you have already configured Batch Controller to run the out-of-order event handler (OODHandler), remove those entries or comment them out.

To configure Batch Controller to start the ConfigurableValidityHandler batch handler:

1. Open the Batch Controller **Infranet.properties** file in *BRM_home***/apps/batch_controller**.
2. Edit the file to include entries for the ConfigurableValidityHandler batch handler.

> **Note**
>
> - If you have already configured the Batch Controller **Infranet.properties** file for RE Loader, you can use the existing RE Loader entries and most of their values or you can change them. Entries that you must change are marked with an asterisk (*).
> - If you have not yet configured the Batch Controller **Infranet.properties** file for RE Loader, you must add ConfigurableValidityHandler values for all of the following entries.

Table 2-2 lists the entries you must set and the default values used for RE Loader.

Table 2-2 Entries for RE Loader

Entry	Description
batch.random.events	Identifies this specific configuration for triggering Batch Controller. For example: `batch.random.events CdrFileEvent`
event_name.**name**	(Optional) A name for the configuration identifier. For example: `CdrFileEvent.name CdrFileEvent`
event_name.**handlers**	The batch handler identifier. For example: `CdrFileEvent.handlers ConfigurableValidityHandler` The default is **relHandler**.
event_name.**file.location**	The full path to the pipeline output directory where the rated-event files are deposited. For example: `CdrFileEvent.file.location /export/portal/integRate`
event_name.**file.pattern**	The rated-event output file name. You can use an asterisk (*****) to represent zero or more characters in the file name. No other wildcards (metacharacters) are supported. For example: `CdrFileEvent.file.pattern cdr*.dat`

***handler_name*.name**	The name of the batch handler that is run. For example:
	`ConfigurableValidityHandler.name ConfigurableValidityHandler`
	Note: If you did not change the default RE Loader value for *event_name.handlers*, this entry should be:
	`relHandler.name ConfigurableValidityHandler`

Table 2-2 (Cont.) Entries for RE Loader

Entry	Description

handler_name.**max.at.lowload.time**
handler_name.**max.at.highload.time**
	The number of batch handler instances that can run concurrently during periods of low load and high load usage. Typical default settings are **6** at low load and **3** at high load. For example:
	`ConfigurableValidityHandler.max.at.highload.time 3`
	`ConfigurableValidityHandler.max.at.lowload.time 6`

***handler_name*.start.string**	The full path to the ConfigurableValidityHandler handler. For example:
	`ConfigurableValidityHandler.start.string BRM_home/apps/pin_rel/ConfigurableValidityHandler.pl`
	Note: If you did not change the default RE Loader value for event_name.handlers, this entry should be:
	`relHandler.start.string BRM_home/apps/pin_rel/ConfigurableValidityHandler.pl`

3. Save and close the file.
4. Stop and restart Batch Controller.

Setting Up Recycling for Events whose Product or Discount Validity Starts on First Usage

To suspend and recycle EDRs while product/charge offer and discount/discount offer validity is being set, you must configure BRM standard recycling. See "Configuring Standard Recycling" in *BRM Suspending and Recycling Pipeline EDRs*.

For information about standard recycling, see "About Standard Recycling" in *BRM Suspending and Recycling Pipeline EDRs*.

To recycle events suspended due to first-usage validity, you run the **pin_recycle** utility. Use the **-k** parameter and specify the **FirstUsageValidity** recycle key. For example:

```
pin_recycle -k FirstUsageValidity
```

For more information, see "Using Standard Recycling to Recycle Suspended EDRs" in *BRM Suspending and Recycling Pipeline EDRs* and "pin_recycle" in *BRM Pipeline Manager Reference*.

About First-Usage Validity for Events Rated Out of Order

In the batch rating pipeline, if events are rated in a different order than they occurred, validity periods that are based on first usage may be incorrectly set. This can happen if the first event rated, which initiated the validity period, wasn't actually the first usage event. To correct the validity periods, you must rerate the events. Rerating corrects the order of the events and resets the validity period based on the actual first-usage event.

About Customer Rating

Customer rating assigns a charge to an EDR based on customer data. The FCT_CustomerRating module performs customer rating. The module creates an associated

charge breakdown record and one charge packet, which includes the charge code. The FCT_MainRating module uses the charge code to rate the event.

To assign the charge, the FCT_CustomerRating module does the following:

- If the service that generated the event includes a service-level charge extended rating attribute (ERA), that charge is used.

  If you're using subscription services, and a subscription service and member service both own a service-level charge ERA, the member service's ERA has priority and is used for selecting the charge.
- If there is no service-level charge ERA, but there is an account-level charge ERA, that charge is used.
- If there is no charge ERA, the charge associated with the last product/charge offer found is used when there is more than one product/charge offer available.

To configure customer rating, see "FCT_CustomerRating" in *BRM Pipeline Manager Reference*.

Assigning a Default Charge and Default Segment for Customer Rating

When you configure the FCT_CustomerRating module, you can assign a default charge and default segment to use if no customer information for the A number is found.

- Use the **DefaultRateplan** entry to specify the default charge name in case there is not enough information to assign a charge. You can change this value by using a semaphore.
- Use the **DefaultSegment** entry to specify if segment rating is used, the default segment to use if no segment is found. You can change this value by using a semaphore.

See "FCT_CustomerRating" in *BRM Pipeline Manager Reference*.

About Customer Rating and Service Level Agreements

You define service-level agreement (SLA) codes in Pricing Center or Pipeline Configuration Center (PCC) to map service level agreements (SLAs) to a usage-scenario (USC) group, rate-service class (RSC) group, and rule set. The SLA mapping is stored in the IFW_SLA database table.

During customer rating, if the FCT_CustomerRating module finds an SLA code, the module looks up the code in the IFW_SLA database table and adds the following data to the charge packet:

- RSC group
- USC group
- Rule set

Pipeline Manager can use any one of these to determine the rate for the service level usage:

- The RSC group is used by the FCT_RSC_Map module to find the RSC map. FCT_RSC_Map maps the usage class, usage type, service code, and impact category to a new service class. See "About Setting Up RSC Mapping".
- The USC group is used by the FCT_USC_Map module to find the USC map. FCT_USC_Map maps the usage class, usage type, service code, service class, and zone to a new usage type and impact category.

About Rate-Service Class Mapping

You map a rate-service class (RSC) to perform rating based on the quality of service (QoS) when you set up service-level agreements (SLAs). RSC mapping is performed by the FCT_RSC_Map module, which maps the usage class, usage type, service code, and impact category to a new service class.

You create RSC map groups to provide a different service class for each level of service quality. You link the RSC map group to an SLA code in Pricing Center or Pipeline Configuration Center (PCC). The FCT_CustomerRating module looks up the SLA code and adds the associated RSC group to the charge packet (see "About Customer Rating and Service Level Agreements"). FCT_RSC_Map uses the RSC group to evaluate the EDR and find the correct RSC map.

The RSC map group is specified in the INTERN_SLA_RSC_GROUP field of the EDR. This field is filled in by FCT_CustomerRating when the product/charge offer includes a service-level agreement ERA. If INTERN_SLA_RSC_GROUP is empty, the default RSC group is used. You specify the default RSC group in the **DefaultRscGroup** entry of the FCT_RSC_Map registry.

About Setting Up RSC Mapping

To set up RSC mapping, do the following:

1. Use Pricing Center or Pipeline Configuration Center (PCC) to create RSC maps and RSC map groups.
2. Use Pricing Center or PCC to create SLA codes and link them to RSC map groups.
3. Configure FCT_RSC_Map. See "FCT_RSC_Map" in *BRM Pipeline Manager Reference*.

About RSC Maps

To assign a new service class to the EDR, FCT_RSC_Map reads data from the EDR and evaluates it according to the RSC map. An RSC map includes one or more mapping rules that specify the data that must be matched to apply the new service class.

When you create RSC maps, you create a mapping rule for each new service class. You can also define the order in which the rules are evaluated. The new service class is derived from the first rule that matches the data. If no matching rule is found, FCT_RSC_Map uses the default RSC map as defined in the registry. If no default value exists, no mapping is performed.

You can use the following EDR data to create an RSC mapping rule:

- The charge used to rate the EDR
- The QoS requested
- Usage class
- Usage type
- Service code
- Service class
- Impact category

When you create the mapping rules, you can use regular expressions. For information on the regular expressions you can use, see "About Using Regular Expressions when Specifying the Data to Extract".

To create a valid mapping, the data in the EDR must match all of the mapping data.

About Main Rating

The FCT_MainRating module carries out the pipeline rating functionality.

When an EDR is ready for rating, it includes all the data needed by FCT_MainRating; for example, the service class, usage class, zone, and charge. The module uses the charge to rate the EDR. To rate the EDR, the module uses information about dates, times, and the pricing to apply charges. When rating is finished, the EDR contains complete charge breakdown data, including charge packets with charges.

FCT_MainRating uses criteria in the charge configuration that apply to an EDR to find the correct pricing to use for rating. If a single event is rated by using different time periods, such as peak and off-peak, more than one pricing can be used. If an event is mapped to a price selector, the model selector's rules are evaluated to choose a pricing.

If the charge configuration includes an alternative pricing, the module creates a new charge packet. The EDR is rated again by using the alternative pricing. The charge packet is flagged with the A (Alternative) pricing type.

During main rating, the module checks for the RUM group associated with the service code. At least one charge packet is added to the EDR for each RUM assigned to the RUM group. A single charge packet is generated for each time period, RUM, and balance element that is used. (The balance elements are defined in the pricing.)

To configure main rating, see "FCT_MainRating" in *BRM Pipeline Manager Reference*.

About Rate Adjustment

You use rate adjustments to provide discounts based on date, time, service, and other event attributes.

To set up rate adjustment, do the following:

1. Create rules that specify which EDRs to adjust and how to adjust them. You have two options:
   - Create rate adjustment rules in Pricing Center or Pipeline Configuration Center (PCC). In this case, the rate adjustment data is stored in the Pipeline Manager database.
   - Create a file that defines usage scenario maps. In this case, the FCT_RateAdjust module reads the file. For information on creating the file, see "Creating a Rate Adjustment Rules File".
2. Configure FCT_RateAdjust. See "FCT_RateAdjust" in *BRM Pipeline Manager Reference*.

Creating a Rate Adjustment Rules File

The rate adjustment rules can be defined in an ASCII file:

- Each rule consists of a list of fields. The following table describes the meaning of each field.
- Every new line defines an adjustment rule.
- Fields are separated by semicolons (;).
- Comment lines start with #.

- Empty lines are allowed.

> **Note**
>
> The value of the field rank is ignored. The evaluation order of the rules is given by the order of the rules within the file.

Table 2-3 lists the fields in the file.

Table 2-3 Fields in a Rate Adjustment File

Position	Field	Description
1	Rank	Specifies the evaluation order of the rules. This is ignored because the evaluation order is specified within the file.
2	Rate plan	Specifies the charge to adjust.
3	Rate plan version	Specifies the charge version.
4	Valid from	Specifies the start date for the rate adjustment. This can be either a date with the format *YYYYMMDD* or a weekday with an optional timestamp; for example: • 19990524 • SAT • MON16:00 If the field is left empty, the earliest possible date (19010101) is used.
5	Valid to	Specifies the end date for the rate adjustment. This can be either a date with the format *YYYYMMDD* or a weekday with an optional timestamp. If the field is left empty, the latest possible date (20370205) is used.
6	Time from	Specifies the start time for the rate adjustment. The format is *HH*:*MM*. The default is **00:00**.
7	Time to	Specifies the end time for the rate adjustment. The format is *HH*:*MM*. Example: To set up a discount that is valid on weekends between 13:00 and 14:00, you have to set the following: • **ValidFrom** = SAT • **ValidTo** = SUN • **TimeFrom** = 13:00 • **TimeTo** = 14:00
8	Quantity value	Specifies the maximum quantity value for an EDR container. If this maximum is exceeded, the mapping rule will not be used. If this field is left empty or if a 0 is specified, the rule is valid for every quantity value. Example: You can use this entry to avoid discounting for calls longer than 120 seconds by setting the field to 120.
9	Usage class	Specifies the compare pattern for the usage class.
10	Usage type	Specifies the compare pattern for the usage type.

Table 2-3 (Cont.) Fields in a Rate Adjustment File

Position	Field	Description
11	Service code	Specifies the compare pattern for the service code.
12	Service class	Specifies the compare pattern for the service class.
13	Impact category	Specifies the compare pattern for the impact category.
14	Source network	Specifies the compare pattern for the source network.
15	Destination network	Specifies the compare pattern for the destination network.
16	Discount type	Specifies the discount type.
17	Discount value	Specifies the discount value.
18	Comment	Specifies the rate adjustment name.

About Consolidation for BRM Billing

The FCT_BillingRecord module consolidates charge packets and discount packets into an associated BRM billing record in the EDR. This data is loaded as a rated event by RE Loader.

The associated BRM billing record includes the POIDs of the **/account** object and the **/service** object and the POID of the item that receives the balance impact. If an event affects more than one customer balance, an associated BRM billing record is created for each balance.

An associated BRM billing record can contain one or more *balance impact packets*. The data in a balance impact packet is loaded into an **/event** object balance array. Therefore, the data includes information about the charged amount, the charge, and balance elements.

The balance impact packet also includes data in the PIN_INFO_STRING field. This field contains the information about the individual charge packets.

Each balance impact packet includes data for one balance impact per balance element. If there are different G/L IDs for the same balance element, a balance impact packet is created for each G/L ID.

To configure FCT_BillingRecord, see "FCT_BillingRecord" in *BRM Pipeline Manager Reference*.

> **Note**
>
> Don't use FCT_BillingRecord in a CIBER roaming revenue assurance environment. For more information, see "Billing Consolidation with CIBER Roaming and Revenue Assurance".

How the FCT_BillingRecord Module Works

FCT_BillingRecord uses the data in the EDR to determine if the charges should be included in the associated BRM billing record. To do so, the module checks the following data. If any of these do not match, no associated billing record is created.

- The record type must be 981. This record type is created by the FCT_CustomerRating module for customer rating.

- The charge packet must use the following:
  - The standard pricing type: PRICEMODEL_TYPE = S
  - A retail charge: RATEPLAN_TYPE = R
  - A currency type that matches the currency type specified in the FCT_BillingRecord module registry. The options are Home, Billing, and Rating.
  - A currency that matches one of the entries specified in the FCT_BillingRecord module registry. See "FCT_BillingRecord" in *BRM Pipeline Manager Reference*.

If the charge packet meets the criteria, the module sums the amount, discount, and quantity for each BRM balance element and creates the associated BRM billing record and one or more balance impact packets.

For balance monitoring, FCT_BillingRecord generates a monitor packet for each monitor group.

To get data, FCT_BillingRecord connects to the following data modules:

- The DAT_AccountBatch module, which provides data about items.
- The DAT_ItemAssign module, which provides data about items assigned for sponsorship events.
- The DAT_Currency module, which provides data for converting currency symbols to numeric values.
- The DAT_BalanceBatch module, which provides the **ObjectCacheType** value from the ***/balance_group*** object.

Billing Consolidation with CIBER Roaming and Revenue Assurance

For CIBER roaming and revenue assurance, use the ISC_PostRating iScript instead of FCT_BillingRecord.

ISC_PostRating adds all the retail and wholesale charges and puts them in the DETAIL.RETAIL_CHARGED_AMOUNT_VALUE and DETAIL.WHOLESALE_CHARGED_AMOUNT_VALUE fields.

> **Note**
>
> ISC_PostRating and FCT_BillingRecord perform similar billing consolidation, but ISC_PostRating doesn't load balance impact data into the database.

See "ISC_PostRating" in *BRM Pipeline Manager Reference*.

How the ISC_PostRating iScript Works

ISC_PostRating sets the following attributes:

- Wholesale and retail impact category
- Charged amount value
- Amount currency
- Tax treatment

for an EDR based on these values specified in the Pipeline Manager registry:

- Balance Element type
- Pricing type
- Currency type

> **Note**
>
> This iScript accesses only EDR fields. It doesn't access the database.

Adding Pipeline Rating Data to an Invoice

When you rate usage by using pipeline rating, information about how events are rated are stored in the EDR. You can display this information on invoices. For example, if a call spans two rates, for peak and off-peak time, you can display the rate used for each part of the call. For example:

Table 2-4 Displaying Pipeline Rating Data

Date/time	Called number	Duration	Average rate per unit	Total charge
12/12/2003 1200	4085551212	10min	$0.125	$1.25
12/12/2003 1200	4085551212	5min	$0.15	$0.75
12/12/2003 1200	4085551212	5min	$0.10	$0.50

In the example shown in Table 2-4, the data is stored in the INTERN_PRICE_MDL_STEP_INFO EDR field.

To add pipeline rating invoice data to your invoices, you need to configure the OUT_GenericStream module **AddInvoiceData** registry entry to add the data to the BRM billing record.

Specifying Invoice Data from Pipeline Manager and Custom Applications

If you process events that originate in Pipeline Manager or a custom application, you can create a template to specify the event invoice data so that it can be included in your invoices.

BRM provides a default template file (**pin_invoice_data_map**) located in *BRM_home***/sys/data/config**. This file includes a default **INTEGRATE** template for Pipeline Manager invoice data. You can modify the **INTEGRATE** template or add new templates to the file.

To use this feature, you must do the following:

- Load the new invoice data template. See "Loading the Invoice Data Map Templates".
- Enable event caching in the Connection Manager (CM). See "Enabling Event Caching".
- Configure the output module to add invoice data. See "Adding Invoice Data to Pipeline Output".

Using Data Map Templates

Templates in the **pin_invoice_data_map** file define the fields in invoice data records. If you use different invoice record formats, you can create a template for each record format. The data map file can include any number of templates.

When you define templates, you specify the BRM flist fields that map to the invoice record fields. When the invoice data is processed, the fields are passed in an flist to the invoicing opcodes for processing.

The default **INTEGRATE** template defines Pipeline Manager invoice data. You can modify the **INTEGRATE** template or create new templates.

To create or modify templates in the **pin_invoice_data_map** file:

1. Open the *BRM_home***/data/config/pin_invoice_data_map** file.
2. Change the **INTEGRATE** template or add a new template to the end of the file. Use the following syntax:

   ```
   ID template_name
   field_level   field_name
   field_level   field_name
   field_level   field_name
   ...
   ```

   where:

   - *template_name* is the name of the template.
   - *field_level* is the level of the field in the flist.
   - *field_name* is the associated BRM field.

   For example:

   ```
   ID INTEGRATE
   0 PIN_FLD_CALLING_NUMBER
   0 PIN_FLD_CALLED_NUMBER
   . . .
   0 PIN_FLD_BAL_IMPACTS
   1 PIN_FLD_RATE_TAG
   1 PIN_FLD_AMOUNT
   ```

3. Save and close the file.

The order of the fields in the template must correspond to the order of the fields in the invoice record. Any custom invoice data records that you process must follow these rules:

- Fields must be separated by a pound symbol: #
- Arrays and substructs must be enclosed in angle brackets: **< >**
- Each balance impact element must end with a pipe symbol: **|**

> **Note**
>
> The pipe is optional after the last element.

- The first field in the record must be the name of the corresponding invoice data template and be preceded by the @ symbol.

For example: **@INTEGRATE**

- The template name in the record must match the template name in the **load_pin_invoice_data_map** file.

Loading the Invoice Data Map Templates

After defining invoice data map templates, load them into the BRM database.

> **Note**
>
> - When you run **load_pin_invoice_data_map**, it overwrites the existing invoice data templates. If you are updating a set of templates, you cannot load new templates only. You must load complete sets of invoice data templates each time you run **load_pin_invoice_data_map**.
> - To connect to the BRM database, **load_pin_invoice_data_map** needs a configuration file in the directory from which you run the utility.
>
> If you defined a custom field for invoicing, you must add the full path name of the mapping file to the **load_pin_invoice_data_map** utility's **pin.conf** file.

To load the data map templates:

1. Go to the directory that contains the utility's configuration file.
2. Run the following command, which loads the data map template:

```
load_pin_invoice_data_map -d -v pin_invoice_data_map
```

Enabling Event Caching

Pipeline Manager caches invoice data before sending it to BRM for processing. Therefore, you must enable event caching to include Pipeline Manager invoice data in your invoices.

To enable event caching:

1. Go to *BRM_home*/**sys/data/config**.
2. Use the following command to create an editable XML file from the invoicing instance of the **/config/business_params** object:

```
pin_bus_params -r BusParamsInvoicing bus_params_Invoicing.xml
```

   This command creates an XML file named **bus_params_Invoicing.xml.out** in your current directory. If you do not want this file in your current directory, specify the path as part of the file name.

3. In **bus_params_Invoicing.xml.out**, set **EventCache** to **enabled**:

```
<EventCache>enabled</EventCache>
```

> **Caution**
>
> BRM uses the XML in this file to overwrite the existing instance of the **/config/business_params** object. If you delete or modify any other parameters in the file, these changes affect the associated aspects of the BRM configuration.

4. Save and exit the file.
5. Rename the **bus_params_Invoicing.xml.out** file to **bus_params_Invoicing.xml**.
6. Use the following command to load your changes into the **/config/business_params** object:

   ```
   pin_bus_params bus_params_Invoicing.xml
   ```

   You should run this command from the *BRM_home***/sys/data/config** directory, which includes support files used by the utility. To run it from a different directory, see "pin_bus_params" in *BRM Developer's Guide*.
7. Read the object with the **testnap** utility or the Object Browser to verify that all fields are correct.

   For general instructions on using **testnap**, see "Using the testnap Utility to Test BRM" in *BRM Developer's Guide*. For information on how to use Object Browser, see "Reading Objects" in *BRM Developer's Guide*.
8. Stop and restart the CM.

   For more information, see "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

Adding Invoice Data to Pipeline Output

To add data from a pipeline to your invoices, configure the OUT_GenericStream module to add the data to the BRM billing record.

To output invoice data, add the following registry parameter to all OUT_GenericStream modules:

```
AddInvoiceData = TRUE
```

This parameter is read by the output grammar. When set to **TRUE**, the output module adds invoice data to each BRM billing record.

Using the pin_virtual_time Utility with Pipeline Manager

If you use **pin_virtual_time** to test charging and billing, you need to set the **VirtualTime** parameter for the DAT_BalanceBatch module to **True** to ensure that balances are calculated correctly.

3

Configuring EDR Input Processing

Learn how to set up input processing for the Oracle Communications Billing and Revenue Management (BRM) Pipeline Manager.

Topics in this document:

About the Input Process

To process incoming data, Pipeline Manager input modules convert data into an internal EDR format understood by the pipeline function modules. The input data is contained in files, such as CDRs for telco rating.

The input process works as follows:

1. Your mediation system automatically places CDR files into a directory.

   If you start a pipeline and the directory already includes input files, they are processed according to the last-modified timestamp.

2. The input module uses the stream format description file to create separate records from the data. For example, the data is separated into HEADER records, DETAIL records, and TRAILER records. DETAIL records can include data for one service only (for example, GSM *or* GPRS).

3. The input module uses the input grammar file to process each record. The module verifies that the syntactical order for each record is correct. For example, if a particular field is supposed to include 10 characters, the input module uses the grammar file to check that. It also uses the grammar file to normalize data, such as the A number.

   If an error is found, processing stops and an error message is logged.

4. The input module creates an EDR container for the data. See "About EDRs" in *BRM Setting Up Pipeline Pricing*.

5. The input module uses an input mapping file to copy data from the external file into the appropriate EDR container fields.

> **Note**
>
> A separate input mapping file is required to process each external data format. See "Setting Up an Input Mapping File".

6. The input module puts the EDR into the input buffer for processing by the function modules.

> **Note**
>
> A separate input grammar file is required to process each external data format. See "Setting Up an Input Grammar File".

The following examples show how the A number in a CDR is mapped to a rating EDR container.

> **Note**
>
> These examples show only selected portions of the stream format description, grammar, and mapping files.

The following example shows part of a stream format description file. This section of the file describes how a DETAIL record is formatted and lists the fields in the record (fields are separated by a semicolon). The first field (SERVICE) stores the service code, the second (A_NUMBER) stores the A number, and so forth. Data in each of the fields must be of type **AscString()**.

```
DETAIL(SEPARATED)
{
Info
{
Pattern = ".*\n";
FieldSeparator = ';';
RecordSeparator = '\n';
}
SERVICE     AscString();
A_NUMBER    AscString();
B_NUMBER    AscString();
...
```

The next example shows part of an input grammar file. The first two lines create a new block of data in an EDR container (**edrNew**) and specify the location in an external file from which the data should be copied (**edrInputMap**). Inside the new block, the next two lines normalize the A number and then add a field to the EDR container called DETAIL.A_NUMBER.

```
edrNew( DETAIL, CONTAINER_DETAIL );
edrInputMap( "SAMPLE.DETAIL.STD_MAPPING" );
...
number = normalizeNumber( edrString( DETAIL.A_NUMBER ), "00", 0 );
edrString( DETAIL.A_NUMBER ) = number;
```

The next example shows part of an input mapping file. In this example, the A_NUMBER defined in the stream format description example is mapped to the DETAIL.A_NUMBER field

defined in the input grammar example. Note how the nested blocks of data correspond to the **edrInputMap** entry (**SAMPLE.DETAIL.STD_MAPPING**) in the input grammar example.

```
SAMPLE
  DETAIL
  {
    HDR_MAPPING
    {
    "010"       -> HEADER.RECORD_TYPE;
    ...
  }
  TRL_MAPPING
  {
    "090"       -> TRAILER.RECORD_TYPE;
    ...
  }

  STD_MAPPING
  {
    "020"       -> DETAIL.RECORD_TYPE;
    0           -> DETAIL.DISCARDING;
    "00"        -> DETAIL.A_MODIFICATION_INDICATOR;
    0           -> DETAIL.A_TYPE_OF_NUMBER;
    "0"         -> DETAIL.A_NUMBERING_PLAN;
    A_NUMBER    -> DETAIL.A_NUMBER;
...
```

You can map one field in an external file to multiple fields in an EDR container. The following example shows part of the same input mapping file. In this part of the file, however, the data block for the GSM service maps the A_NUMBER to a different field:

```
GSMW_MAPPING
  {
  "520"       -> DETAIL.ASS_GSMW_EXT.RECORD_TYPE;
  A_NUMBER    -> DETAIL.ASS_GSMW_EXT.A_NUMBER_USER;
  B_NUMBER    -> DETAIL.ASS_GSMW_EXT.DIALED_DIGITS;
...
  }
```

About Setting Up Input Processing

To set up input processing, do the following:

1. **(CDR processing only)** Define a CDR file input directory in your pipelines. Configure your mediation system to put the files in this folder automatically.

   > **Note**
   >
   > You can configure your system to route CDR files from a single input directory to multiple identical pipelines.

2. Set up a stream format description file. You can start with the sample files that are provided. See "Creating a Stream Format Description File".
3. Set up an input mapping file. You can start with the sample files that are provided. See "Setting Up an Input Mapping File".
4. If necessary, set up an input grammar file. In most cases, you do not need to modify the default grammar file. If you modify an EDR container (for example, if you add fields), you

might need to modify the input grammar to ensure that data in your EDR containers is correctly formatted. See "Setting Up an Input Grammar File".

> **Note**
>
> If you customize an EDR container description, you must ensure that your customizations do not affect existing module functionality. Many modules require data from default EDR fields.

5. Configure these input sections in the registry:
   - The **DataDescription** section. See "Configuring the Input DataDescription Registry Section".
   - The **InputBuffer** section in the **Pipeline** section.
   - The **Input** section. See "Configuring the Input Section in the Registry".

The sample Pipeline Manager registry files include stream format description, input mapping, and input grammar files that convert data using the rating EDR and TAP formats.

About Input Processing File Types

Input processing uses the following types of files:

- **Input file**: The file from the external system.
- **Temporary file**: The same file as the input file, but renamed as a temporary file during processing. If the file is rejected, this file is not used.
- **Done File**: The same file as the input file, but renamed as a done file after processing has been successfully completed.
- **Error File**: The same file as the input file, but renamed as an error file if the file is rejected.

These files are managed by the EXT_InFileManager module. See "About Getting Pipeline Input from Files".

Creating a Stream Format Description File

To create a stream format description file, first identify the data in your external files that you need to use for rating. You can then either start with one of the sample files or create your own.

In the following example, the external file is a CDR. It includes three records: a HEADER, a TRAILER, and a DETAIL. Each record starts with a character that identifies its EDR container content type (**H** for HEADER, **T** for TRAILER, and **D** for DETAIL), and each record has a fixed structure.

- The example HEADER record in Table 3-1 has two fields.

Table 3-1 Header Record in Stream Format Description

Field	Length	Description
IDENTIFIER	1	The character **H**.
CREATION_TIME	14	The creation time of the CDR stream in the format *YYYYMMDDHHMMSS*.

- The example DETAIL record in Table 3-2 has five fields.

Table 3-2 Detail Record in Stream Format Description

Field	Length	Description
IDENTIFIER	1	The character **D**.
CALLING_PARTY	15	The A number.
CALLED_PARTY	15	The B number.
START_TIMESTAMP	14	The start time of the call in format YYYYMMDDHHMMSS.
DURATION	9	The duration of the call in seconds.

- The example TRAILER record in Table 3-3 has two fields.

Table 3-3 Trailer Record in Stream Format Description

Field	Length	Description
IDENTIFIER	1	The character **T**.
NUMBER_OF_DETAILS	9	The total number of DETAIL records in the stream.

A sample CDR input stream might be the following:

```
H20010613123410D4943311217 4957641506 200106131001120000000045D494106136432 49401531224
200106131002150000000056T000000002
```

The **INP_GenericStream** input module uses the stream format description file to break the CDR input stream into the following records:

```
H20010613123410

D4943311217 4957641506 2001061310011200000045

D494106136432 49401531224 2001061310021500000056

T000000002
```

The input module uses regular expressions to find each record. Table 3-4 lists the three record types and their regular expressions.

Table 3-4 Regular Expressions for the Records

Record	Regular expression	Description
HEADER	"H.{14}"	H followed by 14 arbitrary characters.
DETAIL	"D.{53}"	D followed by 53 arbitrary characters.
TRAILER	"T.{9}"	T followed by 9 arbitrary characters.

The description of the record must contain the following:

- The regular expression used by the input module to recognize the physical record
- The position and types of the fields inside the physical record

A sample format stream description for this example looks like this:

```
SampleFormat
{
  Header(FIX)
  {
    Info
    {
      Pattern = "H.{14}";
    }
    IDENTIFIER        AscString(1);
    CREATION_TIME     AscDate("YYYYmmddHHMMSS");
  }

  Detail(FIX)
  {
    Info
    {
      Pattern = "D.{53}";
    }

    IDENTIFIER        AscString(1);
    CALLING_PARTY     AscString(15);
    CALLED_PARTY      AscString(15);
    START_TIMESTAMP   AscDate("YYYYmmddHHMMSS");
    DURATION          AscInteger(9);
  }

  Trailer(FIX)
  {
    Info
    {
    Pattern = "T.{9}";
  }

    IDENTIFIER        AscString(1);
    NUMBER_OF_DETAILS AscInteger(9);
  }
}
```

Each field in the record is defined by the field type and value. For example, the fields in the DETAIL record are defined as follows:

```
    IDENTIFIER        AscString(1);
    CALLING_PARTY     AscString(15);
    CALLED_PARTY      AscString(15);
    START_TIMESTAMP   AscDate("YYYYMMDDHHMMSS");
    DURATION          AscInteger(9);
```

Record Types

The **INP_GenericStream** input module uses regular expressions to recognize the data records in the input stream. Different types of data records define fields in different ways:

- Fields can be defined by fixed-widths.
- Fields can be separated by special characters.
- The length of the field can be included in the input data (as in ASN.1 input).

Therefore, each data record has a record type that tells the input module how to split up the record into fields. Each data record has an **Info** block in its definition that contains some setup parameters for the record, for example, the field separator for a separated record. The record type determines which parameters can be used.

For example, this DETAIL record uses the record type SEPARATED:

```
DETAIL(SEPARATED)
{
Info
{
Pattern = ".*\n";
FieldSeparator = ';';
RecordSeparator = '\n';
}
SERVICE     AscString();
A_NUMBER    AscString();
B_NUMBER    AscString();
...
```

Record Type SEPARATED

The record type SEPARATED is used for records in which fields are separated by a special field delimiter character. The record itself can be terminated by another character (for example, the end-of-line symbol **\n**). Because there is no length information for the record, the regular expression specified as a pattern must match the full record, including the record separator.

There are no restrictions for the data types that can be used inside the SEPARATED record, although it makes no sense to use binary data types inside this record. The length information calculated from the position of the field delimiters overwrites length information specified in the data types. See Table 3-5.

Table 3-5 Parameters in SEPARATED

Parameter	Value	Description	Mandatory
Pattern	String	Regular expression that defines the entire record. This includes all records and the record separator character.	Yes
FieldSeparator	Character	Character that delimits single fields. Default = Comma (,)	No
RecordSeparator	Character	Character that delimits records. Default = No record separator	No

Record Type FIX

This record type is used for records with predefined width for each field. The record must contain all the fields. The record length is calculated as a sum of the widths of individual fields. Only data types with width information can be used. See Table 3-6.

Table 3-6 Parameters in FIX

Parameter	Value	Description	Mandatory
Pattern	String	Regular expression that identifies the record.	Yes

Record Type ASN

This record type is used for file formats defined in ASN.1, for example, TAP. You can use only the TAP and ASN data types in this record type. See Table 3-7.

Table 3-7 Parameters in ASN

Parameter	Value	Mandatory
Application	Integer	No
Context	Integer	No
Private	Integer	No
Universal	Integer	No

Syntax of the Stream Format Description File

The stream format description file is a simple ASCII file. The following grammar defines the syntax of the stream format description file:

```
<format-description-file> ::=  (<use_directive> | <stream-format>)*
<boolean> ::=                  "true" | "false"
<character> ::=                "'" "single character" "'"
<decimal> ::=                  "0..9"* "." "0..9"+
<extension-field-type> ::=     "any field type defined in a user extension"
<extension-record-type> ::=    "any record type defined in a user extension"
<field-name> ::=               <identifier>
<field-type> ::=               "AscString" | "AscDecimal" | "AscLong" | "AscDate" | ... |
                               <extension-field-type>
<format-name> ::=              <identifier>
<identifier> ::=               "a..z,A..Z,_" "a..z,A..Z,0..9,_"*
<info-block> ::=               "Info" "{" <info-parameter>* "}"
<info-parameter> ::=           <identifier> "=" <value> ";"
<integer> ::=                  "0..9"+
<record-definition> ::=        <record-name> "(" <record-type> ")" "{" <info-block>
                               <record-field>* "}"
<record-field> ::=             <field-name> <field-type> "(" [<field-parameter>
                               [, <field-parameter>]*] ")" ";"
<record-name> ::=              <identifier>
<record-type> ::=              "FIX" | "SEPARATED" | "ASN" | <extension-record-type>
<stream-format> ::=            <format-name> "{" <record-definition>* "}"
<string> ::=                   "\"" "any character"* "\""
<use_directive> ::=            "use" <identifier> ";"
<value> ::=                    <decimal> | <integer> | <identifier> | <string> |
                               <character> | <boolean>
```

Supported Data Types for the Stream Format Description File

Each entry in the stream format description file assigns a data type to a field. For example, this line assigns the **AscString** data type to the A number:

```
CALLING_PARTY     AscString(15);
```

Pipeline rating supports the following categories of data types:

- ASCII Data Types
- ASN.1 Data Types
- TAP Data Types

ASCII Data Types

Pipeline Manager supports the following ASCII data types:

- AscDate
- AscDecimal
- AscInteger
- AscString
- AscRawString

AscDate

Use the **AscDate** data type to read and write date/time information as an ASCII string. The **AscDate** data type can be used without any parameters or with a string specifying the used date format.

```
AscDate( [String format] )
```

The format string uses the following patterns:

- **%Y**: The year including the century (1901 ... 2037)
- **%y**: The year without the century (00 ... 99)
- **%m**: Month number (01 ... 12)
- **%d**: Day of the month (01 ... 31)
- **%H**: Hour (00 ... 23)
- **%M**: Minute (00 ... 59)
- **%S**: Seconds (00 ... 59)

When no format string is defined, the following default format is used:

```
%Y%m%d%H%M%S
```

AscDecimal

Use **AscDecimal** to read and write decimal values to and from ASCII streams.

```
AscDecimal( [Integer len [, Bool withPoint [, Integer precision [, Char pointChar [, Identifier
rounding[, Char padChar]]]]]] )
```

- **len**: The total length of the decimal value (default is 0 => unspecified).
- **withPoint**: Boolean flag to specify if there is a decimal point in the string (default is true).
- **precision**: Number of digits after the decimal point (default is 6).
- **pointChar**: Character used as decimal point (default is the point '.').
- **rounding**: Rounding method to use (PLAIN, UP, DOWN, BANK) (default is DOWN).
- **padChar**: Padding character to use (default is '0').

AscInteger

Use **AscInteger** to read and write integer values to and from ASCII streams.

Integer values are supported in the range from -2147483648 to 2147483647.

> **Note**
>
> AscInteger cannot be NULL (empty).

```
AscInteger( [Integer len [, Char padChar]] )
```

- **len**: The total length of the integer value (default is 0 => unspecified)
- **padChar**: The character used to pad integer values to a fixed length (default is the '0')

AscString

Use the AscString data type to read and write strings to and from an ASCII stream.

```
AscString( [Integer len [, Char padChar [, Bool isLeftJustified]]] )
```

- **len**: The total length of the string (default is 0 => unspecified).
- **padChar**: The character used to pad string values to a fixed length (default is a space character).
- **isLeftJustified**: Flag indicating that the string is left justified (default is **true**).

AscRawString

Equivalent to **AscString**, but preserves leading and trailing spaces while **AscString** strips all spaces from strings.

ASN.1 Data Types

Pipeline Manager supports the following ASN.1 data types:

- ASN_Integer
- ASN_LegacyOctetString
- ASN_OctetString
- ASN_RawOctetString
- ASN_BcdString
- ASN_NumberString
- ASN_HexString
- ASN_Tag
- ASN_Blob

ASN_Integer

Use **ASN_Integer** to read and write integer values from and to ASN.1 streams. Integer values are supported in the range from -2147483648 to 2147483647.

> **Note**
>
> ASN_Integer cannot be null (empty).

```
ASN_Integer( Integer TagValue [, String Asn1Class] )
```

- **TagValue**: The value to use as ASN.1 Tag.
- **Asn1Class**: The Class of the ASN.1 object. Values are:
  - **Application**
  - **Context**
  - **Private**
  - **Universal**

  The default is **Application**.

ASN_LegacyOctetString

Use ASN_LegacyOctetString to read and write an Octet string, which is a Byte string without any specific encoding for the data, for example, ascii or hex, from and to ASN.1 streams. ASN_LegacyOctetString removes the leading and trailing spaces after decoding an octet string.

This data type is similar to the ASN_OctetString data type except for the following difference:

- ASN_LegacyOctetString encodes an empty octet string with length = 0 and no value.
- ASN_OctetString builds an empty octet string with length = 1 and a space for the value. See "ASN_OctetString".

```
ASN_LegacyOctetString( Integer TagValue [, String Asn1Class] )
```

- **TagValue**: The value to use as ASN.1 Tag.
- **Asn1Class**: The Class of the ASN.1 object. Values are:
  - **Application** (Default)
  - **Context**
  - **Private**
  - **Universal**

ASN_OctetString

Use ASN_OctetString to read and write strings from and to ASN.1 streams. An Octet string is a Byte string without any specific encoding for the data, for example, ascii or hex.

```
ASN_OctetString( Integer TagValue [, String Asn1Class] )
```

- **TagValue**: The value to use as ASN.1 Tag.
- **Asn1Class**: The Class of the ASN.1 object. Values are:
  - **Application** (Default)
  - **Context**
  - **Private**
  - **Universal**

ASN_RawOctetString

Use ASN_RawOctetString to read and write an Octet string, which is a Byte string without any specific encoding for the data, for example, ascii or hex, from and to ASN.1 streams. Unlike the ASN_LegacyOctetString, ASN_RawOctetString does *not* remove the leading and trailing spaces after decoding an octet string.

See also "ASN_LegacyOctetString".

```
ASN_RawOctetString( Integer TagValue [, String Asn1Class] )
```

- **TagValue**: The value to use as ASN.1 Tag.
- **Asn1Class**: The Class of the ASN.1 object. Values are:
  - **Application** (Default)
  - **Context**
  - **Private**
  - **Universal**

ASN_BcdString

ASN_BcdString is an extension of the **ASN_OctetString** used to read and write strings containing data coded in the Binary Coded Decimal form to and from ASN.1 streams. This type automatically decodes and encodes BCD, so the user accesses the data seamlessly.

```
ASN_BcdString( Integer TagValue [, String Asn1Class] )
```

- **TagValue**: The value to use as ASN.1 Tag
- **Asn1Class**: The Class of the ASN.1 object. Values are:
  - **Application** (Default)
  - **Context**
  - **Private**
  - **Universal**

ASN_NumberString

ASN_NumberString is an extension of the **ASN_OctetString** used to read and write strings containing only numbers (and spaces) but packed in an ascii string from ASN.1 streams. An **ASN_NumberString** can be read and written as a string, date, or long.

```
ASN_NumberString( Integer TagValue [, String Asn1Class] )
```

- **TagValue**: The value to use as ASN.1 Tag.
- **Asn1Class**: The Class of the ASN.1 object. Values are:
  - **Application (Default)**
  - **Context**
  - **Private**
  - **Universal**

ASN_HexString

ASN_HexString is an extension of the **ASN_OctetString** used to read and write strings containing data coded in the Hexadecimal form, but stored as ASCII strings, from and to ASN.1 streams. This type is used because iScript cannot directly manipulate hexadecimal byte strings, so the strings are stored as ASCII representation of hexadecimal strings. For example, 0x28F3 is stored as 28F3.

ASN_HexString supports cases in which a special conversion method of read or write access is necessary.

```
ASN_HexString( Integer TagValue [, String Asn1Class] )
```

- **TagValue**: The value to use as ASN.1 Tag.
- **Asn1Class**: The Class of the ASN.1 object. Values are:
  - **Application** (Default)
  - **Context**
  - **Private**
  - **Universal**

ASN_Tag

Use **ASN_Tag** to read and write constructed ASN.1 objects to and from ASN.1 streams. Only the Parser should create this type of objects, out of record definitions in the block description file.

The **ASN_Tag** object can read both definite and indefinite length ASN.1 objects.

ASN_Blob

ASN_Blob is a special type used to store a complete structured (constructed) ASN.1 Object in the form of a byte string. This is useful when you need to transmit a block of data from the input to the output without processing, thus not needing to map the data into EDR container fields.

The only limitation for this type is that the ASN.1 object must have a definite length.

```
ASN_Blob( Integer TagValue [, String Asn1Class [, String Asn1Form]] )
```

- **TagValue**: The value to use as ASN.1 Tag
- **Asn1Class**: The Class of the ASN.1 object. Values are:
  - **Application** (Default)
  - **Context**
  - **Private**
  - **Universal**
- **Asn1Form**: The Form of the ASN.1 object. Values are:
  - Constructed (Default)
  - Primitive

TAP Data Types

Pipeline Manager supports the following TAP data types. These are defined only to match the type name used in the TAP format description file.

- **TAP_AsciiString.** Same type as a standard **ASN_OctetString**.
- **TAP_Description.** Same type as a standard **ASN_OctetString**.
- **TAP_Currency.** Same type as a standard **ASN_OctetString**.
- **TAP_PercentageRate.** Same type as a standard **ASN_Integer**.

Setting Up an Input Mapping File

To create an input mapping file, first identify the data in your external files that you need to map from the external files to the EDR container. You can then either start with one of the sample input mapping files or create your own.

Each mapping entry contains a list of mappings either from data record fields to EDR container fields or from constant values to EDR container fields. You can map a data record field to more than one EDR container field by adding more than one mapping. For example:

```
A_NUMBER    -> DETAIL.A_NUMBER;
.
.
.
A_NUMBER    -> DETAIL.ASS_GSMW_EXT.A_NUMBER_USER;
```

The following grammar defines the syntax of the input mapping file:

```
<input-mapping-file> ::=    <file-format-mapping>*
<constant> ::=              <integer> | <decimal> | <string>
<constant-mapping> ::=      <constant> "->" <edr-field>
<decimal> ::=               "0..9"* "." "0..9"+
<edr-field> ::=             <identifier> ("." <identifier>)+
<field-mapping> ::=         <field-name> "->" <edr-field>
<field-name> ::=            <identifier>
<file-format-mapping> ::=   <file-format> "{" <record-mapping>* "}"
<identifier> ::=            "a..z,A..Z,_" "a..z,A..Z,0..9,_"*
<integer> ::=               "0..9"+
<mapping-entry> ::=         <field-mapping> | <constant-mapping>
<mappings> ::=              <mapping-name> "{" <mapping-entry>* "}"
<record-mapping> ::=        <record-name> "{" <mappings>* "}"
<string> ::=                "\"" "any character"* "\""
```

Setting Up an Input Grammar File

The input grammar contains iScript statements to create an EDR container.

The syntax of the input grammar file is similar to the syntax used in Yacc grammars. This file defines the grammar of the input data that are parsed and of the iScript statements that are run when a certain symbol is found in the input data stream.

Configuring the Input DataDescription Registry Section

You configure a **DataDescription** section in the registry for each pipeline. The **DataDescription** section includes the following entries:

- **StreamFormats**: Specifies the input stream format file.

  See "About the Order of Listing Stream Format Description Files".
- **InputMapping**: Specifies the input mapping description file.
- **OutputMapping**: Specifies the output mapping description file.

> **Note**
>
> You specify the input grammar description file in the **Input** section.

This sample shows the **DataDescription** section:

```
DataDescription
  {
   Standard
   {
   ModuleName = Standard
   Module
   {
   StreamFormats
   {
     Format1 = ./formatDesc/Formats/Flist/Flist_v01.dsc
   }
   InputMapping
   {
    Mapping1 = ./formatDesc/Formats/Flist/Flist_v01_InMap.dsc
   }
   OutputMapping
   {
    Mapping1 = ./formatDesc/Formats/Flist/Flist_v01_OutMap.dsc
   }
   }
   }
}
```

> **Note**
>
> The DataDescription section also includes an entry for the output mapping file. See "Configuring EDR Output Processing".

About the Order of Listing Stream Format Description Files

Pipeline module instances prioritize the order in which formats are considered for parsing in the order that the stream format description files are listed in the **StreamFormats** section of the registry.

Parsing errors can occur in a pipeline if the stream format description files are listed in the incorrect order. For example, the stream format description file that applies to your custom input stream would be listed in the **StreamFormats** section *before* the output stream format description file.

Configuring the Input Section in the Registry

> **Note**
>
> When you configure the Input section, you configure the Pipeline Input Controller. See "Input Controller" in *BRM Pipeline Manager Reference*.

To configure the **Input** section in the registry, do the following:

- The **UnitsPerTransaction** entry: Use this entry to improve performance.
- The **INP_GenericStream** module: Specify the input grammar file in this section. See "INP_GenericStream" in *BRM Pipeline Manager Reference*.

  When configuring the INP_GenericStream module, configure one of the following modules as a submodule of the INP_GenericStream module:

  - **EXT_InFileManager**: Configure this module if the pipeline receives input from files. This module manages the input, temporary, and done files. See "EXT_InFileManager" in *BRM Pipeline Manager Reference*.
  - **EXT_InEasyDB**: Configure this module if the pipeline receives input from a database. See "EXT_InEasyDB" in *BRM Pipeline Manager Reference*.
  - **EXT_InSocketMgrFlist**: Configure this module for input from flist-based networks if the pipeline receives input from prepaid networks.

About Getting Pipeline Input from Files

To configure a pipeline to read data from files, use the EXT_InFileManager module.

When you configure the module, you specify the directories, suffixes, and prefixes for the following files:

- **Input files**: CDR files from the mediation system. The prefix and suffix are used by the input module to identify which files to process.

  The input module checks periodically for files in this folder with the specified prefix and/or suffix.
- **Done files**: Created when a transaction is successfully completed.
- **Error files**: Created after a transaction rollback.

To manage file names, you can specify the following:

- The prefix for temporary files. Temporary files are used as input until the transaction is complete.
- Whether to replace or append prefixes and suffixes.
- The time period (in seconds) for which the input directory must be empty before the EVT_INPUT_DIR_EMPTY event is sent.

See "EXT_InFileManager" in *BRM Pipeline Manager Reference*.

About Getting Pipeline Input from a Database

To get pipeline input from a database, use the EXT_InEasyDB module.

To set up a pipeline for database input:

1. Create a job file that consists of SQL statements. It can also include iRule variables, which are defined in a parameter file. The EXT_InEasyDB module uses the commands to create EDRs.
2. Place the job file in a specific directory. When Pipeline Manager starts, the EXT_InEasyDB module finds the directory from the registry and starts the input process according to the commands in the job file.

You can stop and restart the pipeline after a system crash by configuring a restart file.

You can start the pipeline by using the **ReadDatabase** semaphore. When the module receives a start command while in process, the new SQL command is written to a job file.

The returned values of the database input stream contain all fields of each selected row divided by the configurable delimiter.

To configure the pipeline to read data from a database, see "EXT_InEasyDB" in *BRM Pipeline Manager Reference*.

Specifying the Maximum Errors Allowed in an Input File

You can configure the Output Controller to reject an entire input stream that exceeds a maximum percentage of errors. For example, you can specify to reject an input stream if over 20% of the EDRs have a particular error.

You specify the error threshold by using the **MaxErrorRates** entry in the Output section of the registry file.

> **Note**
>
> You can also configure a pipeline to reject individual EDRs by using the FCT_Reject module. For information, see "About Standard Recycling" in *BRM Suspending and Recycling Pipeline EDRs* and "Recycling EDRs in Pipeline-Only Systems" in *BRM Suspending and Recycling Pipeline EDRs*.

When an input stream exceeds the error threshold, the Output Controller:

- Deletes all output streams associated with the input stream. For example, if a pipeline splits an input stream into five output streams, the Output Controller deletes all five output streams.
- Moves the input stream to the error directory. You define the location of the error directory by using the ErrorPath registry entry. For information, see "EXT_InFileManager" in *BRM Pipeline Manager Reference*.

To set an error threshold:

1. Stop Pipeline Manager, if necessary.
2. Open your registry file in a text editor.
3. Edit the Pipeline Output Controller's **MaxErrorRates** registry entries, making sure you:
   - List all error codes that the Pipeline Output Controller should monitor.
   - Specify an error threshold for each error code. The threshold specifies the maximum percentage of EDRs that are allowed to have the particular error.

   For example, to configure the Output Controller to reject a stream if any of the following are true:
   - Over 10% of the EDRs have an INF_EDR_REJECTED error.
   - Over 8% of the EDRs have an ERR_CUST_NOT_FOUND error.
   - Over 20% of the EDRs have an ERR_CHARGED_ZONE_NOT_FOUND error.

```
Output
{
    ...
    MaxErrorRates
```

```
        {
            INF_EDR_REJECTED = 10
            ERR_CUST_NOT_FOUND = 8
            ERR_CHARGED_ZONE_NOT_FOUND = 20
        }
        ...
    }
```

4. Save and close the registry file.
5. Restart Pipeline Manager.

Reading TAP Files

Pipeline Manager can read the following TAP versions:

- TAP-0301 from the TD57v3.04.00 specification
- TAP-0303 from the TD57v3.07.01 specification
- TAP-0304 from the TD57v3.08.02 specification
- TAP-0309 from the TD57v3.90 specification
- TAP-0310 from the TD57v3.10.01 specification
- TAP-0311 from the TD57v28 specification
- TAP-0312 from the TD57v30.1 specification
- TAP-0312 from the TD57v32.1 specification

You can specify TAP input grammar files when you set up your input modules. The files are located in the *Pipeline_home***/formatDesc/Formats/TAP3** directory. *Pipeline_home* is the directory where you installed Pipeline Manager.

> **Note**
>
> Because the TAP 3.10 standard introduced fundamental changes, files produced according to earlier versions of the TAP standard are not compliant with the TAP 3.10 standard. Therefore, the TAP 3.10 grammar files cannot be used to process previous TAP versions.

Note the following implementation details:

- The following records are generated when TAP is processed by Pipeline Manager:
  - 1 Header record (010): 1 EDR
  - 1 Trailer record (090): 1 EDR
  - 1 GPRS record (040 for SGSN, 042 for GGSN or mixed ticket): 1 EDR
  - 1 mobile supplementary service (MSS) record (029): 1 EDR
  - 1 service center usage (SCU) record (050): 1 EDR
  - 1 value added service (VAS) record (060): 1 EDR
  - 1 content transaction (CONT) record (999): 1 EDR
  - 1 location service (LOCN) record (998): 1 EDR

  - 1 mobile originating call (MOC) record (021): *n* EDRs: one EDR for every basic service used.
  - 1 mobile terminating call (MTC) record (031): *n* EDRs: one EDR for every basic service used.

- For each supplementary service, an SS_PACKET is created and attached to the corresponding EDR. For every charge detail of the VAS array, a charge packet is added to the latest generated EDR. The VAS short description is stored in the DETAIL.ASS_CBD.CP.PRODUCTCODE_USED field.
- The Value added service used block is used to build an associated charge breakdown record containing data for rating.
- The CAMEL service information is stored in the ASSOCIATED_CAMEL_EXTENSION ("700") block of the EDR. The associated charge packets are stored on the same ASS_CBD as others but with the PRODUCTCODE_USED field set to CAMEL to identify them.
- Mobile originating and terminating records (MOC and MTC records) are split into multiple records downstream by the ISC_TapSplitting iScript. This iScript generates one EDR for every basic service in the basic service used array. For more information, see "ISC_TapSplitting" in *BRM Pipeline Manager Reference*.

Note the following restrictions:

- The size of the ASN.1 string is not checked during input.
- Pipeline Manager does not add the MIN to the EDR.
- Pipeline Manager does not add the electronic serial number (ESN) to the EDR.

The TAP output grammar recognizes all record types generated by Pipeline Manager.

About Customizing Mapping of Flist Fields to Rating EDR Container Fields

To process events received from the Connection Manager (CM), a real-time pipeline converts the event data from flist format to rating EDR format for processing by Pipeline Manager.

BRM provides default flist-to-rating-EDR mappings for GSM, GPRS, and SMS events in the *Pipeline_home***/formatDesc/Formats/Realtime/rate_event.xml** file. When the real-time pipeline starts, the INP_Realtime module uses the descriptions in this XML file to construct an in-memory representation of the flist-to-rating-EDR mapping. The pipeline uses the in-memory mapping to convert incoming flists to rating EDR format.

Table 3-8 displays the XML elements are used in the flist-to-rating-EDR mapping.

Table 3-8 XML Elements in flist-to-rating-EDR Mappings

XML element	Description
OpcodeMap	The root node of the XML document. The XML document can have only one OpcodeMap element. This element requires the *opcode* attribute, which is a unique string used to differentiate between OpcodeMap elements in other XML files.
InMap	The flist-to-rating-EDR mappings for the input flist. The XML document can have only one InMap element. The *containerType* attribute specifies the root-level name of the rating EDR container.

Table 3-8 (Cont.) XML Elements in flist-to-rating-EDR Mappings

XML element	Description
FlistField	A field in the input flist. If the *target* attribute is present, the flist field is mapped to the target field in the rating EDR container.
EdrBlock	Indicates when a rating EDR Block should be created.

Sample input flist:

```
0 PIN_FLD_POID           POID [0] 0.0.0.1 /account 12035 15
0 PIN_FLD_EVENT        SUBSTRUCT [0] allocated 52, used 52
1     PIN_FLD_POID           POID [0] 0.0.0.1 /event/delayed/session/telco/gsm
1373193266068995136 0
1     PIN_FLD_ACCOUNT_OBJ    POID [0] 0.0.0.1 /account 12035 0
1     PIN_FLD_START_T      TSTAMP [0] (1081882800) Tue Apr 13 12:00:00 2004
1     PIN_FLD_END_T        TSTAMP [0] (1081883200) Tue Apr 13 12:00:00 2004
1     PIN_FLD_QUANTITY    DECIMAL [0] 400
1     PIN_FLD_TELCO_INFO   SUBSTRUCT [0] allocated 20, used 12
2         PIN_FLD_CALLING_FROM    STR [0] "0049100050"
2         PIN_FLD_CALLED_TO       STR [0] "0049100051"
2         PIN_FLD_USAGE_CLASS     STR [0] "NORM"
2         PIN_FLD_TERMINATE_CAUSE   ENUM [0] 0
1     PIN_FLD_GSM_INFO     SUBSTRUCT [0] allocated 20, used 14
2         PIN_FLD_CALLED_NUM_MODIF_MARK   ENUM [0] 0
2         PIN_FLD_DIRECTION          ENUM [0] 0
2         PIN_FLD_CELL_ID            STR [0] "123456"
2         PIN_FLD_SUB_TRANS_ID       STR [0] "S"
2         PIN_FLD_DESTINATION_SID    STR [0] ""
2         PIN_FLD_NUMBER_OF_UNITS    DEC[0] 1.0
```

The default flist-to-rating-EDR mapping in XML for the above flist:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<OpcodeMap xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="./OpcodeMapping.xsd"
opcode="PCM_OP_RATE_PIPELINE_EVENT">
   <InMap containerType="DETAIL">
      <!--CONSTANT EDR DETAIL ITEMS-->
      <EdrField name="DETAIL.RECORD_TYPE" value="020" />
      <EdrField name="DETAIL.DISCARDING" value="0" />
      <!--EVENT FLIST ITEMS-->
      <FlistField name="PIN_FLD_EVENT">
         <!--EVENT POID-->
         <FlistField name="PIN_FLD_POID" format="type" target="DETAIL.EVENT_TYPE" alias="EventType"/>
         <!--ACCOUNT_OBJ-->
         <FlistField name="PIN_FLD_ACCOUNT_OBJ" format="short" target="DETAIL.CUST_A.ACCOUNT_PARENT_ID" />
         <!--CHARGING_START-->
         <FlistField name="PIN_FLD_START_T">
            <EdrField name="DETAIL.CHARGING_START_TIMESTAMP" />
            <EdrField name="DETAIL.NE_CHARGING_START_TIMESTAMP" />
         </FlistField>
         <!--CHARGING_END-->
         <FlistField name="PIN_FLD_END_T">
            <EdrField name="DETAIL.CHARGING_END_TIMESTAMP" />
            <EdrField name="DETAIL.NE_CHARGING_END_TIMESTAMP" />
         </FlistField>
         <!--QUANTITY FIELD-->
         <FlistField name="PIN_FLD_QUANTITY" target="DETAIL.DURATION" />
         <!--TELCO FLIST ITEMS-->
```

```
         <FlistField name="PIN_FLD_TELCO_INFO">
            <!--A_NUMBER-->
            <FlistField name="PIN_FLD_CALLING_FROM">
               <EDRField name="DETAIL.A_NUMBER"/>
               <EDRField name="DETAIL.ASS_GSMW_EXT.A_NUMBER_USER" onAliasName="EventType" onAliasValue="/
event/delayed/session/telco/gsm" />
            </FlistField>
            <!--B NUMBER-->
            <FlistField name="PIN_FLD_CALLED_TO">
               <EDRField name="DETAIL.B_NUMBER"/>
               <EDRField name="DETAIL.ASS_GSMW_EXT.DIALED_DIGITS" onAliasName="EventType" onAliasValue="/
event/delayed/session/telco/gsm" />
            </FlistField>
            <!--USAGE_CLASS-->
            <FlistField name="PIN_FLD_USAGE_CLASS" target="DETAIL.USAGE_CLASS" />
            <!--CALL_COMPLETION_INDICATOR-->
            <FlistField name="PIN_FLD_TERMINATE_CAUSE" target="DETAIL.CALL_COMPLETION_INDICATOR" />
         </FlistField> <!--END TELCO FLIST ITEMS-->
         <!--GSM FLIST ITEMS-->
         <FlistField name="PIN_FLD_GSM_INFO" onAliasName="EventType" onAliasValue="/event/delayed/
session/telco/gsm" optional="true">
            <EdrBlock name="DETAIL.ASS_GSMW_EXT"/>
            <!--CONSTANT DETAIL.ASS_GSMW_EXT EDR ITEMS-->
            <EdrField name="DETAIL.ASS_GSMW_EXT.RECORD_TYPE" value="520" />
            <FlistField name="PIN_FLD_CALLED_NUM_MODIF_MARK" target="DETAIL.B_M ODIFICATION_INDICATOR" />
            <FlistField name="PIN_FLD_DIRECTION" target="DETAIL.USAGE_DIRECTION" />
            <FlistField name="PIN_FLD_CELL_ID" target="DETAIL.ASS_GSMW_EXT.CELL_ID"/>
            <FlistField name="PIN_FLD_SUB_TRANS_ID" target="DETAIL.LONG_DURATION_INDICATOR" />
            <FlistField name="PIN_FLD_NUMBER_OF_UNITS" target="DETAIL.NUMBER_OF_UNITS" />
         </FlistField> <!--END GSM FLIST ITEMS-->
         <!--END EVENT FLIST ITEMS-->
      </FlistField>
   </InMap>
</OpcodeMap>
```

You can customize the default GSM, GPRS, and SMS mappings and create custom mappings for other types of events. To customize flist-to-rating-EDR mappings, you must be familiar with the following topics:

- BRM flists.
- XML
- XML Schema

To create and use a custom mapping:

1. Edit the pipeline event XML file (*Pipeline_home*/**formatDesc/Formats/Realtime/rate_event.xml)** to add your custom mappings.
2. Validate the XML file using *Pipeline_home*/**formatDesc/Formats/Realtime/opcode_ifw_mapping.xml** file.

> **Note**
>
> Using an invalid XML file prevents the INP_Realtime module from successfully starting. Make sure you validate the XML file against the XML schema.

About the POID Format in the Rating EDR Container

If an flist field is a POID, the *format* attribute is required to indicate the format of the POID in the rating EDR container. The following format types are supported: long, short, id, and type.

For example, the format of POID 0.0.0.1 /account 12065 is mapped as follows:

- **1_12065 /account** when the format attribute is *long*
- **1_12065** when the format attribute is *short*
- **12065** when the format attribute is *id*
- **/account** when the format attribute is *type*

Mapping an Flist Field to Multiple Rating EDR Container Fields

If the FlistField element contains child **EDRField** elements, the flist field is mapped to multiple rating EDR fields.

In the following example, PIN_FLD_END_T is mapped to multiple fields in the DETAIL EDR block:

```
<FlistField name="PIN_FLD_END_T">
<EdrField name="DETAIL.CHARGING_END_TIMESTAMP" />
<EdrField name="DETAIL.NE_CHARGING_END_TIMESTAMP" />
</FlistField>
```

Using Conditions to Map an Flist Field to a Rating EDR Container Field

You can use the FlistField attributes *alias*, *onAliasName*, and *onAliasValue* for conditional mappings.

In this example, the input flist is a **/event/delayed/session/telco/gsm** event. The PIN_FLD_GSM_INFO substruct of the event is mapped to the DETAIL.ASS_GSMW EDR block.

```
<FlistField name="PIN_FLD_POID" format="type" target="DETAIL.EVENT_TYPE" alias="EventType"/>
<!--GSM FLIST ITEMS-->
<FlistField name="PIN_FLD_GSM_INFO" onAliasName="EventType" onAliasValue="/event/delayed/session/telco/
gsm" optional="true"> <EdrBlock name="DETAIL.ASS_GSMW_EXT"/>
<!--CONSTANT DETAIL.ASS_GSMW_EXT EDR ITEMS-->
<EdrField name="DETAIL.ASS_GSMW_EXT.RECORD_TYPE" value="520" />
<EdrField name="DETAIL.ASS_GSMW_EXT.TIME_BEFORE_ANSWER" value="0" />
<EdrField name="DETAIL.ASS_GSMW_EXT.NUMBER_OF_SS_PACKETS" value="0" />
<FlistField name="PIN_FLD_CALLED_NUM_MODIF_MARK" target="DETAIL.B_MODIFICATION_INDICATOR" />
<FlistField name="PIN_FLD_DIRECTION" target="DETAIL.USAGE_DIRECTION" />
<FlistField name="PIN_FLD_CELL_ID" target="DETAIL.ASS_GSMW_EXT.CELL_ID" />
<FlistField name="PIN_FLD_SUB_TRANS_ID" target="DETAIL.LONG_DURATION_INDICATOR" />
<FlistField name="PIN_FLD_NUMBER_OF_UNITS" target="DETAIL.NUMBER_OF_UNITS" />
</FlistField> <!--END GSM FLIST ITEMS-->
```

You can also map an alias to an **EdrField** defined elsewhere. For example:

```
<FlistField name="PIN_FLD_START_T" target="DETAIL.CHARGING_START" alias="startTime"/>
<!--GSM FLIST ITEMS-->
<FlistField name="PIN_FLD_GSM_INFO" >
<EdrBlock name="DETAIL.ASS_GSMW_EXT"/>
<!--CONSTANT DETAIL.ASS_GSMW_EXT EDR ITEMS-->
```

```
<EdrField name="DETAIL.ASS_GSMW_EXT.CHARGING_START" useAlias="startTime" />
</FlistField> <!--END GSM FLIST ITEMS-->
```

4

Configuring EDR Output Processing

Learn how to set up output processing for the Oracle Communications Billing and Revenue Management (BRM) Pipeline Manager. The input can be call detail records (CDRs) for telco rating.

Topics in this document:

About the Output Process

Pipeline Manager can generate data for various purposes:

- To provide rated events for the Rated Event (RE) Loader to load into the BRM database.
- To collect rejected EDRs for recycling.
- To collect duplicate EDRs.
- To send data to the Pipeline Manager database for additional processing in another pipeline.
- To handle discarded EDRs.

The output process works as follows:

1. The completed EDRs are moved to the output buffer.
2. The output module reads data from the output buffer and processes the data. The processing performed depends on the output configuration, for example:
    - If the output consists of rated EDRs, the output module uses grammar and description files to convert the EDRs to a format that the RE Loader uses.
    - If the output consists of rejected EDRs, the output module creates reject files.
3. The output module writes the data to the specified output stream. The destination of an output stream can be a file directory, a location in the database, or an external network. You can configure different directories, file names, and network destinations for each output stream.

About the Output Processing System Components

Output processing is managed by the output controller and the output collection module.

- The *output controller* manages the overall output process. You can configure output properties that apply to all streams. See "Output Controller" in *BRM Pipeline Manager Reference*.

  The Output controller performs the following functions:

  - Manages the output stream's trailer information.
  - Rejects input streams when they exceed a specified maximum number of errors.
  - Checks for duplicate CDR files.
  - Notifies the Transaction Manager when a transaction ends.
  - Stops the pipeline when critical errors occur.

  The input stream's trailer information can become invalid when a pipeline discards or rejects individual EDRs or when a pipeline splits EDRs into multiple output streams. For example, the input stream's trailer contains a total charge amount field, which contains the charge amount for all EDRs. When a pipeline discards some EDRs in the input stream, the total charge amount is no longer valid. To correct this, the Output Controller automatically recalculates the trailer information for each output stream.

- *Output collection* defines all the output streams for the pipeline. The Output Collection module performs the following functions:

  - Generates the output devices that are specified in the registry at system startup.
  - Passes the EDR container to the specified output device.

  You configure the Output Collection module by editing a pipeline's **OutputCollection** section of the registry file. For information, see "Output Controller" in *BRM Pipeline Manager Reference*.

About the Output Modules

The following output modules are available:

- **OUT_GenericStream**: Used to process rated events. This module converts the EDR format to a format needed for further processing, for example, RE Loader format to load rated events or TAP format for outcollect processing of roaming records.

  See "Configuring Output for Rated Events" and "OUT_GenericStream" in *BRM Pipeline Manager Reference*.

- **OUT_Reject**: Used to process rejected or duplicate events. See "Configuring Output for Rejected or Duplicate EDRs" and "OUT_Reject" in *BRM Pipeline Manager Reference*.
- **OUT_DB**: Used to send data to the database. See "Sending Output to a Database" and "OUT_DB".
- **OUT_DevNull**: Used to discard EDRs that you don't want to recycle. See "Configuring Output of Discarded EDRs" and "OUT_DevNull" in *BRM Pipeline Manager Reference*.
- **EXT_OutFileManager**: Used to manage the output files of the OUT_GenericStream module and the OUT_Reject modules. See "EXT_OutFileManager" in *BRM Pipeline Manager Reference*.

About Output Processing File Types

Output processing uses the following types of files:

- **Temporary data file**: Stores records during processing. There is one temporary file for each external file being processed. If the entire external file is rejected, the temporary file is not used.
- **Output File**: Stores the EDRs after processing. This file is derived from the temporary file. When processing is completed successfully, the temporary file is renamed and becomes the output file.
- **Temporary stream file list**: Stores the names of temporary data files. This is required when the Replace registry entry is enabled. In that case, the output file name is appended with a sequence number. Each output stream uses a separate temporary stream file list. For example, the telephony output stream registry includes these entries:

```
TempDataPath       = ./samples/wireless/data/telout
TempDataPrefix     = tel.tmp.
TempDataSuffix     = .data
```

> **Note**
>
> These registry entries are used for internal pipeline processing only and should not be changed.

These output processing files are managed by the "EXT_OutFileManager" module in *BRM Pipeline Manager Reference*.

About ASN.1 Output

ASN.1 objects are composed of a triplet TLV (Tag/LengthOfValue/Value). Therefore, before writing ASN.1 to the output, you must calculate the **LengthOfValue** field for every element of the ASN.1 tree that you are building. To do this, use the EXT_AsnTree module.

This extension has functions to perform the following tasks:

- Builds a tree of ASN.1 objects.
- Updates the **LengthOfValue** fields of the ASN.1 objects.
- Flushes the resulting ASN.1 data block to an output stream.

About Configuring Output Processing

To configure output processing, do the following:

1. Create the directories for the output streams.
2. For output of rated events:
   - Set up an output stream format description file.
   - Set up a mapping file.
   - Set up a grammar file.

   In most cases, you need to modify only the sample stream format description and output mapping files.
3. Configure these output sections in the registry:
   - The output mapping in the **DataDescription** section. This specifies the output mapping for rated events. See "Configuring the Output DataDescription Registry Section".

- The **OutputBuffer** section in the **Pipeline** section.
- The **Output** section. This section includes the configuration for the pipeline output controller and for output modules such as the OUT_GenericStream module. See "About the Output Modules".

4. Configure the output stream for the function modules that create the output. For example, configure the FCT_Reject module to send rejected EDRs to the reject output stream.

About Configuring the Output Section in the Registry

The output section in a pipeline has this hierarchy:

```
Pipeline
  Output
      Output controller
         Output collection
             Output streams
```

- The *output controller* manages the overall output process. You can configure output properties that apply to all streams. See "Output Controller" in *BRM Pipeline Manager Reference*.
- *Output collection* defines all the output streams for the pipeline. You configure the Output Collection module by editing the **OutputCollection** section of the registry file. For information, see "Output Collection" in *BRM Pipeline Manager Reference*.
- Output streams send EDRs to the appropriate output location. See "About the Output Modules".

About Configuring Statistics Information in the Output Section

> **Note**
>
> Instrumented statistics are designed for batch rating and not real-time rating. Statistics are related to transactions. Since real-time rating does not use transactions, instrumented statistics are unavailable for real-time rating.

The **Statistic** subgroup controls the statistics related to Pipeline Manager's EDR processing rate. You can view these statistics in the output logs or HTTP browser.

The **Statistic** subgroup has one registry option, **EDRCountCriteria**. This option can have two possible values:

- **INPUT**: The Pipeline Manager considers only the detailed CDRs that are passed through it as input for calculating the EDR statistics.
- **ALL**: This is the default option. The Pipeline Manager considers all the EDRs that are passed through it for calculating the EDR statistics. This includes the duplicate EDRs created and the EDRs directed to an additional output stream.

> **Note**
>
> The **Statistic** subgroup is optional. If this subgroup is not present, the behavior of the Pipeline Manager is similar to that when **EDRCountCriteria** is set to **ALL**.

This sample shows the output hierarchy:

```
Pipelines
{
    W_SAMPLE
    {
        Output
        {
            WriteDefaultEdr = False
            MaxErrorRates
        {
        }
          #The following subgroup is optional
            Statistic
            {
                    EdrCountCriteria = ALL
            }
            OutputCollection
            {
                Output stream 1
                {
                    ModuleName = OUT_Module_1
                    ...
                }
                Output stream 2
                {
                    ModuleName = OUT_Module_2
                    ...
                }
            } # end of Output
        } # END W_SAMPLE
    } # END Pipelines
```

Configuring Output for Rated Events

To output rated events from pipeline rating, you configure the OUT_GenericStream module. This module converts EDRs to the output format used by RE Loader. The output is a file that is loaded by RE Loader.

To convert data from EDR format to a file, the module uses the following files:

- **Stream format description**
- **Output grammar**: Specifies which records to include in the output.
- **Output mapping**: Specifies how to map the data in EDRs to data in the format defined by the output grammar file.

The sample registry files include stream format description, output grammar, and output mapping files that convert data from the BRM EDR, TAP, and CIBER formats.

To configure the OUT_GenericStream module, you specify the following:

- The output grammar file.
- The BRM event type that the output file contains, such as **/event/delayed/session/gsm**.
- The type of pipeline, for example, rating or backout.
- Whether to delete empty output streams.
- The output module, either "EXT_OutFileManager" in *BRM Pipeline Manager Reference*.

See "OUT_GenericStream" in *BRM Pipeline Manager Reference*.

Creating Separate Output Streams for Each Service

Events from different services (GSM, SMS, and so forth) must be delivered to the RE Loader or a prepaid network in separate files. To do this:

- Configure the IRL_EventTypeSplitting iScript to split EDRs by service code. See "Sending EDRs to Pipeline Output Streams".
- In the registry, configure an instance of the OUT_GenericStream module for each service.

Creating Multiple Output Streams in One Output Registry

Each EDR must have a default output stream and may also contain additional output streams. Use the following iScript functions to manipulate multiple output streams in a single registry:

- "edrAddAdditionalStream" in *BRM Pipeline Manager Reference*
- "edrRemoveAdditionalStream" in *BRM Pipeline Manager Reference*
- "edrGetAdditionalStream" in *BRM Pipeline Manager Reference*
- "edrContainsAdditionalStream" in *BRM Pipeline Manager Reference*

To add output streams to the default stream in the sample registry:

1. Create an iScript file that adds the stream.

   The **edrAddAdditionalStream** iScript document contains an example **addoutmod.isc** iScript file that shows how to add two additional output module streams.

2. Reference this iScript file in the pipeline FunctionPool registry section.

   The **edrAddAdditionalStream** iScript document contains an example function registry section that shows how to use the **addoutmod.isc** file to add an output stream in the function registry.

3. Define the stream in the pipeline output registry section.

   The **edrAddAdditionalStream** iScript document contains an example output registry section that shows how to configure additional output streams by using OUT_GenericStream.

See **edrRemoveAdditionalStream** for an example that shows how to remove an EDR output stream.

Configuring the Output DataDescription Registry Section

You configure a **DataDescription** section in the registry for each pipeline. The **DataDescription** section specifies the stream format description and input mapping files (see "About the Input Process") and the output mapping file.

```
DataDescription
  {
   Standard
   {
   ModuleName = Standard
   Module
   {
   StreamFormats
   {
     Format1 = ./formatDesc/Formats/Flist/Flist_v01.dsc
   }
```

```
    InputMapping
    {
     Mapping1 = ./formatDesc/Formats/Flist/Flist_v01_InMap.dsc
    }
    OutputMapping
    {
     Mapping1 = ./formatDesc/Formats/Flist/Flist_v01_OutMap.dsc
    }
    }
    }
}
```

About the Order of Listing Stream Format Description Files

Pipeline module instances prioritize the order in which formats are considered for parsing in the order that the stream format description files are listed in the **StreamFormats** section of the registry.

Parsing errors can occur in a pipeline if the stream format description files are listed in the incorrect order. For example, the stream format description file that applies to your custom input stream would be listed in the **StreamFormats** section *before* the output stream format description file.

Configuring Output for Rejected or Duplicate EDRs

The OUT_Reject module writes rejected EDRs to a reject file. A rejected EDR is identical to the EDR input.

To configure output for rejected EDRs, do the following:

- For rejected EDRs, configure the FCT_Reject module. See:
  - "Configuring Standard Recycling" in *BRM Suspending and Recycling Pipeline EDRs*
  - "Recycling EDRs in Pipeline-Only Systems" in *BRM Suspending and Recycling Pipeline EDRs*
- For duplicate EDRs, configure the FCT_DuplicateCheck module. See "Handling Duplicate EDRs".
- In the registry, configure an instance of the OUT_Reject module for rejected EDRs and an instance for duplicate EDRs. See "OUT_Reject" in *BRM Pipeline Manager Reference*.

  You can use the same output directory with different file suffixes and/or prefixes for rejected or duplicate EDRs.

The OUT_Reject module is a submodule of the Output Collection module. All registry parameters and error messages are handled by the Output Collection module. See "Output Collection" in *BRM Pipeline Manager Reference*.

File handling for rejected and duplicate EDRs is performed by the EXT_OutFileManager module. See "Sending Output to a File".

Sending Output to a File

To send output to a file, use the "EXT_OutFileManager" in *BRM Pipeline Manager Reference* module. The EXT_OutFileManager module handles file prefixes, suffixes, and paths for the OUT_GenericStream and OUT_Reject modules.

The EXT_OutFileManager module writes the output data to a temporary data file. When the TAM reports a successful completion of a transaction, the file is renamed to an output file and the temporary data file is removed.

If a transaction is rolled back, the original input is restored, and the temporary data file is moved to an error directory and renamed with an error prefix and/or suffix.

By default, the EXT_OutFileManager module is configured to delete empty output files. There is a short period of time during which empty output files are renamed and deleted. If another process, such as one that manages output files, tries to manipulate the file during this period, the pipeline may experience errors. To avoid this problem, configure any output file management processes to wait approximately one minute before attempting to manipulate files.

Configuring the Temporary File Name

Use the **TempPrefix** registry entry to specify the prefix for temporary data files.

> **Note**
>
> Do not change the **TempDataPrefix**, **TempDataSuffix**, and **TempDataPath** registry entries. These entries are used by the pipeline for internal data processing only.

See "EXT_OutFileManager" in *BRM Pipeline Manager Reference*.

Configuring File Prefixes and Suffixes

Use the **OutputPath**, **OutputPrefix**, and **OutputSuffix** registry entries to manage the output files for each stream.

> **Note**
>
> To ensure output file integrity, specify a unique combination of **OutputPath**, **OutputSuffix**, and **OutputPrefix** values for each output stream defined in the registry.

See "EXT_OutFileManager" in *BRM Pipeline Manager Reference*.

Creating an Output File Name from the Input File Name

Use the **UseInputStreamName** registry entry to specify to use the input file name to build the output file name.

For example, when **UseInputStreamName = [2,4;4,6;8,&]**, the following characters from the input file name are used to build the output file name:

- Characters 2 to 4.
- Characters 4 to 6.
- Characters 8 to end of string, where the '&' symbol indicates the end of the string.

For instance, if the name of the input file is **test12345678**, **Outputprefix** is **test**, and **Outputsuffix** is **.edr**, the output file name will be **testestt1245678.edr**

See "EXT_OutFileManager" in *BRM Pipeline Manager Reference*.

Applying a Prefix to the Sequence Number

Use the **SequencerPrefix** registry entry to specify a prefix to the sequence number before it gets appended to the generated output file name.

> **Note**
>
> This entry is used only when **AppendSequencerNumber** is set to **True**.

For example, when **SequencerPrefix** is "+", **Outputprefix** is test, **Outputsuffix** is **.edr**, the output file name will be **test+000002.edr**.

By default, the **SequencerPrefix** is "_" and if no **SequencerPrefix** is needed, it has to be specified as **SequencerPrefix** = "".

> **Note**
>
> Do not use the characters #, $, =, /, or \ to specify SequencerPrefix.

Using the Output of One Pipeline as the Input to Another Pipeline

If you want to use the output of one pipeline as the input for another pipeline, you must move the output file to another directory before it can be used by the next pipeline. You move the files by using "Event Handler" in *BRM Pipeline Manager Reference* and an external script. Event Handler would run your external script when prompted by a specified internal event. The external script would move the output file from one directory to another.

For example, if you want to use the output file from pipeline A as the input file to pipeline B:

- Configure pipeline A to generate output files in directory 1.
- Configure pipeline B to process input files from directory 2.
- Create a simple script (*ExternalScript*) that moves the output files from directory 1 to directory 2.
- Configure Event Handler to run *ExternalScript* whenever pipeline A generates an output file. For example, you can configure Event Handler to run *ExternalScript* when it receives an EVT_OUTPUT_FILE_READY event from the "EXT_OutFileManager" module in *BRM Pipeline Manager Reference*.

Sending Output to a Database

Use the OUT_DB module to send data to a database. You configure the following:

- A SQL command parameter file
- Files that define the following:
  - SqlBeginStream statement
  - HEADER records

  - DETAIL records
  - TRAILER records
  - SqlEndStream statement
- The OUT_DB module

The OUT_DB module reads the parameter file for each new output stream.

When you configure the OUT_DB module, you configure the following:

- The database to load data into.
- Path and file names for the configuration files.
- Aliases for the stream name and row number.
- Source and destination for the header record.
- File management options.

For more information about the OUT_DB module registry entries, see "OUT_DB" in *BRM Pipeline Manager Reference*.

About the OUT_DB Module Configuration Files

The parameter file contains key and value pairs. The keys are used by other configuration files (**SqlBeginStream**, **HeaderTableDefinition**, **DetailTableDefinition**, **TrailerTableDefinition**, **SqlEndStream**). The syntax to recognize keys is ${KEY}. When the other files are processed, all found keys are replaced by the corresponding values. The files are not deleted.

It is possible to change the database tables in a running system if there are no open streams.

The **SqlBeginStream** file and the **SqlEndStream** file are used to define SQL statements that are passed to the database. The begin stream statement is run before the first EDR arrives, and the end stream statement is run after the last EDR is processed.

The **HeaderTableDefinition**, **DetailTableDefinition** (only if the **NumberOfRows** registry entry is set to **1**), and **TrailerTableDefinition** files are used to describe the table in which the EDRs must be stored. For more information, see "HEADER, TRAILER, and DETAIL Table Definitions".

First the table name is defined with ${TABLE}, which is replaced by the definition in the parameter file. Then the column names of the table are defined. Each line holds a column of the table and its EDR container field, which is mapped into this column. The value is the EDR container field with external and alias names delimited by a comma (**,**).

```
${HEADER_TABLE}
;RECORD_TYPE      = HEADER.RECORD_TYPE   , HDR_RECORD_TYPE
```

Each column starts with the **FieldDelimiter** after the table name.

Because schema definition is not supported by the **RWDBBulkInserter**, when the **NumberOfRows** registry entry is set to **1**, the columns are defined like the example above. For bulk insertion into the database when **NumberOfRows** is greater than 1, the column definition is deleted from the table definition file and the schema of the table definition in the database is used to create the **BulkInsert**. The following example shows the DETAIL table definition file for **BulkInsert**:

```
${DETAILTABLE}
;DETAIL.RECORD_TYPE   , RECORD_TYPE
;DETAIL.RECORD_NUMBER , RECORD_NUMBER
...
```

In this case, each sequence of the EDR container field definition in the table definition file must be in the same order as the column definition in the database table.

Each logical block must end with two number signs (##) except within the table definition file. A comment line must start with two slashes (//).

All these files result in one final configuration file for each stream with replaced parameter keys within the SQL statements, except optional registry keys. The registry keys (**StreamNameAlias** and **RowNumAlias**) are replaced immediately before the SQL statement is passed to the database.

The name of the final configuration file is the stream name. It is located in the ControlPath. If the **SaveConfigurationFile** registry entry is enabled, a suffix is added. If processing is successful, the suffix is **.done**; otherwise, it is **.err**.

Specifying the Destination

The destination value can be applied to an appropriate field in the HEADER record of an output module. In the BRM format, this entry is used to fill the recipient field in the HEADER record.

Specifying the Source

The source value can be applied to an appropriate field in the HEADER record of an output module. In the BRM format, this entry is used to fill the sender field in the HEADER record.

Handling Empty Output Streams

Because of splitting and rejecting, there is a possibility that an output stream may contain only the HEADER and TRAILER records. In this case, the production of a default DETAIL record can be forced. If the **WriteDefaultEdr** registry entry is set to **True**, every DETAIL table contains at least one DETAIL record.

If the **DeleteWithoutDetails** registry entry is set to **True**, all insert and update operations will be rolled back and the default record will be deleted.

These settings are ignored if the output is for a reject stream.

Parameter File

Items in bold text are parameter keys.

```
DETAIL_TABLE
sol42_out
##
HEADER_TABLE
sol42_out_header
##
TRAILER_TABLE
sol42_out_trailer
##
```

HEADER, TRAILER, and DETAIL Table Definitions

HeaderTableDefinition

The format of **HeaderTableDefinition** does not depend on the **NumberOfRows** registry entry. The format of **HeaderTableDefinition** is as follows:

```
${HEADER_TABLE}
;RECORD_TYPE      = HEADER.RECORD_TYPE   , HDR_RECORD_TYPE
```

TrailerTableDefinition

The format of **TrailerTableDefinition** does not depend on the **NumberOfRows** registry entry. The format of **TrailerTableDefinition** is as follows:

```
${TRAILER_TABLE}
;RECORD_TYPE      = TRAILER.RECORD_TYPE   , TRR_RECORD_TYPE
```

DetailTableDefinition

The format of **DetailTableDefinition** depends on the **NumberOfRows** registry entry. When NumberOfRows is set to **1**, the format of **DetailTableDefinition** is as follows:

```
${DETAIL_TABLE}
;record_type      = DETAIL.RECORD_TYPE   , RECORD_TYPE
;record_number    = DETAIL.RECORD_NUMBER , RECORD_NUMBER
```

When NumberOfRows is greater than **1**, the format of **DetailTableDefinition** is as follows:

```
${DETAIL_TABLE}
;DETAIL.RECORD_TYPE   , RECORD_TYPE
;DETAIL.RECORD_NUMBER , RECORD_NUMBER
```

SqlBeginStream

Identifiers in bold text are registry entries. They are replaced before the statement is passed to the database.

One statement:

```
insert into stream_process (streamname, startdate,runmode,numberofrow,destination,source,info)
values
// this is a comment line
(__StreamName__, sysdate,'Debug',500,'${TABLE}','sol42_detailin','testing')
```

More statements:

```
begin
insert into tab1 (col1, col2) values (val1, val2);
insert into tab2 (col1, col2, col3) values
(val1, val2, val3);
// if an SQL block is used there must be a semicolon at the end
end;
```

SqlEndStream

Identifiers in bold text are registry entries. They are replaced before the statement is passed to the database.

```
update stream_process set enddate = sysdate,
edrnum = __RowNum__,
// duration only valid, if startdate and sysdate within one day
duration=
(3600*to_char(sysdate,'HH24')+60*to_char(sysdate,'MI')+to_char(sysdate,'SS'))-
(3600*to_char(startdate,'HH24')+60*to_char(startdate,'MI')+to_char(startdate,'SS'))
where
streamname = __StreamName__
```

Generated Configuration File

```
SqlBeginStream
insert into stream_process (streamname, startdate,runmode,numberofrow,destination,source,info)
values
(__StreamName__, sysdate,'Debug',500,'sol42_out','sol42_detailin','testing')
##
HeaderTableDefinition
sol42_out_header
##
DetailTableDefinition
sol42_out
##
TrailerTableDefinition
sol42_out_trailer
##
SqlEndStream
update stream_process set enddate = sysdate,
edrnum = __RowNum__,
duration=
(3600*to_char(sysdate,'HH24')+60*to_char(sysdate,'MI')+to_char(sysdate,'SS'))-
(3600*to_char(startdate,'HH24')+60*to_char(startdate,'MI')+to_char(startdate,'SS'))
where
streamname = __StreamName__
##
```

5

Configuring EDR Preprocessing

Learn how to configure the modules used for preprocessing event data records (EDRs) in the Oracle Communications Billing and Revenue Management (BRM) Pipeline Manager. For example, it describes how to discard duplicate EDRs.

Topics in this document:

Handling Duplicate EDRs

Use the FCT_DuplicateCheck module to find EDRs that have already been processed and send them to a special output stream. This prevents you from charging a customer twice for the same usage.

As a pipeline processes EDRs, the following occurs:

1. The FCT_DuplicateCheck module keeps a record of EDRs that have already been processed.
2. As new EDRs are processed, the module checks for duplicate EDRs by comparing data in the incoming EDRs with data in the EDRs that have already been processed.

   The FCT_DuplicateCheck module uses the following criteria to check for duplicate EDRs:

   - **The date of an EDR.** If an EDR is older than a certain date, it is not processed and no further checking is performed. It is unlikely that a duplicate EDR will be processed much later than an original EDR.
   - **The data in an EDR.** You configure which data to use when comparing a new EDR against EDRs that have already been processed.
   - **A search key**. An EDR is considered a duplicate if it has the same search key and the same values contained in the fields used for data comparison.
3. If an EDR is a duplicate, it is flagged with an error so other modules do not need to process it, and it is moved to a separate output stream.

Configuring Duplicate EDR Checking

To enable duplicate EDR checking, configure the FCT_DuplicateCheck module. See "FCT_DuplicateCheck" in *BRM Pipeline Manager Reference* and the following topics:

- Setting Date Parameters for Storing Processed EDRs

- Specifying the Fields to Use for Duplicate Check
- Specifying a Search Key for Duplicate Check
- Managing FCT_DuplicateCheck Data Files
- About Storing EDRs in a Database Instead of Files
- Using Duplicate Check with Multiple Pipelines
- Suspending Duplicate EDRs

To define the output stream for duplicate EDRs, use the **Output** configuration in the registry. You typically use the OUT_Reject module. See "Sample Output Configuration" and "OUT_Reject" in *BRM Pipeline Manager Reference* for more information.

> **Note**
>
> When you enable duplicate checking, you must also set the transaction manager **RedoEnabled** entry to **True**.

Setting Date Parameters for Storing Processed EDRs

To check for duplicate EDRs, you need to store a record of the EDRs that have already been processed. The FCT_DuplicateCheck module then checks incoming EDRs against the previously processed EDRs. If you receive a high volume of EDRs, you cannot store a record of *all* EDRs. Therefore, to limit the number of EDRs you store, you specify date criteria. If an EDR is older than the specified date, it is not processed.

> **Note**
>
> The date of an EDR is derived from its DETAIL.CHARGING_START_TIMESTAMP field.

To specify date parameters for EDR storage, you use the following date settings, specified in the FCT_DuplicateCheck registry. Depending on an EDR's date relative to these settings and on whether the FCT_DuplicateCheck module is connected to the database, the EDR is ignored, stored in a file, stored in the database, or stored in memory.

- **StoreLimit**: Specifies the oldest date that previously-processed EDRs can be stored. EDRs dated earlier than the **StoreLimit** date are ignored and not processed by the FCT_DuplicateCheck module.
- **BufferLimit**: Specifies the oldest date that previously-processed EDRs can be stored in memory. All EDRs whose date is equal to or later than the **BufferLimit** date are stored in memory. The FCT_DuplicateCheck module searches the memory directly, thus improving performance.

The **StoreLimit** date must be equal to or earlier than the **BufferLimit** date. For example, if the **StoreLimit** is June 1, the **BufferLimit** date can be June 1 or later.

> **Note**
>
> Because **StoreLimit** and **BufferLimit** are specified as absolute dates (for example, August 2, 2025), you need to change them daily. You change them by using an FCT_DuplicateCheck semaphore file entry. See "FCT_DuplicateCheck" in *BRM Pipeline Manager Reference* for more information.

Specifying the Fields to Use for Duplicate Check

You configure which data to use when comparing a new EDR against EDRs that have already been processed. A typical duplicate check for a phone call compares the A number, B number, and start time in a new and processed EDR. If the data in all fields match, the new EDR is flagged as a duplicate.

To specify which fields to use, use the FCT_DuplicateCheck **Fields** registry entry. See "FCT_DuplicateCheck" in *BRM Pipeline Manager Reference* for more information.

The following example shows a typical configuration:

```
Fields
{
    1  =  DETAIL.BASIC_SERVICE
    2  =  DETAIL.B_NUMBER
}
```

> **Note**
>
> Do not use the DETAIL.CHARGING_START_TIMESTAMP field for duplicate checking.

Specifying a Search Key for Duplicate Check

The duplicate check search key identifies duplicate EDRs. An EDR is considered a duplicate if it has the same time, the same search key value, and the same values for fields listed in the IFW_DUPLICATECHECK table or in the **Fields** lists stored as a record in memory.

The search key is used as a key to the internal data structure. For example, if the search key is **A_NUMBER**, then **A_NUMBER** is the hash key used to find the data in memory or in a file for the EDR being checked.

Managing FCT_DuplicateCheck Data Files

The FCT_DuplicateCheck module uses data files to store EDR data while the EDRs are processed. The file name syntax is:

FileName_TransactionId.**dat**

FileName is the value entered in the **FileName** registry entry:

```
FileName = duplicateData
```

The file contains the data from the fields specified in the registry and the EDR date. At the end of each transaction, the FCT_DuplicateCheck module saves the data from memory to disk in a new transaction file.

> **Note**
>
> - To store EDRs in files for duplicate checking, configure unique **FileName** settings, **Path** registry settings, or both for each module.
> - To store EDRs in the database for duplicate checking, use the FCT_DuplicateChecking module's **DataConnection** registry entry to connect the module to the database. See "About Storing EDRs in a Database Instead of Files".

The duplicate check transaction files should be backed up routinely when the pipeline is not processing EDRs. Temporary files, however, should not be backed up.

To restore transaction files from a backup, shut down the pipeline and restart after restoring backed up transaction files. You can restore duplicate check data without shutting down provided that the pipeline is not processing any EDRs. There is no semaphore to reload the data file, but resetting the **StoreLimit** or **BufferLimit** settings results in a reload. The values of these settings do not need to be changed from their original startup registry settings.

After an abnormal termination, temporary files may be left behind (there should be only one because file mode should only be used with a single pipeline). These files correspond to transactions that were never committed, and the input files associated with these transactions will be reprocessed upon restarting. You should delete these temporary files before restarting. Temporary files use the suffix **.tmp**.

About Storing EDRs in a Database Instead of Files

If you use the FCT_DuplicateCheck module's **DataConnection** registry entry to connect the module to the database, the module handles EDRs as follows:

- EDRs whose date is equal to or later than the **StoreLimit** date and earlier than the **BufferLimit** date are stored in the database instead of in files. If an EDR is stored in the database, files are not created.
- EDRs whose date is equal to or later than the **BufferLimit** date are stored in memory and in files.

> **Note**
>
> To avoid using excessive disk space when checking for duplicate EDRs, use the FCT_DuplicateCheck module's **DataConnection** registry entry to connect the module to the database. (The **StoreLimit** and **BufferLimit** date settings, and connecting the module to the database, are designed to help maintain performance without overloading memory.)

When the FCT_DuplicateCheck module is connected to the database, an entry is added to the IFW_DUPLICATECHECK database table for each unique EDR. For each duplicate EDR, the error INF_DUPLICATE_EDR is reported and no entry is added to the table.

If the pipeline transaction is canceled, all the rows with the current transaction ID are removed.

If you do not use the FCT_DuplicateCheck module's **DataConnection** registry entry to connect the module to the database (the default), the module handles EDRs as follows:

- EDRs whose date is equal to or later than the **StoreLimit** date and earlier than the **BufferLimit** date are stored in files instead of the database. (Each EDR is assigned a

value that is stored in memory. If the EDR is stored in files, the module checks the memory for the value, which points to the EDR in the file. The module then reads the EDR data from the file.)

- EDRs whose date is equal to or later than the **BufferLimit** date are stored in memory and in files.

> **Note**
>
> Although not connecting the module to the database enables faster checking for duplicate EDRs, it uses a large amount of disk space.

Create Database Tables for Duplicate Check Data

Use the FCT_DuplicateCheck **TableSuffix** registry entry to create multiple IFW_DUPLICATECHECK tables when you run multiple pipelines. You typically do this when you run a separate pipeline for each type of service.

For example, if the pipeline processes GSM EDRs, you can use GSM as the table suffix to use a table named IFW_DUPLICATECHECK_GSM.

You need to manually create the tables and the indexes. For example:

- IFW_DUPLICATECHECK_GSM
- BIDX_DUPCHK_DATA_GSM

Using Duplicate Check with Multiple Pipelines

When you use the FCT_DuplicateCheck module, you must process the EDRs for each account in the same pipeline. If you use duplicate check in multiple pipelines for the same account, duplicate calls may get processed by different pipelines and will not be recognized as being duplicate.

Suspending Duplicate EDRs

By design, a duplicate EDR is not considered as a suspended EDR and is sent to a different output stream than suspense handling stream. Therefore, duplicate EDRs are not handled by suspense handling.

If your business requires to process duplicate EDRs as suspended EDRs, you can do the following:

1. Change the entry **StreamName = DuplicateOutput** to **StreamName = SuspenseCreateOutput** in the FCT_DuplicateCheck registry.

   This will cause the duplicate check reject file to be routed to the suspense handling stream.

2. Add an iScript in the pipeline after duplicate check processing, to check for EDR error. If the error is duplicate check error, set the error to some other error and set error status as major.

   This will cause the duplicate EDR to be processed by FCT_Reject and FCT_Suspense plugins.

> **Note**
>
> You should ensure that duplicate EDRs are associated with a specific error code to be able to manage and monitor the EDRs in Suspense Manager Center.

Assembling EDRs

Use the FCT_CallAssembling module to assemble multiple CDRs into a single EDR that pipeline modules can rate. You typically need to assemble CDRs for long phone calls or GPRS sessions that have been recorded in multiple CDRs.

The default behavior of FCT_CallAssembling is designed to assemble *time duration* calls. This is appropriate for wireless voice calls that are rated based on how long the call lasts. You can also configure this module to assemble calls in a manner appropriate for EDRs rated based on the *volume of data sent*. This is appropriate for a long data transfer session, such as downloading a movie. You can choose to collect both time duration and data volume from multiple CDRs, and a number of other matrixes, such as:

- Volume sent
- Volume received
- Number of units
- Retail charge amount
- Wholesale charge amount

For example, a GPRS session might last for 24 hours. The network might be configured to generate an intermediate CDR every 30 minutes. This GPRS data session is recorded by several partial CDRs. If you rate by volume per session, you use the FCT_CallAssembling module to assemble the partial CDRs into one EDR before rating.

However, remember that the more metrics that you collect, the more system resources you will use.

How FCT_CallAssembling Classifies EDRs

FCT_CallAssembling uses the LONG_DURATION_INDICATOR EDR field to classify assembled calls.

The purpose of LONG_DURATION_INDICATOR changes when calls are assembled. CDRs arrive with this field set to one of these values, which identify the type of call segment:

- **F**: The first segment of the call.
- **L**: The last segment of the call.
- **I**: An intermediate segment of the call.

After FCT_CallAssembling assembles these call segments into an EDR, it gives the EDR one of these long duration indicators to specify its status.

- **C**: Complete call. The first and last segments have both arrived, enabling the call's duration to be calculated. If intermediate call segments arrive after the call is complete, they are given a long duration indicator of **XC**, **XO**, or **XP** and may not be rated.
- **S:** A single CDR containing the entire call.

- **SL**: Slice of a call. Used if **KeepCallOpen = True** or **MaxDuration = True**. Composed of the first call segment and any intermediate call segments that have arrived when the call is flushed. This part of the call is rated. When any other intermediate segments and the last segment arrive, the call is given a long duration indicator of **C** (complete), and these segments are rated.
- **P**: Partially assembled call. Used if **FlushLimit = True** or **KeepCallOpen = false**. Partially assembled calls are rated with the information in whatever segments have arrived. Subsequent call segments are given a long duration indicator of **XP** and are not rated.
- **XC:** Late intermediate EDR. This long duration indicator is for intermediate call segments that arrive after the call is marked complete and rated. Late intermediate call segments are not rated; instead they are used for auditing.
- **XO**: Late overlap EDR. Used if time duration rating is used, and **DropLateCDRs=False**. This status indicates that the call segment was flushed before this CDR arrived, and it represents a time duration period already rated. These late segments are not rated; instead they are used for auditing.
- **XP**: Late timeout EDR. Used if **DropLateCDRs** = **False**. This status indicates that the call timed out before this CDR arrived. These late segments are not rated; instead they are used for auditing.

Managing the Call Assembling Data Files

Open EDRs are stored in a data file. You configure the path and file name in the **Path** and **FileName** entries in the FCT_CallAssembling registry. The syntax for the file name is:

*FileName_TransactionID***.dat**

This file is read at startup and reloaded after rollback. While processing, data is stored in a temporary file.

The FCT_CallAssembling module processes the EDRs in a single transaction. You should backup the work files routinely when the pipeline is not processing any EDRs.

To restore from a backup, shut down the pipeline and restart after restoring backed up work files. Work files use the suffix **.dat**.

After abnormal termination, temporary files may be left behind. Temporary work files use the suffix **.tmp**. These correspond to transactions that never committed. Therefore, the input files associated with these transactions will be reprocessed upon restarting. Delete these temporary files before restarting.

> **Note**
>
> Never delete the most recent **.dat** file.

Configuring Call Assembling

To configure call assembling, see "FCT_CallAssembling" in *BRM Pipeline Manager Reference* and the following topics:

- Rating Calls by Time Duration
- Rating Calls by Implied Time Duration
- Rating Calls by Volume of Data Sent

In addition to specifying how to perform call assembly, you can configure FCT_CallAssembling to do the following:

- Specify an amount of time (in seconds) that is an acceptable amount of time error for each call. See "Specifying a Time Error" for more information.
- Keep calls open indefinitely, and rate them in segments periodically. See "Rating Continuous Data Calls by Segment" for more information.
- Limit the effect of **FlushLimit** to calls with specific service codes. See "Rating Partial Calls by Service" for more information.
- Capture data from **Basic Detail Record** fields for the **L** call segment. See "Capturing Fields From the Last Call Record " for more information.
- Get a report about calls being assembled. See "Tracking the Status of Assembled Calls" for more information.

If you are upgrading, see "Migrating Call Assembling Data Between Releases and Pipelines".

Rating Calls by Time Duration

By default, the FCT_CallAssembling module assembles the time duration of a call so it can be rated.

To assemble EDRs for their time duration, the FCT_CallAssembling module identifies partial EDRs by using the following EDR attributes:

- **The chain reference.** This ID identifies which event the partial EDR belongs to. Multiple partial EDRs that belong to the same event all have the same chain reference.

  > **Note**
  >
  > Chain reference must be unique for each call instance. Oracle does not support identical chain references across several call events.

- **The long duration indicator.** For a list of these indicators, see "How FCT_CallAssembling Classifies EDRs".

Parts of EDRs can be processed out of order; for example, the Last segment might arrive before the First segment. The FCT_CallAssembling module manages EDRs by tracking their status:

- As a soon as a first or intermediate call segment record arrives, the EDR is stored in a data file and the state is set to *Open*. See "Managing the Call Assembling Data Files".
- By default, if a First or Last segment is already stored in a file, and the matching Last segment or First segment arrive, the record state is changed to *Closed*. The EDR is moved back into the pipeline. Any Intermediate segments that belong to the assembled EDR that arrive after the assembled EDR is closed are ignored.

  You can also direct this module to wait for all CDRs before closing a call, or close it after you send in a semaphore. This allows you to rate incomplete calls or calls that never receive a first or last record.

The time duration for a call is calculated as follows:

```
Call Duration = [(Start_Time_Of_Last_Segment) - (Start_Time_Of_First_Segment)] +
Duration of the Last Segment
```

For example, if there is a call which starts at Jan 01, 2009 12:00:00 PM and ends at Jan 01, 2009 at 12:30:00 PM. The EDR will have the following information:

- First Segments TimeStamp as Jan 01, 2009 12:00:00 PM and Duration
- Last Segment TimeStamp would be Jan 01, 2009 12:25:00 PM + 300(Seconds)

In this case, the call duration will be calculated as follows:

```
[(12:25:00 PM) - (12:00:00 PM)] + 300 = 1800 Seconds
```

Rating Incomplete Time Duration Calls

Some EDRs can never be completely assembled because the First, Intermediate, or Last segment never arrives. In this situation, you can choose to close these calls using either the **FlushLimit** semaphore entry or the **MaxDuration** startup registry entry. Both of these entries force calls to be flushed and rated after the time limit you set.

Here is a comparison of the two entries:

- **MaxDuration** is a *startup* registry entry. It takes effect when you start the pipeline. Each time a new call segment arrives, it recalculates the total time duration for that call. If the new time duration exceeds the limit you set, the call is flushed and it remains open for more CDRs. You set the time duration in seconds. To change the setting, you must restart the pipeline.

  For details, see "Using MaxDuration to Rate Incomplete Calls".

- **FlushLimit** is a *semaphore* registry entry used with **KeepCallOpen=True**. It sets a maximum age a call can have before being flushed. When you send in the semaphore, the pipeline calculates whether a call exceeds the maximum age. If a call exceeds the **FlushLimit** setting, FCT_CallAssembling creates and rates the EDR and then closes the call. You set the time limit in days. A setting of **0** flushes all calls. A setting of **1** flushes all calls that have been opened more than 1 day. You can change the **FlushLimit** setting every time you send in a semaphore.

  For details, see "Using FlushLimit to Rate Incomplete Calls".

Using MaxDuration to Rate Incomplete Calls

MaxDuration is a startup registry entry that directs FCT_CallAssembling to rate segments of a wireless call periodically. This entry specifies the total amount of time duration (in seconds) that a call can have before the call segments that have arrived are rated. FCT_CallAssembling recalculates the call duration for every call each time a new call segment arrives and compares it to the **MaxDuration** setting. If the new time duration equals or exceeds the setting for **MaxDuration,** FCT_CallAssembling creates an EDR to rate the existing portion of the call.

To use the **MaxDuration** entry, add it to the startup registry and start (or restart) the pipeline.

Table 5-1 illustrates FCT_CallAssembling behavior with **MaxDuration** set. In the example, the CDRs did not arrive in chronological order.

Example call with KeepCallOpen=True and MaxDuration=28800 (8 hours)

Table 5-1 FCT_CallAssembling and CDRs

CDR	Start time	End time	Duration (hours)	FCT_CallAssembling takes this action...
F	1:00	4:00	3	Waits. **MaxDuration** setting not reached.

Table 5-1 (Cont.) FCT_CallAssembling and CDRs

CDR	Start time	End time	Duration (hours)	FCT_CallAssembling takes this action...
I1	4:00	7:00	3	Waits. **MaxDuration** setting not reached.
I2	7:00	10:00	3	Creates EDR for 9 hours of call duration. **MaxDuration** setting exceeded.
I3	10:00	13:00	3	Waits. **MaxDuration** setting not reached.
I5	16:00	19:00	3	Creates EDR for 9 hours of call duration, even though a CDR is missing. **MaxDuration** setting exceeded.
I4	13:00	16:00	3	Not rated; call duration already rated. This EDR is either emitted as **XO** or dropped, depending on the **DropLateCDRs** setting.

In this example, if **MaxDuration** is set to **28800** seconds (8 hours), FCT_CallAssembling rates the EDRs of a call with a total time duration of more than 8 hours. This call arrives in 3-hour segments and will be rated after the third segment arrives, and FCT_CallAssembling calculates that the 9-hour call duration exceeds the 8-hour limit. When the **MaxDuration** setting is reached, the call segments are flushed and rated, the long duration indicator for the call is set to **SL**, and the call is left open for more call segments.

If a CDR is missing, FCT_CallAssembling adds the missing call time represented by the EDR if it can. In the example above, the I4 call segment arrived last, but the time duration it represented had already been rated when the I5 segment arrived. FCT_CallAssembling calculated the time duration by subtracting the start time from I3 and the end time from I5. The difference was 9 hours of time duration, which exceeds the 8-hour setting, so an EDR was created to rate that duration.

Using FlushLimit to Rate Incomplete Calls

FlushLimit assembles the calls and emits them into the pipeline for rating. The call has a long duration indicator set to **P** for "Partial."

In this example, **FlushLimit** flushes all calls the have a CHARGING_START_TIMESTAMP older than five days from today.

```
FlushLimit=5
```

To flush all incomplete calls, use 0; for example:

```
FlushLimit=0
```

The flush operation does not happen immediately. Instead, it happens at the arrival of the next pipeline transaction. This is done to ensure that Flush operations happen within the context of a transaction.

Removing Incomplete Time Duration Calls

When you flush EDRs, they re-enter the pipeline as part of the current transaction. The EDRs are still stored in the work files (**.dat** and **.EDR**), with a state of Timeout. This prevents a late-arriving call segment from re-opening a call that has already been flushed. If you are sure that no further segments will arrive, use the **RemoveLimit** semaphore entry to remove calls from the work files.

The **RemoveLimit** entry removes all calls with the Closed or Timeout status, but leaves calls with a state of Closed_Rejected or Timeout_Rejected alone. Closed_Rejected or Timeout_Rejected calls will be recycled. However, if you are sure that calls with the Closed_Rejected or Timeout_Rejected status will not be recycled, use a **RemoveRejectedLimit** semaphore entry to remove these calls from the work files.

Dropping Late Calls

You can drop late EDRs from the pipeline entirely, or send them through as non-valid EDRs. If you send them through the pipeline, you can use them for auditing.

Use the **DropLateCDR** registry entry to specify how to handle the output of late EDRs:

- **True** (Default) = Drop late EDRs from the pipeline. The EDRs are counted in the report of late-arriving EDRs.
- **False** = Send late EDRs through the pipeline as non-valid. The EDRs are not counted in the report of late-arriving EDRs. They are not rated.

Rating Calls by Implied Time Duration

By default, FCT_CallAssembling calculates a call's time duration by using the difference between the first EDR's start time and the last EDR's end time. This includes the time duration for any CDRs between the first and last that have not yet arrived. The time duration for these missing CDRs is included in the EDR and rated.

If **KeepCallOpen=True** calls are rated in segments. When a semaphore with **KeepCallOpen=True** is sent in:

- All the CDRs that have arrived for a call are assembled.
- An EDR is emitted and rated.
- The call is kept open for more CDRs.

When the next semaphore is sent in, all CDRs that have arrived since the last semaphore are assembled as an EDR and rated.

By default, FCT_CallAssembling calculates a call's time duration by subtracting the end time of the last CDR that has arrived from the start time of the first. Because the CDRs between the first and last are not used in the calculation, it does not matter if they have arrived when the EDR is created. If a CDR between the first and last CDRs is missing when the time duration is calculated, the missing CDR is dropped when it finally does arrive.

Example time duration registry

```
CallAssembling
 {
  ModuleName = FCT_CallAssembling
  Module
   {
      Active            = True
      AssembleVolume    = False
      AssembleSGSN      = False
      SplitAtGaps       = False
      MaxDuration       = 900
      Path              = ./data/assy
      FileName          = calls
      Mode              = Normal
      RejectMissingChain = True
      CallDurationTolerance = 59
```

```
    DropLateCDRs          = False

    AssembledEDR  {
        1 = Detail.custom_fields_from_last_edr1
        2 = Detail.custom_field_from_last_edr2...
    }
  }
}
```

> **Note**
>
> If you use both the FCT_CallAssembling and FCT_Reject in the same pipeline, use the FCT_Reject module **CallAssemblingModule** registry entry to ensure that complete EDRs are recycled. See "FCT_Reject" in *BRM Pipeline Manager Reference*.

Rating Calls by Volume of Data Sent

Use the **AssembleVolume** and/or **AssembleSGSN** registry entries to direct FCT_CallAssembling to rate calls by the volume of data sent. Both **AssembleVolume** and **AssembleSGSN** ensure that you capture the entire volume of data sent for a single call, and they both direct FCT_CallAssembling to rate a call only after all of its call records have arrived.

If you do not plan to rate calls by volume of data sent, leave **AssembleVolume** set to **False**. This saves system resources by disabling the registry entries which rate by volume.

The volume-based rating entries protect against lost revenue. Call records can arrive in any order, so it is not unusual for an intermediate segment to arrive after the first and last segments have arrived and been rated. In this case, any intermediate segments that arrive after the call is closed are dropped, and all the volume they contain are lost (and so is the revenue).

By default, FCT_CallAssembling calculates the time duration for each CDR *individually*. FCT_CallAssembling subtracts the end times from the start times of each CDR to calculate the time duration. This module then adds the time durations of all CDRs as they arrive to create a grand total for the call.

If your business requires that all non-contiguous CDRs be rated as separate EDRs (TAP requires this) set the **SplitAtGaps** registry entry to **True**. If not, then set this entry to **False**, and non-contiguous CDRs will be collected into a single EDR, saving system resources.

However, if you are rating calls by volume of data sent, lost intermediate call records will cause calls to remain open indefinitely. Send in a **Flushlimit** entry frequently to avoid this.

For information on data calls that you want to keep open indefinitely and rate periodically, see "Rating Continuous Data Calls by Segment".

Example volume of data call registry

```
CallAssembling
 {
  ModuleName = FCT_CallAssembling
  Module
   {
      Active            = True
      AssembleVolume    = True
      AssembleSGSN      = True
      SplitAtGaps       = False
      Path              = ./data/assy
      FileName          = calls
```

```
      RejectMissingChain = True
      CallDurationTolerance = 59
      DropLateCDRs          = False

      AssembledEDR  {
          1 = Detail.custom_fields_from_last_edr1
          2 = Detail.custom_field_from_last_edr2...
      }
   }
}
```

Example TAP volume of data call registry

This example adheres to the TAP 3.10 standard. Volume and SGSN data are recorded precisely at the start and end times of each call segment. Because **SplitAtGaps = True**, if any call segments are missing, the segments before and after it are emitted as separate EDRs.

```
CallAssembling
 {
  ModuleName = FCT_CallAssembling
  Module
   {
      Active            = True
      AssembleVolume    = True
      AssembleSGSN      = True
      SplitAtGaps       = True
      Path              = ./data/assy
      FileName          = calls
      RejectMissingChain = True
      CallDurationTolerance = 59
      DropLateCDRs          = False

      AssembledEDR  {
          1 = Detail.custom_fields_from_last_edr1
          2 = Detail.custom_field_from_last_edr2...
      }
   }
}
```

Specifying a Time Error

Use the **CallDurationTolerance** startup registry entry to specify an amount of time (in seconds) that is an acceptable amount of time error for each call.

> **Note**
>
> The default value for this entry correctly handles most calls, so you do not need to change this entry unless you notice a problem.

Mobile phone calls are commonly split into multiple call records. Each call record should start at the same second that the previous one ended. For example, a call record with an *end* time of 12:01:30 should be followed by the next call record with a *start* time of 12:01:30. Unfortunately, it is common for these start and end times to either not quite match, or to overlap slightly (time error). The reason may be that the call records come from different routing switches with clocks that have not been synchronized, or the switches themselves have different time tolerances.

The **CallDurationTolerance** default value is 60 seconds of tolerance to compensate for this time error. If a call has less than 60 seconds of error, a call is considered complete and sent for rating. Otherwise the call is left open and must be closed with a semaphore entry. A single call with 10 call records, each overlapping by 3 seconds, creates a call with 30 seconds of time error. This 30-second time error is less than the **CallDurationTolerance** 60-second time limit, so the call is considered complete and sent down the pipeline for rating.

> **Note**
>
> Setting this entry too low causes an inordinate number of calls to be left open indefinitely. Setting this entry too high can cause the pipeline to rate calls with missing call segments. The 60-second default value is appropriate for most BRM implementations.

Rating Continuous Data Calls by Segment

Use the **KeepCallOpen** semaphore registry entry together with **FlushLimit**, to keep calls open indefinitely, and rate them in segments periodically. This feature is designed for data calls that are kept open continuously (days at a time). These long data calls are usually rated periodically to capture revenue.

KeepCallOpen is an update registry entry sent in the **FlushLimit** semaphore.

For example, a bank might keep a continuous call open to each of its ATMs to pass data back and forth. Using the default behavior, the call would not be rated for days. You will probably want to capture the revenue for these types of calls periodically, perhaps every 12 or 24 hours. Setting **KeepCallOpen** to **True** keeps, these calls open. You then rate these calls in segments by sending in a **FlushLimit** semaphore entry.

If you set **KeepCallOpen** to **True** and send it in with the **FlushLimit** semaphore every 12 hours, an EDR is created twice a day, each EDR rating the previous 12 hours of call time and volume.

KeepCallOpen is an entry in the **FlushLimit** update registry entry.

By default **KeepCallOpen** is set to **False**. The default behavior directs this module to rate the call when the first and last segments have arrived, or when the **FlushLimit** is sent, whichever comes first. Any subsequent records are ignored.

Rating Partial Calls by Service

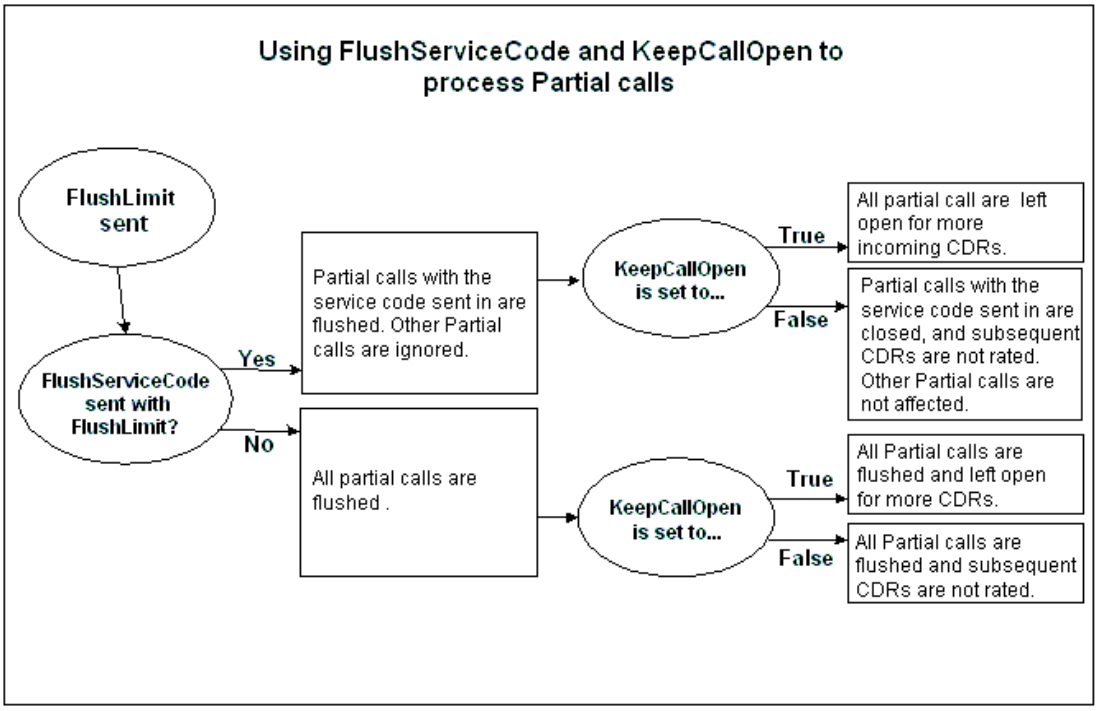
Use the **FlushServiceCode** semaphore registry entry to limit the effect of **FlushLimit** to calls with specific service codes. **FlushServiceCode** is sent as an entry in the **FlushLimit** update registry entry. If the **FlushServiceCode** entry matches the value of the DETAIL.INTERNAL_SERVICE_CODE field of the call segment, the call is flushed. All other partial calls are ignored.

For example, the following **FlushServiceCode** entry directs the pipeline to rate only calls with the **dat** service code:

```
{
  FlushLimit = 0
  FlushServiceCode = dat
}
```

Figure 5-1 shows how the **FlushServiceCode** and **KeepCallOpen** semaphore registry entries interact.

Figure 5-1 FlushServiceCode and KeepCallOpen Semaphore Interaction



Capturing Fields From the Last Call Record

Use the **AssembledEDR** startup registry entry to capture data from **Basic Detail Record** fields for the **L** call segment.

The final record of a call has a long duration indicator of **L** (last), however the **L** call record may not be the last to arrive at the pipeline. In some cases it is important that information be captured from the **L** call record. For example, the number that terminated the call can only come from the **L** call record.

If **AssembledEDR** is used, the pipeline captures **Basic Detail Record** data from the **L** call record and adds it to the EDR emitted, regardless of whether it was actually the last call record received.

List any **Basic Detail Record** fields to capture data from in the **AssembledEDR** entry, including any custom fields that your business requires. The pipeline will then include the data from those fields on the EDR that is compiled and processed. See "Sample Registry" in *BRM Pipeline Manager Reference* for an example.

Tracking the Status of Assembled Calls

You can use a semaphore command to receive a report about the status of calls currently being assembled. The report provides the following information:

- The number of partially assembled calls.

- The number of EDRs currently waiting to be assembled.
- The number of late EDRs that are parts of calls that are no longer being assembled, for one of the following reasons:
  - The call was assembled and sent to the pipeline. In this case, the first and last portion of a call was processed, and an intermediate part arrived after the EDR had been sent into the pipeline.
  - The call was flushed by a **FlushLimit** semaphore, and another portion arrived after the call was flushed.

  The report includes the following data about late EDRs:
  - The number of late-arriving EDRs for flushed calls.
  - The number of late-arriving EDRs for assembled calls.
  - The total number of late-arriving EDRs.

In addition to creating the report, each assembled EDR includes a NUMBER_OF_CDRS field that stores the number of CDRs that were included in the EDR. You can use this data in an aggregation scenario to gather additional data about assembled calls.

> **Note**
>
> For EDRs that are not part of split calls, the FCT_CallAssembling module enters **1** in the NUMBER_OF_CDRS field.

Migrating Call Assembling Data Between Releases and Pipelines

When you upgrade BRM from one release to another or apply patches, you need to migrate the call data in your **.dat** files to the new format by using the XML support provided with the FCT_CallAssembling module.

You use the following semaphore registry entries to change the format of the data from one release or pipeline to another:

- **ExportDataToXml**
- **ImportDataFromXml**

For more information, see the semaphore file entries in "FCT_CallAssembling" in *BRM Pipeline Manager Reference*.

To migrate call assembly data, perform the following steps:

1. Export the data from your existing data files to an XML file by using a semaphore registry file with the **ExportDataToXml** entry:

```
ExportDataToXml
{
   CallsPerFile = Value
}
```

2. Import the data from the XML file into the data file with the new format by using the **ImportDataFromXml** entry:

```
ImportDataFromXml
{
   FileName = filename.xml
}
```

Assembling Calls with Multiple Pipelines

When you use the FCT_CallAssembling module, you must process the EDRs for each account in the same pipeline. If you assemble calls in multiple pipelines for the same account, the call segments may get processed by different pipelines and cannot be assembled.

Discarding and Skipping EDRs

You can use the FCT_Discard module to skip or discard EDRs:

- Skipping an EDR removes it from the pipeline.
- Discarding an EDR sends it to a different output stream. You will probably want to audit the information in EDRs with a LONG_DURATION_INDICATOR of **XC**, **XO**, or **XP** before you discard them.

In both the cases the state of the EDR becomes invalid. (To indicate a discarded EDR, a value is entered in the DETAIL.DISCARDING field.)

For example, you can filter the following EDRs:

- Discard EDRs that are older than three days and have a B_NUMBER that begins with 0049.
- Discard EDRs that have a RECORD_TYPE that begins with a 9 followed by two digits, an INTERN_SERVICE_CODE that ends with a 2, and a WHOLESALE_CHARGED_AMOUNT_VALUE of 0.

For information on the regular expressions you can use, see "About Using Regular Expressions when Specifying the Data to Extract".

To create a valid mapping, the data in the EDR must match all of the mapping data.

Configuring EDR Discarding

To configure EDR discarding:

1. Specify which EDRs to discard or skip. See "About Configuring Discard and Skip Expressions".
2. Configure the OUT_DevNull module as the output stream for discarded EDRs. See "Configuring Output of Discarded EDRs".
3. Configure the FCT_Discard module. See "FCT_Discard" in *BRM Pipeline Manager Reference*.
   - Use the FCT_Discard module **StreamName** registry entry to specify the output stream.
   - To send EDRs to different output streams without making them invalid, use the FCT_EnhancedSplitting module. To process EDRs that belong to accounts in different BRM database schemas, see "FCT_AccountRouter" in *BRM Pipeline Manager Reference*.

About Configuring Discard and Skip Expressions

To specify which EDRs to discard or skip, you create a set of regular expressions using a set of data fields. These fields are not mandatory so if they do not exist in the EDR description it does not matter. If all expressions match the available fields, the EDR will be removed.

You can create different discard rules for separate pipelines.

You define discard rules in Pricing Center or Pipeline Configuration Center (PCC). The discard rules are stored in the IFW_DISCARDING table.

Use the **Reload** semaphore file entry to reload regular expression patterns after you change them.

You can discard or skip EDRs based on the following:

- Rank. This specifies the order in which to evaluate the rules.
- Record type.
- Source network.
- Destination network.
- Call complete indicator.
- Long duration indicator.
- Usage class.
- Internal service code.
- GSM switch or GPRS switch.
- Tariff class.
- Tariff subclass.
- Connection type.
- Connection subtype.
- B number.
- The age of the EDR.
- If the wholesale charge should be zero.

For information on the regular expressions you can use, see "About Using Regular Expressions when Specifying the Data to Extract".

To configure the FCT_Discard module. See "FCT_Discard" in *BRM Pipeline Manager Reference*.

Configuring Output of Discarded EDRs

Use the OUT_DevNull module to handle EDRs that should be discarded from a pipeline.

To discard EDRs, do the following:

- Configure the FCT_Discard module. See "Discarding and Skipping EDRs ".
- In the pipeline, configure the OUT_DevNull module. See "OUT_DevNull" in *BRM Pipeline Manager Reference*.

OUT_DevNull is a submodule of the Output Collection module. All registry parameters and error messages are handled by the Output Collection module. See "Output Collection" in *BRM Pipeline Manager Reference*.

Generating Multiple TAP MOC and MTC Records

When you process TAP files, you use the ISC_TapSplitting iScript to split mobile originating and terminating EDRs into multiple EDRs when the CDR contains more than one basic service. The ISC_TapSplitting creates a new EDR for each additional basic service.

Splitting mobile originating and terminating EDRs enables CDR rejection when a basic service record associated with a CDR is in error. It also permits custom validations to be added prior to splitting.

Service information for all secondary services (supplementary, VAS, CAMEL), which are part of a basic service EDR, are added to the last basic service EDR. Charge information for these secondary services are added to the last charge breakdown record of the last basic service EDR.

EDR splitting is performed after TAP validation. TAP files and EDRs that are rejected due to errors are not split. After the new EDRs are created, the original EDR is deleted.

To configure ISC_TapSplitting, see "ISC_TapSplitting" in *BRM Pipeline Manager Reference*.

Using Rules to Send EDRs to Different Output Streams

> **Note**
>
> To send EDRs to different output streams for the Rated Event (RE) Loader, use the IRL_EventTypeSplitting iScript. See "Sending EDRs to Pipeline Output Streams".

Use the FCT_EnhancedSplitting module to specify different output streams for EDRs based on rules. For example:

- You can split EDRs for different service types into different output streams.
- You can split EDRs from roaming outcollects and incollects into different streams.

To send EDRs to different output streams, you define a set of rules. For example, you can send all telephony EDRs from a specific network to a different output stream.

You can use the following data in the rules:

- Record type.
- Service code.
- Usage class.
- Source and destination network.
- Switch.
- Trunk in/out.
- A number and B number area code.
- Normalized C number area code (forwarded or routed number).

This example assigns all telephony EDRs with an A number starting with 49 to the Out49 system brand:

```
Service code :   TEL
A number:        0049.*
System brand:    Out49
```

In addition you can specify the order in which to evaluate the rules, and the dates when the rule is valid.

The FCT_EnhancedSplitting module evaluates each EDR against the rules. The first rule that matches the criteria defines the system brand to use. The system brand is identified by a code. You use that code to map the system brand to an output stream in the registry. In this example, system brand **Out49** is mapped to the **EdrOutputOut49** output:

```
SystemBrands
{
Out49 = EdrOutputOut49
}
```

You can create separate splitting rules for different pipelines.

You can use a semaphore file entry to reload a new set of rules.

Configuring Enhanced Splitting

To configure enhanced splitting, you do the following:

1. Configure system brands. Each system brand is mapped to an output stream.
2. Use Pricing Center or Pipeline Configuration Center (PCC) to configure the splitting rules. Each rule is associated with a system brand. If an EDR matches a rule, it uses the system brand defined in the rule.

   To create a valid mapping, the data in the EDR must match all of the mapping data.

   For the data fields, you use regular expressions. For information on the regular expressions you can use, see "About Using Regular Expressions when Specifying the Data to Extract".
3. Configure an output stream for each system brand. See "Configuring EDR Output Processing".
4. Configure the FCT_EnhancedSplitting module. This defines the output stream for each system brand. See "FCT_EnhancedSplitting" in *BRM Pipeline Manager Reference*.

Sending EDRs to Pipeline Output Streams

When you load rated events in the BRM database, RE Loader loads events for separate services from separate directories. Therefore, your pipeline needs to send EDRs to separate output streams for each internal service code. To do so, you use the IRL_EventTypeSplitting iScript.

To specify the output stream, you edit the **IRL_EventTypeSplitting.data** file. This file maps service codes to output streams. For example, this entry maps the TEL service code to the TelOutput stream:

```
TEL;TelOutput
```

> **Note**
>
> You can also use the FCT_EnhancedSplitting module to send EDRs to different output streams based on EDR content. See "Using Rules to Send EDRs to Different Output Streams".

The IRL_EventTypeSplitting iScript must run after the FCT_ServiceCodeMap module and before FCT_Reject module.

To configure the IRL_EventTypeSplitting iScript, you configure it as part of the FCT_IRules module. See "FCT_IRules" in *BRM Pipeline Manager Reference*.

For more information, see "IRL_EventTypeSplitting" in *BRM Pipeline Manager Reference*.

Using Pipeline Manager with Multiple Database Schemas

In a multischema environment, you start a separate instance of Pipeline Manager for each BRM database schema.

- Use the FCT_AccountRouter module to find the account and send the EDR to the correct instance of Pipeline Manager. The FCT_AccountRouter runs in its own instance of Pipeline Manager.
- Use the DAT_AccountBatch module to supply data to the FCT_AccountRouter module. It runs in the same instance of Pipeline Manager as the FCT_AccountRouter module. See "DAT_AccountBatch" in *BRM Pipeline Manager Reference*.

To find the A number customer and the B number customer, the FCT_AccountRouter module does the following:

1. The module looks for the following data in the EDR:
   - The internal service code that indicates which data can be used to identify the account. For example, if the internal service code is a telephony service, the identifying data is the A number. A different service might use the IMSI as the identifier.

     You identify which data to use by using Pricing Center or Pipeline Configuration Center (PCC).
   - The timestamp for the EDR. The timestamp is important because telephone numbers can be used by different accounts at different times.
2. The module uses the DAT_AccountBatch module to look up the account. See "DAT_AccountBatch" in *BRM Pipeline Manager Reference*.

   > **Note**
   >
   > - If no A customer is found, the EDR is rejected. If the B customer is missing, no error is generated.
   > - Because phone numbers can be recycled, the search is made on data from BRM audit objects.
3. The DAT_AccountBatch module returns the database number.
4. The FCT_AccountRouter sends the EDR to the correct pipeline, using the configuration defined in the registry.

Setting Up Account Identification in Multischema Systems

To set up account identification in a multischema system, do the following:

1. Configure service and account data in the Pipeline Manager database:
   - Configure internal service codes. The DAT_AccountBatch module retrieves account information based on which services are rated.
   - Configure how the FCT_AccountRouter module looks up accounts (for example, by telephone number or IMSI). You specify which data is used for identifying accounts when you configure the FCT_Account module.
   - (Optional) Configure account ID prefixes to use for handling duplicate telephone numbers. You configure this when you configure the FCT_Account module.
2. Configure the FCT_AccountRouter. See "FCT_AccountRouter" in *BRM Pipeline Manager Reference*.

   > **Note**
   >
   > Since the FCT_AccountRouter module needs the internal service code, the FCT_AccountRouter module must be placed after the FCT_ServiceCodeMap module in the pipeline.

3. Configure the DAT_AccountBatch module **UseAsRouter** registry entry.

   If set to **True,** the module is used by the FCT_AccountRouter module to route EDRs to separate Pipeline Manager instances. See "FCT_AccountRouter" in *BRM Pipeline Manager Reference*.

   If set to **False** (the default), the module is used by the FCT_Account module to get account data.

   See "DAT_AccountBatch" in *BRM Pipeline Manager Reference*.

6

Setting Up EDR Enrichment

Learn how to enrich event data record (EDR) data for rating by the Oracle Communications Billing and Revenue Management (BRM) Pipeline Manager.

Topics in this document:

Identifying the Network Operator/Service Provider

If your network supports multiple network operator/service providers (NO/SPs), you can use the FCT_NOSP module to identify the various NO/SPs. This module uses the source and destination codes and A number prefix in the EDR to assign a new source and destination code for the NO/SP.

Use this module when you need to identify the NO/SP and the NO/SP information is not available in the EDR; for example, when mobile networks are separated by means of the A number and you need to segment calls.

To map NO/SP data, you do the following:

1. Use Pricing Center or Pipeline Configuration Center (PCC) to create NO/SP maps. See "Creating an NO/SP Map".

   You can also use a file to configure the NO/SP map.
2. Configure the FCT_NOSP module. See "FCT_NOSP" in *BRM Pipeline Manager Reference*.
3. Configure the DAT_NOSP module. See "DAT_NOSP" in *BRM Pipeline Manager Reference*.

   If you store NO/SP data in a file, see "Creating an NO/SP Data File".

Creating an NO/SP Map

To create NO/SP mappings, you use Pricing Center or Pipeline Configuration Center (PCC) or a text file to set up the data that specifies how to map NO/SPs.

The mappings are based on the following data:

- Map group.
- The order in which mappings are applied. The first NO/SP map that matches is used.
- The source network and network destination recorded in the CDR.

- The phone number prefix to match.
- The new network source and destination to use in the EDR.

For information on the regular expressions you can use, see "About Using Regular Expressions when Specifying the Data to Extract".

Pricing Center or Pipeline Configuration Center (PCC) stores the mappings in the IFW_NOSP database table.

For information on using a file to specify NO/SP maps, see "Creating an NO/SP Data File".

Creating an NO/SP Data File

You can store data for the DAT_NOSP module in the Pipeline Manager database or in a text file.

The configuration file must be an ASCII file. Each row defines one mapping rule. All fields in a row are separated by a semicolon (;). The fields in one row have the following order:

1. MAPGROUP (NOT NULL)
2. RANK (NOT NULL)
3. OLD_SOURCE
4. OLD_DESTINATION
5. A_PREFIX
6. NEW_SOURCE (NOT NULL)
7. NEW_DESTINATION (NOT NULL)

Setting Up Social Numbers

In some cases, customers want a B number to not appear on an invoice. For example, some countries require that certain called numbers remain anonymous, such as the number for a treatment center. You can set up social numbers to hide specific B numbers.

The social flag functionality is run by the FCT_SocialNo module. When it finds a social number, the module sets a flag in the EDR B_MODIFICATION_INDICATOR field. You can use this flag to customize how to handle the number; for example, remove the last three digits or not allow the EDR to be included in an invoice.

To set up social numbers:

1. Use Pricing Center or Pipeline Configuration Center (PCC) to specify social numbers.

   To specify social number, you specify the number that identifies it internally and give the number a descriptive name.

   The FCT_SocialNo module looks for an exact match of this number in the EDR.

   You can also use a file to specify social numbers. See "Creating a Social Number Data File".
2. Configure the FCT_SocialNo module. See "FCT_SocialNo" in *BRM Pipeline Manager Reference*.

Creating a Social Number Data File

You can specify social numbers in Pricing Center, Pipeline Configuration Center (PCC), or a file.

The social number data file uses the following syntax:

```
number1
number2
...
```

The number uses the same format as the normalized B number in the EDR.

```
International-access-code c\Country-code National-destination-code Access-code
```

For example:

```
0014085555555
```

Creating Call Destination Descriptions

You can set up descriptions for call destination area codes. For example, a prefix of 001408 can be described as "San Jose, California." The description is displayed on the customer's bill.

Prefix/description mapping is performed by the FCT_PrefixDesc module. You set up a mapping between prefixes and descriptions. The module finds the best match for the prefix and adds the description to the EDR DESCRIPTION field.

To set up prefix/description mapping:

1. Map prefixes and descriptions in Pricing Center or Pipeline Configuration Center (PCC).

   > **Note**
   >
   > You can also use a text file. See "Creating a Prefix/Description Data File".

2. Configure the FCT_PrefixDesc module. See "DAT_PrefixDesc" in *BRM Pipeline Manager Reference*.
3. Configure the DAT_PrefixDesc module. See "DAT_PrefixDesc" in *BRM Pipeline Manager Reference*.

Setting Up Prefix/Description Mapping

To set up prefix/description mapping in Pricing Center or Pipeline Configuration Center (PCC), enter the following:

- The area code prefix. The prefix must have the same format that is used for normalization within a pipeline. The FCT_PrefixDesc module does not normalize numbers.
- The prefix type:
  - National
  - International
  - Special

> **Note**
>
> If you use a file to store the mappings, you do not enter any prefix type. See "Creating a Prefix/Description Data File".

- The description to use, for example, "New York City."

> **Note**
>
> All prefix/description pairs must be unique.

Creating a Prefix/Description Data File

You can define prefix/description mappings in the Pipeline Manager database, or in a file.

The mapping file has the following structure:

```
Prefix1; Description1
Prefix2; Description2
```

Mapping Multiple Phone Numbers to a Single Number

Customers with more than one telephone number sometimes want to get only one bill for all their numbers. You can map multiple telephone numbers to one number. This mapping is performed by the FCT_CliMapping module. The module uses the A number to search for a mapping entry. If there is a mapping entry, the A number is replaced by the Caller Line Identification (CLI) number

For example, a customer with a primary number 14085722000 could have five extra telephone numbers but would like to be billed as if they originated from the same number. To accomplish this for all the calls originating from the other five numbers, the A number is mapped to 14085722000 in the EDR.

To configure CLI mapping:

1. Define the mapping in a file. See "Creating a CLI Mapping File ".
2. Configure the FCT_CliMapping module. See "FCT_CliMapping" in *BRM Pipeline Manager Reference*.

Creating a CLI Mapping File

As shown in Table 6-1, the FCT_CliMapping module requires an ASCII mapping file that contains the numbers to map to a single number.

- Every new line defines a mapping.
- Fields are separated by semicolons (;).
- There should be no semicolon at the end of a line.

Table 6-1 FCT_CliMapping Module File Structure

Column	Name	Position	Length	Format	Description
1	CLI_FROM	1	25	X(25)	Start CLI
2	SERVICE_CODE	27	5	X(5)	Service Code
3	MAPPING_CLI	33	25	X(25)	Mapping CLI
4	CUST_NUMBER	59	10	X(10)	Customer number
5	SUBSC_NUMBER	70	20	X(20)	Subscriber number
6	ISDN_RANK	91	1	X(1)	Rank of the ISDN number

Managing Number Portability

Customers may want to change network operators but retain their existing phone number. You can maintain number portability data to keep a track of the network operator a customer uses.

For maintaining number portability data, the DAT_NumberPortability module loads the data from the number portability file into the memory (pipeline's run-time memory).

The FCT_NumberPortability gets the number portability data from the DAT_NumberPortability module. The data includes the CLI, the new network operator ID, and the date that the customer changed network operators. The FCT_NumberPortability module uses the date of the event to determine which network operator applies. If the new network operator applies, FCT_NumberPortability module updates the new network operator ID in the EDR.

BRM's number portability feature support batch pipeline (offline mode). See "Number Portability for the Batch Pipeline".

For batch pipeline rating, the FCT_NumberPortability module updates the EDR with the appropriate network operator ID based on the time stamp when the network operator changed.

For real-time pipeline rating, the FCT_NumberPortability module updates the EDR with the appropriate network operator ID based on the time stamp when the network operator changed. The FCT_Opcode module creates an opcode input flist containing the new network operator information. The flist is used for rating the calls in real time.

Number Portability for the Batch Pipeline

BRM uses semaphores to load the number portability data file using the batch process in the pipeline.

The number portability process for the batch pipeline makes the system inactive when the new number portability records are updated in the system.

About Number Portability Files

The number portability file is an ASCII file that stores the number portability data. The data from these files is loaded into the DAT_NumberPortability module.

You enter the following information in the number portability file:

- CLIs.
- The time stamp when CLIs are ported to a new network operator.
- The new network operator ID.

See "Creating a Number Portability Data File".

The following example shows a sample of number portability records in a number portability file:

```
Call Line ID            Date              Network Operator ID
408555                  20080101000000    D030
408555                  20080110000000    D010
```

The records in the example show that the subscriber changes the network operator to D030 on January 01, 2008. From January 01 to January 1, the subscriber is with the network operator D030. On January 10, the subscriber changes the network operator to D010.

BRM uses the following number portability files:

- Primary number portability file: Contains the primary number portability records that already exist.
- Delta number portability file: Contains the additional number portability records that need to be added to the memory and the primary number portability file.

You use the primary number portability file when you use the **reload** probe or the **Reload** semaphore to reload number portability data.

You use the delta number portability file when you use the **deltaLoad** probe to update the primary number portability file.

To manage number portability data, you can do the following:

- Purge data from the primary number portability file. To purge data from the file, use the purge utility (**purge.np_data.pl**). See "Purging and Reloading the Memory Records".
- Reload the data from the primary number portability file to the memory. See "Purging and Reloading the Memory Records".
- Append the data from the delta number portability file to the memory and subsequently to the primary number portability file. See "Appending Additional Number Portability Records".

Creating a Number Portability Data File

The DAT_NumberPortability module reads the required data from a reverse-sorted ASCII file.

Each row of the number portability file contains a CLI number, a date and time (when the numbers are ported), and a network operator ID as shown in Table 6-2.

Table 6-2 Input File for DAT NumberPortability module

Column	Format	Description
Call Line Identity Number	String	Specifies the telephone number or a part of the telephone number.
Portation date and time	*YYYYMMDDhhmmss*	Specifies the date and time of the number portation.
Network Operator ID	D*xxx*	Specifies the new telephone carrier.

An example of a sample file:

```
089761   20020910101230   D030
089761   20020912101230   D018
```

```
089545   20020820084000   D017
089545   20020920230010   D030
```

> **Note**
>
> The columns are separated by spaces.

Use the DAT_NumberPortability FileName registry entry to specify the file name for the number portability data file.

Purging and Reloading the Memory Records

The purge utility is used to delete existing records (from the number portability file) that are older than a time stamp specified in the utility. You can purge the data from the memory by using the purge utility. For purging the records from the memory, you must first purge records from the number portability file and then use the **reload** probe to ensure that the records in the number portability file and the memory are in sync.

To purge and reload number portability records:

1. Run the **purge_np_data.pl** utility to purge the primary number portability data file:

   `purge_np_data.pl` *`NP_FileName TimeStamp`* [**`-b`** *`backup_filename`*] [**`-n`**]

   where,

   - *NP_FileName* specifies the name of the primary number portability file. For example, **Primary_NP_Data.data**.
   - *TimeStamp* specifies the date prior to which all the number portability records are purged. For example, 20080105000000.
   - **-b** *backup_filename* specifies the name of the backup file that will contain the unpurged number portability records. For example, **Primary_NP_Data.bak**.

   This command takes a backup of the existing number portability records in the backup file and deletes all number portability records prior to the date (timestamp) specified from the primary number portability file.

2. Initiate the Reload semaphore to reload the purged number portability data into the memory.

   This ensures that the records in the number portability file and the memory are in sync.

   > **Note**
   >
   > For the sample entry of **Reload** semaphore, see "Sample Semaphore File Entry" in *BRM Pipeline Manager Reference*.

   If the reload operation fails, the memory data will contain all the unpurged data. The primary number portability data file is moved to a *Pipeline_home***/npdata/error** directory. *Pipeline_home* is the directory where you installed Pipeline Manager.

Appending Additional Number Portability Records

You can use the delta number portability file to append additional or newly ported records to the primary number portability file. The delta number portability file contains the additional records that are in the same format as the primary number portability file. By default, the delta number portability file is stored in the *Pipeline_home* directory. You can also manually add the additional records in the primary number portability file but this process is cumbersome when you need to add many records.

To append additional number portability records:

1. Initiate the **AdditionalNumPortData** semaphore with the delta number portability file name as a value. The delta number portability file contains additional entries in the same format as the number portability file.

   The additional entries are first added to the memory and then the memory data is dumped into the primary number portability file so that the records in the memory and the primary number portability file are in sync.

   > **Note**
   >
   > For the sample entry of **AdditionalNumPortData** semaphore, see "Sample Semaphore File Entry (DAT_NumberPortability)" in *BRM Pipeline Manager Reference*.

2. Initiate a **PrintData** semaphore to print the records to a file.

   All number portability data in the memory is copied to the file. By default, this file is created in the *Pipeline_home* directory.

   > **Note**
   >
   > For the sample entry of the **PrintData** semaphore, see "Sample Semaphore File Entry" in *BRM Pipeline Manager Reference*.

Setting Up Number Portability

You must set up number portability to maintain the number portability data to keep track of the network operator a customer is using. You can configure number portability for the following:

- Batch pipeline
- Real-time pipeline

Setting Up Number Portability for Batch Pipeline

To set up number portability for the batch pipeline:

1. Configure the FCT_NumberPortability module. See "FCT_NumberPortability" in *BRM Pipeline Manager Reference*.
2. Configure the DAT_NumberPortability module. See "DAT_NumberPortability" in *BRM Pipeline Manager Reference*.

When you configure the DAT_NumberPortability module, you specify the following:

- The search method. See "Configuring Number Portability Search".
- Normalization. See "Configuring Normalization for Number Portability".
- The number portability data file. See "Creating a Number Portability Data File".

Setting Up Number Portability for Real-Time Pipeline

To set up number portability for the real-time pipeline:

1. Configure the FCT_NumberPortability module. See "FCT_NumberPortability" in *BRM Pipeline Manager Reference*.
2. Configure the DAT_NumberPortability module. See "DAT_NumberPortability" in *BRM Pipeline Manager Reference*.

   When you configure the DAT_NumberPortability module, you specify the following:

   - The search method. See "Configuring Number Portability Search".
   - Normalization. See "Configuring Normalization for Number Portability".
   - The number portability data file. See "Creating a Number Portability Data File".
3. Configure the ISC_PopulateOpcodeAndUtilBlock_Diameter iScript in the registry file and place it in the processing pipeline after the FCT_NumberPortability module. See "ISC_PopulateOpcodeandUtilBlock_Diameter" in *BRM Pipeline Manager Reference*.
4. Configure the ISC_MapNetworkOperatorInfo iScript and place it after the ISC_PopulateOpcodeAndUtilBlock_Diameter iScript in the registry file. See "ISC_MapNetworkOperatorInfo" in *BRM Pipeline Manager Reference*.
5. Configure the FCT_Opcode module and place it in the processing pipeline after the ISC_PopulateOpcodeAndUtilBlock_Diameter iScript. See "FCT_Opcode" in *BRM Pipeline Manager Reference*.

Configuring Number Portability Search

You can configure which of the following search methods the DAT_NumberPortability module uses to find a phone number's current network operator:

- **Best match** searches the number portability file for objects with the best combination of matching phone number prefix and recent port date. For example, a number portability file includes the following entries and a pipeline module receives a CDR with a date of November 2 and the phone number 4085551234:

  ```
  Call Line ID      Date              Network Operator ID
  408               20081001000000    D030
  408555            20080801000000    D029
  408555            20080708000000    D028
  ```

  The DAT_NumberPortability module returns network operator D029 because the entry contains the most recent port date with the most matching prefix numbers.
- **Exact match** searches for the first object that exactly matches a given phone number.
- **Any prefix match** searches for the first object with a matching prefix.

To configure the search method, use the DAT_NumberPortability **SearchMethod** registry entry. See "DAT_NumberPortability" in *BRM Pipeline Manager Reference*.

Configuring Normalization for Number Portability

Use the following DAT_NumberPortability registry entries to specify number normalization data:

- CountryCode
- NationalAccessCode
- InternationalAccessCode
- InternationalAccessCodeSign

For example, you can normalize this phone number:

04106760279

so that it becomes:

00494106760279

See "FCT_NumberPortability" in *BRM Pipeline Manager Reference*.

7

Setting Up Pipeline Aggregation

Learn how to compile data from event data records (EDRs) using the Oracle Communications Billing and Revenue Management (BRM) Pipeline Manager aggregation feature.

Topics in this document:

About Aggregation

You use the aggregation feature to compile data from event data records (EDRs). Aggregation is typically used for the following:

- To filter and summarize data for compiled statistics about service usage, customer activity, network activity, dealer commissions, and so forth. For example, you can compile the sum of charges per customer per month.
- To aggregate data and use for rating. For example, you can compile GPRS usage records and rate the aggregated amount.
- To output a back-out file used as input for rerating.

About Setting Up Aggregation Pipelines

Because the results of aggregation are typically not used by other modules, and because the aggregation process uses more resources than rating, aggregation is typically performed by a separate pipeline. You can run an aggregation pipeline parallel to a rating pipeline, so that the same input files will be processed by both pipelines.

The aggregation pipeline processes EDRs in transactions, typically one transaction per input file. When the transaction is finished, the data is written from memory to a file.

After every transaction, the aggregated results are written to a file that contains the transaction ID. For every transaction and aggregation scenario a result and a control file is created. The control files are used by the Database (DB) Loader utility to load the results into the database. See "Creating Aggregation Scenarios".

You define the control file in the FCT_AggreGate registry entries. After the data is processed by the pipeline, you use DB Loader to load the data from the file to a database. This can be the same database used by Pipeline Manager or a different database. You can run the utility automatically or manually. See "Database Loader" in *BRM Pipeline Manager Reference*.

See "Creating Aggregation Scenarios".

See "FCT_AggreGate" and "DAT_ScenarioReader" in *BRM Pipeline Manager Reference*.

> **Note**
>
> If you configure aggregation for rerating, ensure that the FCT_AggreGate module is working in back-out mode.

About Aggregation Scenarios

You aggregate data by creating aggregation scenarios (see "Creating Aggregation Scenarios"). An aggregation scenario specifies which data to use and how to process the data. You use the following criteria when setting up aggregation scenarios:

- **Filter conditions**: Allows you to choose which EDRs to aggregate; for example, you can specify all calls for a single customer, service, time period, or charge.

  > **Note**
  >
  > Conditional filters for combined attributes are not supported.

- **Grouping rules:** Allows you to group calls according to various criteria; for example, you can aggregate sums based on levels of duration, as follows:
  - All calls less than one minute.
  - All calls from one to five minutes.
  - All calls over five minutes.
- **Aggregation functions:** You can do the following with the data:
  - Count the number of selected events.
  - Sum the values of selected events.
  - Generate the square sum from values of selected events.

You can use any EDR field for filtering, grouping, or aggregating. However, aggregation criteria are limited to the following:

- The data must be in the EDR.
- The data must be available for an overall aggregation, typically of a NUMBER type.

> **Note**
>
> An aggregation scenario is analogous to an SQL SELECT statement with a GROUP BY claus.
>
> For example, select all EDRs where the time period is peak time, then group the calls by levels of duration (for example, 0-1 minute, 1-5 minutes), and aggregate the charges for each level of duration.

Creating Aggregation Scenarios

When you create an aggregation scenario, you define filter, grouping, and aggregation functions. When you run an aggregation pipeline, you specify which scenario to use in the FCT_AggreGate registry.

Defining Filter Criteria

Define the following when defining filter criteria for a scenario:

- The status of the filter criteria (active or inactive).
- The EDR field that the filter is applied to.
- The data type to use; for example, string, number, or date/time.
- The value to filter with.
- The function used for aggregating:
  - Count
  - Sum
  - Square sum
- The decimal precision for the aggregated result.

Specifying Scenario Attributes

In addition to defining the filters, groups, and aggregation functions, you can use the registry to define the following attributes for each scenario:

- The EDR container description for the scenario.

> **Note**
>
> You can only set up filters and groups for fields included in the EDR container description.

- A default table name where the aggregation results are stored.
- The maximum number of aggregations held in memory before writing the data to a file. Enter **0** to specify an infinite number.
- Default directories for the following:
  - Where to store the aggregation results file (temporary and finished data).
  - Database (DB) Loader control files.
- A default delimiter for the single values of grouping and aggregation fields.

About Creating Groups

You use grouping rules to group and order aggregation results. Grouping rules are applied to EDR container fields and associated with a rank. For example, aggregation results can be grouped first by the event originator, then by zones, and then by time periods, or by duration, area code, or geographic zone.

When you define groups, you define the database columns that contain the aggregated data. For example, if you group aggregation results by duration, a column for that value is created in the table in the results file.

A group includes the following attributes:

- The EDR field for which the values are grouped; for example, time period or duration.
- The column name where the aggregations for the group are written in the results file.
- The data type for the grouped data:
  - String
  - Number
  - Date/time
- If applicable, the output format for the data. This can be date/time values or string format; for example UPPER, LOWER or SUBSTR(x,y).

About Creating Classes for Groups

You create classes to group results for a type of data. For example, to group aggregation results by the duration of events, you can create the duration classes shown in Table 7-1.

Table 7-1 Duration Classes

Class	Value to group by
Duration Class A	0 to 60 seconds
Duration Class B	61 to 300 seconds
Duration Class C	301 to 600 seconds
Duration Class D	More than 600 seconds

When creating a class, you define the following categories:

- The name; for example, duration.
- The data type:
  - String
  - Number
  - Date/time
- Class items. For each class item you define the following:
  - The name; for example, duration.
  - The data type (string, number, or date/time)
  - The value; the following is an example duration class item:

    D=1-5 & HH=08-20

    You specify values by using regular expressions. Use the same expressions defined when "Defining Filter Criteria".

> **Note**
>
> For values that do not match any other class items, specify an "undefined" class item for every class.

About Linked Classes

You can create multiple linked classes to group results by more than one category. For example, if you create groups for time period and duration, you can group results as follows:

- Peak time: 0-1 minute
- Peak time: 1-5 minutes
- Peak time: over 5 minutes
- Off-peak time: 0-1 minute
- Off-peak time: 1-5 minutes
- Off-peak time: over 5 minutes

About Defining Class Dependencies

During aggregation, class groupings are processed in order as defined in a tree structure. To define this order, you assign classes to nodes in the tree structure. For example, to aggregate data by time and duration, data is aggregated first by the time class, and then by the duration class.

8

Migrating Pipeline Manager Data Between Test and Production Systems

Learn about the Pipeline Manager data migration features of Oracle Communications Billing and Revenue Management (BRM).

Topics in this document:

About Pipeline Manager Data Migration

You use the Pipeline Manager data migration feature to create, test, and modify pipeline rating data in a development environment, then deploy it to your production environment. This capability provides increased flexibility and security by isolating development and testing activities from production systems.

This is the basic workflow of the Pipeline Manager data migration process:

1. **Set up identical development and production systems.**

   To ensure data integrity and testing validity, the pipeline rating data in the development system must be exactly the same as the data in the production system. The two systems diverge as you make changes to the development environment, but they converge again when you export these changes and import them into the production system.

   See "Setting Up Development and Production Environments".

2. **Create and modify charges and discounts on the development system.**

   Using standard Pricing Center procedures with some modifications, you create and modify pipeline pricing data such as charges, discounts, and pricing. You use change sets to organize your work. Change sets are groups of related changes that are managed and exported as a whole. Change sets ensure data integrity by locking objects when necessary.

   See "Understanding Change Sets" and "Understanding Locks and Associations".

3. **Test your changes on the development system.**

   You should test your changes thoroughly to ensure that they work as you expect. The data migration features in Pricing Center guard against major errors such as referring to objects that don't exist, but you must test the business content of your changes to make sure that you achieve the results that you want.

   See "Testing Change Sets".

4. **Export change sets from the development system.**

After testing, you can export a single change set or a group of change sets to a file. The file contains the information required to recreate the modified objects in the production database. If you export a group of change sets to a single file, you can specify the order in which the change sets are exported.

See "Planning the Export Process".

5. **Import change set files into the production environment.**

   The import process reads the file created during export from the development system. All changes are validated to ensure data integrity. For example, the import process checks that all objects referred to by objects in the change set exist in the production database. If errors are found, the entire file is rejected and no changes are made.

   See "Planning the Import Process".

Understanding Change Sets

Change sets are the basis of the Pipeline Manager data migration features in Pricing Center. Change sets track the changes you make and make it possible to treat those changes as a whole. They also enforce rules about which objects can be added, changed, or deleted. See "Understanding Locks and Associations".

For example, a change in policies at your business makes it necessary to modify several pricings. You can make all these modifications as part of single change set that you export from the development system and import into the production system as a whole. Using a change set guarantees that the production system receives exactly the same changes that you made in the development system.

The content of a change set is determined by the changes you make while that change set is *active*. When data migration is enabled, you must have an active change set before you make any changes to rating and pricing data. You can have only one active change set at a time and a change set can be active for only one user at a time. Every change you make is part of the active change set.

You can activate, inactivate, and reactivate change sets freely. For example, you can activate one change set to make a change to a screen, then switch to another change set to make a different change. When you exit from Pricing Center, the active change set is automatically inactivated to make it available to other users.

You can create as many change sets as you need. In some cases you may use only a single change set to include all the changes associated with a particular pricing change. In other cases, you may need to set up a number of different change sets for various parts of the project. See "Organizing Work into Change Sets".

You use the Change Set Manager, a screen in the Pricing Center application, to create and manage change sets. See "About the Change Set Manager ".

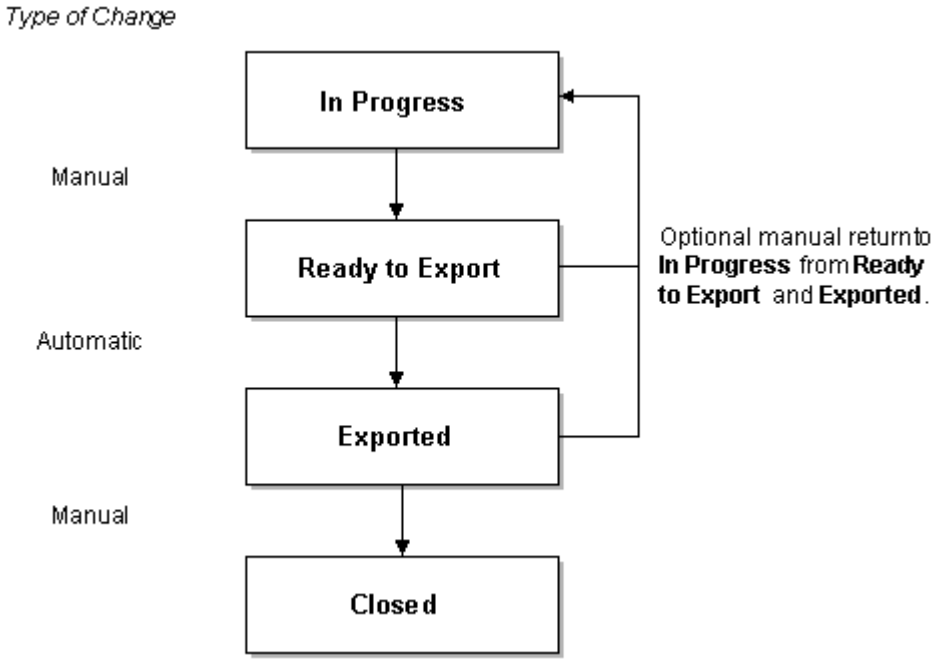
Understanding the Change Set Life Cycle

Change sets follow a life cycle that dictates what you can and can't do. This life cycle is comprised of change set states through which change sets move. By default, the life cycle includes four states. You can add additional states to conform to your business practices.

> **Note**
>
> Managing the change set life cycle is a sensitive task. It is possible to invalidate testing scenarios and to create discrepancies between the test and production environments if you do not monitor and manage change set states carefully.

Figure 8-1 illustrates the default change set life cycle.

Figure 8-1 Default Change Set Life Cycle



There are four default change set states:

- **In Progress**. When you create a new change set, it is in the In Progress state, the working state for change sets. When an In Progress change set is active, you can make any modifications to the pipeline rating data that you need. From In Progress, you can manually change the state to Ready to Export or to a custom state that you define.
- **Ready to Export.** You change the state to Ready to Export when you confirm that all your data is complete and correct. You should test your data before switching to this state.

  In the Ready to Export state, you cannot make any changes to the data in the change set. If you need to make additional changes, you must manually switch back to In Progress.

  The normal next step is to export the change set to a file. When you export a change set, its state automatically switches to Exported.
- **Exported**. This state shows that the change set has been exported, but has not yet been imported into the production database. You cannot make any changes to the data in a change set in the Exported state.

  You must manually change the state to Closed when the file is imported.

> **Note**
>
> Changing to Closed state from Exported is a vital step. If you do not close the change set, all locks and associations remain active, potentially blocking the completion of other change sets.

If there is a problem importing the change set, you can manually return the change set to the In Progress state to make necessary changes. This should be a carefully managed task to avoid confusion about which change set file contains the correct data. See "Planning the Export Process".

- **Closed**. This is the end state for change sets. The Closed state indicates that a change set is complete and in production. The change set is no longer available for use and cannot be reactivated. Its locks and associations are released. You can view information about the modifications made in this change set, but can no longer back them out.

For more information, see "Customizing Change Set States".

Understanding Locks and Associations

Locks and associations are used by change sets to ensure data integrity, prevent contradictory changes, and maintain information about what data has changed or been affected.

- A *lock* prevents a data object from being modified or deleted by a change set other than the one that established the lock.
- An *association* indicates that a data object is referred to by a locked object. While an object can have only a single lock, it can have multiple associations. It can also have a lock and associations at the same time. An object with an association cannot be deleted until the association is removed.

When data migration is enabled, locks and associations are automatically implemented by Pricing Center, preventing you from making changes that violate the locking rules. For example, if a change set has locked a pricing, you can't modify that pricing with any other change set.

Even though locks and associations are enforced automatically, you should understand the rules that are used to enforce them. This knowledge will help you and your colleagues work efficiently and smoothly. For example, if you create a new calendar in a particular change set, locking prevents that calendar from being visible to other change sets until the change set that created the calendar is closed. You need to plan your work accordingly. See "Organizing Work into Change Sets" for more information.

The next section, "Understanding the Pricing Data Model", provides background information about how pipeline rating data is stored in the database. "Locking and Association Rules" provides information about the rules governing locks and associations.

Understanding the Pricing Data Model

When you create a charge, pricing, or another element of pipeline rating data, it is stored in a table in the Pipeline Manager database, but for the purposes of clarity, we can think of Pipeline Manager data as independent objects.

These objects have different kinds of relationships with each other. Many objects are *reusable*. They are elements such as calendars, time models, pricings, and zone models that can be referred to by many other objects. For example, a single calendar can be used by many charges.

Other objects have a *parent-child* relationship. One parent object refers to many different occurrences of the same type of child object.

Locking and Association Rules

When an object is modified, added, or deleted while a change set is active, that change set has a lock on the object. For example, if you make a change to a charge under Change Set 1, that charge is locked by Change Set 1 and cannot be modified or deleted by another change set until the lock is released.

These are the three most basic locking rules:

- An object can have only one lock. In other words, when an object is locked by one change set, it cannot be locked by another.
- An object that is locked by a change set cannot be modified or deleted by another change set.
- A newly created object is locked by the change set that created it.

These rules prevent change sets from making contradictory changes.

In addition to these basic locking rules, additional rules govern the objects related to locked objects. These rules work differently depending on whether the object is reusable or part of a parent-child relationship.

Rules for Reusable Objects

When a change set locks an object that directly refers to a reusable object, the change set creates an association to the reusable object. For example, when you modify a charge, the change set creates a lock on the charge and an association to the calendar referred to by the charge.

Associations are used to keep track of data that is potentially affected by changes made in a change set. They are also used to guard against deletion of objects that might cause data integrity problems.

These are the rules governing locking and associations for reusable objects:

- Objects with associations cannot be deleted until all associations have been removed. The deletion of associated objects is prohibited because it would cause data integrity problems; the locked object would have a reference to an object that no longer exists.
- Multiple change sets can create associations to the same object. For example, suppose Charge A and Charge B both refer to Calendar Z. Change Set 1 modifies Charge A, thereby creating an association to Calendar Z. Change Set 2 then modifies Charge B, which creates an additional association to Calendar Z.
- A change set can create an association to a locked object, except if that object is newly added by another change set. For example, if Change Set 1 has modified Pricing A and therefore locked it, Change Set 2 can still make a change that results in an association to that pricing. However, if Change Set 1 has *added* Pricing B, the new pricing is not visible to other change sets. No associations can be created to it by other change sets.
- A change set can obtain a lock on an associated object to make modifications. (The object cannot be deleted, however.) For example, suppose Change Set 1 modifies Charge A, which locks the charge and creates an association to Calendar Z. Change Set 2 can still lock the calendar for modification. It cannot delete the calendar because that would violate the rule about deleting associated objects.

- Associations are created only for objects directly referred to by a particular locked object. When you lock a charge, you create an association to the calendar it refers to, but you do not create an association to any objects referred to by the calendar.

Rules for Parent-Child Relationships

There is one main locking rule for parent-child relationships: a child object is locked when its immediate parent object is locked. For example, when you modify a charge, the charge and all its charge versions are locked.

Unlike associations, parent-child locking is recursive. In other words, not only the children of the parent object, but the children's children, are locked. For example, when you lock a charge, its versions are locked, which in turn causes the version's charge configurations, rate adjustments, and special day links also to be locked.

Because of the recursive nature of parent-child locks, you need to use some special techniques to minimize their impact. For example, to prevent all of a charge's children from being locked when you want to modify only a particular version, you can open the charge in read-only mode, then open the version you want to edit. Only that version and its children are locked, making it possible for other change sets to change other parts of the charge.

> **Note**
>
> You cannot *delete* a child object when you open its parent in read-only mode. For example, if you open a charge in read-only mode, then try to delete a charge version, you see an error dialog box. You must open and lock the parent object to delete a child object.

About the Change Set Manager

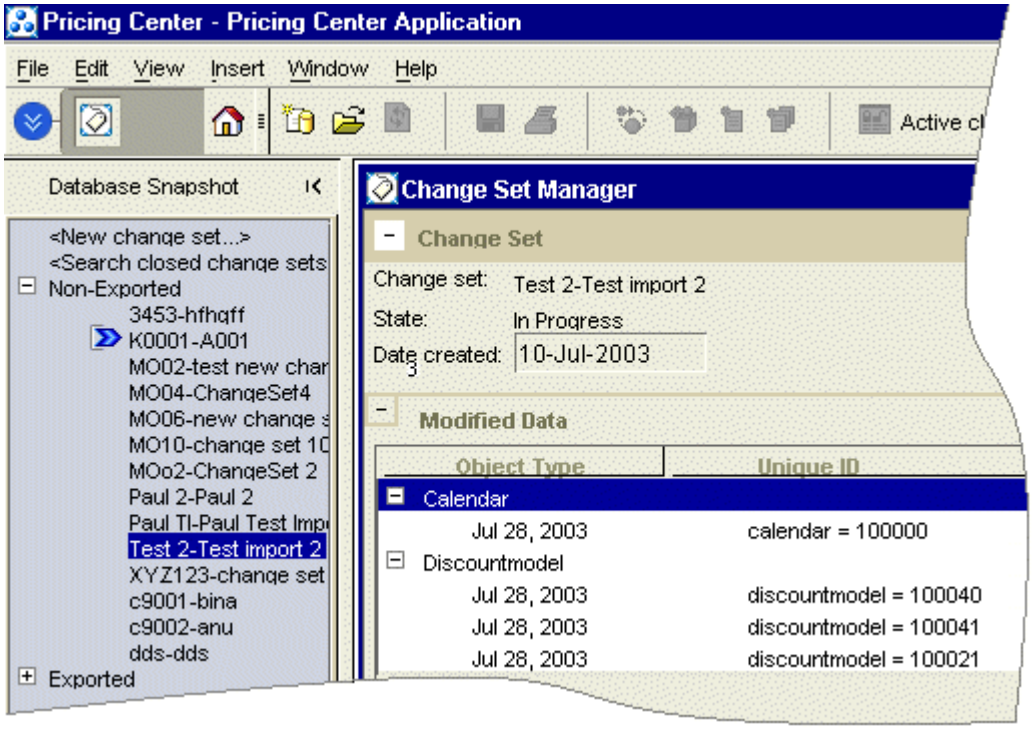
You use the Change Set Manager in Pricing Center to create and work with change sets. The Change Set Manager has two main areas:

- The navigation panel on the left enables you to open and create change sets. It lists all the available change sets in two sections: **Non-Exported** and **Exported**. You also search for closed change sets in the navigation panel.
- The Change Set Manager window displays the details of the open change set, including the name, state, description., the data modified by this change set, and the data associated with it. You can also activate a change set in the Change Set Manager window.

To open the Change Set Manager, click the Change Set Manager button () in the toolbar. Alternatively, you can choose **View - Change Set Manager**.

The Change Set Manager opens. The navigation panel on the left side of the screen shows the available change sets as shown in Figure 8-2.

Figure 8-2 Change Set Manager



In the Change Set Manager, you can do the following:

- Create new change sets
- Activate change sets
- View change set data
- Work with change set states
- Back out change sets
- Export change sets
- Import change sets

Using Pipeline Manager Data Migration Features in Your Business

The Pipeline Manager data migration features in Pricing Center provide a framework that you can use to create and test new pricing data for your business. Because every business is different, you need to develop procedures that meet your needs. This section provides guidance about integrating Pricing Center data migration features into your business.

> **Note**
>
> The Pricing Center data migration features guard against problems in data integrity, but don't validate content. You must be careful and methodical to ensure that your business content is correct and testable.

Setting Up Development and Production Environments

To ensure accurate testing, your development and production data models must be identical to begin with. Each database must contain exactly the same objects with exactly the same contents.

When you create new pricing data on the development system, the two databases diverge, but due to the data that you have introduced. You can therefore be confident that your testing will reveal only issues introduced by the new data. Later, when all your changes have been exported from your development system to your production system, the data models will again be identical. See "Copying Production Data to the Development System".

You must also enable data migration for the two systems. When you install Pricing Center, you can choose not to enable data migration, to enable only import capability, or to enable the entire feature. You can also enable or disable data migration after Pricing Center has been installed. See "Enabling Data Migration in Pricing Center".

Until you enable data migration, you cannot import or export data to or from either system.

> **Note**
>
> If you enable data migration for one instance of Pricing Center, you should also enable data migration for *all* other instances of Pricing Center that have access to the same test and production databases. Instances of Pricing Center without data migration enabled can make changes outside the scope of change sets, thereby causing inconsistencies in the data.

Another configuration step is setting up the change set life cycle that you want to use. The change set life cycle is based on your business process and reflects the stages that a change goes through from development to production. Depending on your business process, you may want to add change set states to the default life cycle. For example, you may want to add states called Testing and Approval. See "Understanding the Change Set Life Cycle" and "Customizing Change Set States".

Planning Your Work

You should plan your pricing data projects offline before using Pricing Center to implement them on the development system. You should know exactly which new pricing objects you need to introduce, which objects need to be changed, and which need to be deleted. You should also make sure to identify all reusable objects that will be modified so that you can test all the objects that refer to them.

The exact planning process you use depends on your business needs and the complexity of the data model you are implementing. Whatever process you use, make sure there is a solid connection between your offline process and your work in Pricing Center. Use a consistent naming policy so that you can track a change from its inception in the planning process to its

implementation in Pricing Center. For example, if your marketing department initiates a change through a formal change request, you can incorporate information from the change request into the Change Set ID, name, and description in Pricing Center.

Organizing Work into Change Sets

The way you organize change sets on your development system; how many change sets you use for a particular project and how changes are divided among them; can have a large impact on how efficiently you complete your work. Here are two reasons why you need to think carefully about change set organization:

- Change sets can be dependent on each other. For example, if you are creating a new reusable object in one change set, other change sets cannot view or reference that object until the first change set is closed. Dependencies can also be based on content. If you make changes to the content of a pricing in one change set, for example, any changes you make in other change sets that rely on this new content create a dependency.
- Locks created by one change set can prevent other change sets from accessing objects. For example, suppose two people are implementing two separate groups of changes, each in its own change set, and each change set requires the modification of the same pricing. When one user modifies the pricing, the other user is blocked until the first change set is closed.

You should use the information from the planning process to decide how many change sets to create and what to include in them. From your planning, you should develop a clear picture of the specific changes you need to make.

These are some of the factors to consider when organizing change sets:

- **The number and complexity of changes.** If you need to make only a few, simple changes, you can make them all in single change set. On the other, if you are working on a major overhaul of your pricing data, you should organize your work into multiple change sets.
- **The types of changes you are making.** Reusable objects can be referenced by many different objects, so changes to them can have a broad impact. To prevent one change set's locks from causing problems for other change sets, you can make all your reusable object changes in one change set that you export and import separately before other changes.
- **How many users are involved.** The larger the number of users involved in the implementation of a pricing data project, the more important it is to be very careful about planning and organizing the work. You can use change sets to divide work among users.

Testing Change Sets

Testing is vital to ensure that the pricing data you create is valid. Pricing Center prevents you from making errors that cause data integrity problems such as references to objects that no longer exist. But it is your responsibility to ensure that the *content* of your pricing data is valid: Pricing Center can ensure that a pricing exists, but it can't verify that it contains the correct information for your business.

Pricing Center does provide warnings in situations where your actions might cause problems. For example, when you open a Pricing Center screen for an object that has an association to another change set, you can get information about which change set created the association.

You should test the data as realistically as possible by using the pricing data in your development system to rate CDRs in an environment that closely resembles your production environment.

These are some guidelines for helping you decide what to test:

- When you modify a reusable object, test to ensure that the objects that reference it are still valid. For example, if you make a change to a calendar, you should test all the charges that refer to that calendar to make sure that the change doesn't cause an unexpected result.
- Keep in mind the possible effects of multiple successive change sets modifying the same object. Locking prevents more than one change set from modifying an object at the same time. After a change set is closed and its locks are released, however, another change set can modify previously locked objects.
- Coordinate the activities of all users and all change sets to ensure that you are testing exactly what will be exported and imported. For example, another change set can modify a pricing to which a charge in your change set has an association. Such a change doesn't cause a referential integrity problem; the pricing still exists; but may cause unexpected results during rating. Ideally, when you are testing a change set or group of change sets, there is no other development activity that might affect the tests.

Planning the Export Process

When a change set or group of change sets is complete, you export it to a file that can be imported into the target system.

Follow these guidelines for exporting change sets:

- Before you export, make sure that you understand any dependencies between change sets.

  Some dependencies are determined by locking rules. For example, if a new reusable object is introduced in one change set, that change set must be exported, imported, and closed before another change set can refer to the new object.

  Other dependencies are based on content. For example, if you use a change set to modify an existing calendar, you should make sure to export and import that change set before any change sets that rely on the modification.
- To ensure that change sets are exported and imported in the proper order, you can include multiple change sets in the same file and specify the sequence in which they are processed.
- Export change sets only when you are sure that they are complete and ready to be imported. There is no point in having incomplete export files in your system. They serve no purpose and there is some risk that they might be imported accidentally.

Managing Change Set Files

Change set files contain sensitive data, so you should manage them carefully. Three tasks are particularly vital:

- **Ensuring file security.** Once a file has been exported, it should be tracked carefully to ensure that it is not lost or corrupted. You should make sure there is no confusion about file names, which files are waiting to be imported, and similar matters.
- **Keeping track of file versions.** It's possible to have more than one version of a change set file. For example, if you export a file and then discover a problem with a change set in the file, you may need to make corrections and export the changes again. You should be very careful to keep track of the file versions to ensure that you are importing the correct one.
- **Keeping track of the file sequence.** In some cases, files must be exported and imported in a particular order. For example, if you modify a calendar in one change set, you should

export and import that change set before other change sets that rely on the modified calendar.

Depending on the complexity of your data model, a version control system may be useful for managing change set files.

You can specify the default locations to which change set files are exported and from which they are imported. For example, you can choose to export files to a location known to your version control system.

Planning the Import Process

Importing data into your production system is obviously a critical task. The Pricing Center import process checks every change to ensure that it is valid. If there are any errors during importation, the entire file is rejected, no changes are made to the target data, and the file is moved to an error directory.

In addition, you can take these steps to ensure that data is imported successfully:

- **Import files in the correct order.** If there are content dependencies among change set files, make sure to import them in the required order. For example, a change set may include a modification that another change relies on. The locking rules and other safeguards prevent data integrity errors such as references to non-existent objects, but you must keep track of dependencies based on business content.
- **Ensure that files reflect final data.** You should check that you are importing files that contain the final versions of your pricing data. If you accidentally import a file that is incomplete, you have to remove or modify the data manually; importing a file cannot be reversed. Careful planning and file management will prevent these problems.

Coordinating Real-Time Rating Data Migration and Pipeline Data Migration

BRM packages contain a mixture of real-time and pipeline data. For example, when you define charge offers, you can map some events to real-time charges and other events to pipeline charges.

Real-time and pipeline pricing data are exported and imported separately using different procedures. This document covers data migration for pipeline pricing data only.

You must manually coordinate the real-time and pipeline migration procedures. For example, if you added new pipeline pricing data associated with rating a new charge offer, you must migrate both the new charge offer (real-time data) and the new pricing data (pipeline data).

Configuring Pipeline Manager Data Migration Features

There are a number of configuration task for Pipeline Manager data migration that you accomplish outside the Pricing Center application, including enabling data migration, copying data to ensure that the development and production systems are identical, and optionally customizing the change set life cycle.

Enabling Data Migration in Pricing Center

The Pipeline Manager data migration features are optional. They are visible only if you enable them. You can enable the ability to import change set files without enabling the full set of data migration features. Import-only data migration is useful for production systems where you want to reduce the risk of accidental data changes.

You normally enable data migration during the Pricing Center installation process. You can also enable data migration after Pricing Center has been installed by making a change to the Pricing Center configuration file.

> **Note**
>
> If you enable data migration for one instance of Pricing Center, you should also enable data migration for all other instances of Pricing Center that have access to the same test and production databases. Instances of Pricing Center without data migration enabled can make changes outside the scope of change sets, thereby causing inconsistencies in the data.

To enable data migration after installation:

1. Exit Pricing Center.
2. Open the Pricing Center configuration file (**custom.properties**) in a text editor. This file is located in the **\lib** subdirectory of the installation directory, normally **C:\Program Files\Portal\PricingCenter**.
3. Change the value for the **pipeline.datamigration** parameter to **2** (for full data migration functionality) or **1** (for import capability only).
4. Save the file.
5. Start Pricing Center.

Copying Production Data to the Development System

When you set up your development environment, you start out with an exact duplicate of the production database.

> **Note**
>
> The test and production databases must be completely identical, including sequence IDs, for data migration to work reliably.

Use the replication tools provided with your database to copy the production database. See your system's documentation for instructions.

Customizing Change Set States

You can customize change set states to define a workflow for your projects. Your need for this customization depends on the complexity of your work. If there are only one or two change sets in progress at any one time and they contain simple changes, it may not be necessary to customize. On the other hand, if you have a team of planners working on multiple change sets in varying states of completion, you should customize the life cycle to reflect your process. See "Understanding the Change Set Life Cycle".

To customize change set states, you modify the **state.config** file, then load it by using the **stateconfigtool** utility. The **state.config** file lists each valid state transition. In other words, it defines all the states that it is valid to move to from any given state. It also lists whether the

transition is manual (accomplished by the user in the Change Set State dialog box) or automatic (accomplished by the BRM).

These are the contents of the default **state.config** file, which defines the standard change set life cycle:

```
# State Transition for Changeset
# currentState,nextState,Action

inprogress,readytoexport,manual
readytoexport,exported,automatic
readytoexport,inprogress,manual
exported,inprogress,manual
exported,closed,manual
```

> **Note**
>
> You can define additional states, but you cannot delete any default states. The custom states you define must come between In Progress and Ready to Export. All customized states must have a manual transition.

For example, the following file defines a new state called Testing. You can switch to Testing from In Progress and can switch from Testing to Ready to Export or back to In Progress. Note that you can't switch from Ready to Export back to Testing.

```
# State Transition for Changeset
# currentState,nextState,Action

inprogress,readytoexport,manual
inprogress,testing,manual
testing,readytoexport,manual
testing,inprogress,manual
readytoexport,exported,automatic
readytoexport,inprogress,manual
exported,inprogress,manual
exported,closed,manual
```

You load the change set configuration into the database by using the **stateconfigtool** utility. When you run this utility, you specify the file name, the database type, the host, the port number, the database instance, a login user name, and a login password.

To customize change set states:

1. Use a text editor to open the **state.config** file located in the Pricing Center directory, typically **Program Files\Portal Software\Pricing Center**.
2. Add new states, being careful to account for all the transitions necessary.

> **Note**
>
> Don't make any changes to the default entries in the **state.config** file. Doing so will cause an error when you load the file.

3. Save the file under a new name. Saving the file under a different name preserves the original file in case you want to return to the default configuration.
4. From the *Pipeline_home***tools\StateConfigTool** directory, run the **stateconfigtool** utility to load the file. *Pipeline_home* is the directory where you installed Pipeline Manager. Use this syntax:

```
stateconfigtool -f file_name -d database_type -h host -n port -i database_id
```

For example:

```
stateconfigtool -f Pipeline_home/tools/StateConfigTool/state.config -d oracle -h
sample_host -n 1521 -i pindb
```

Exporting and Importing Change Sets by Using the loadchangesets Utility

Under certain circumstances, importing and exporting change sets by using Pricing Center may be inconvenient or undesirable. For example, you may prefer not to allow Pricing Center access to your production system to prevent unauthorized or accidental changes to your production pricing data.

In these cases, you can use the **loadchangesets** command-line utility to export change sets from your test environment and import them into your production database.

> **Note**
>
> Exporting and importing change sets by using the **loadchangesets** utility changes only the final stages of the entire pipeline pricing data migration process. You still use Pricing Center to create and manage change sets.

The general process for exporting and importing change sets is similar whether you use Pricing Center or **loadchangesets**.

The **loadchangesets** utility has two modes: interactive and non-interactive. They both enable you to import and export change sets, but work somewhat differently. See "Working in Interactive and Non-Interactive Mode".

Unlike Pricing Center, where you can choose which change sets to export, **loadchangesets** exports all the change sets that are in Ready to Export state. You should therefore be careful to monitor the life cycles of your change sets to ensure that you are exporting all the change sets you want and none that you don't want.

Specifying BRM Servers for the loadchangesets Utility

Before you can use the **loadchangesets** utility, you must specify the BRM servers to export from and import to. You enter the host names (or IP addresses) and port numbers in a configuration file.

To specify BRM servers for import and export:

1. Exit Pricing Center if necessary.

2. Open the Pricing Center configuration file (**custom.properties**) in a text editor. This file is located in the **\lib** subdirectory of the installation directory, normally **C:\Program Files\Portal\PricingCenter**.
3. To specify the server from which to export pricing data, enter the host name (or IP address) and port number in the **pipeline.datamigration.utility.export.environment.host** and **pipeline.datamigration.utility.export.environment.port** entries.

   For example, to export from TestPricingServer, port number 11960, enter the following:

```
pipeline.datamigration.utility.export.environment.host=TestPricingServer
pipeline.datamigration.utility.export.environment.port=11960
```

4. To specify the server to which to import pricing data, enter the host name (or IP address) and port number in the **pipeline.datamigration.utility.import.environment.host** and **pipeline.datamigration.utility.import.environment.port** entries.

   For example, to import to ProductionPricingServer, port number 56968, enter the following:

```
pipeline.datamigration.utility.import.environment.host=ProductionPricingServer
pipeline.datamigration.utility.import.environment.port=56968
```

5. Save the file.

Working in Interactive and Non-Interactive Mode

You can use the **loadchangesets** utility in interactive or non-interactive mode:

- In interactive mode, you issue a command for each step in the process of importing or exporting. After you enter interactive mode, the prompt changes to an angle bracket and commands are single words for performing particular actions. You can view a list of the change sets that will be imported and exported
- In non-interactive mode, you use commands that batch several related parts of the import or export process. You must enter a full command, including the utility name for each set of actions.

Exporting and Importing Change Sets in Interactive Mode

The following procedures describe exporting and importing change sets by using **loadchangesets** in interactive mode.

To export and import change sets by using **loadchangesets** in interactive mode:

1. Make sure the change sets that you want to export are complete and that they, and no others, are in Ready to Export state.
2. Go to the **/lib** directory in the Pricing Center installation directory.
3. To switch to interactive mode, enter the following command:

   ```
   loadchangesets -i
   ```

   The prompt changes to **==>.**

4. To load the change set data from the export database into memory, enter the following command:

   ```
   export
   ```

5. To write the change set data from memory to a change set file, enter the following command. The file name must include the .**exp** file name extension.

   ```
   write change_set_file
   ```

   The change sets are exported to the specified file in the **/export/done** subdirectory in the Pricing Center installation directory. This directory is created automatically if it doesn't exist when you run the utility.

6. Move the change set file that you created into the **/import** subdirectory in the PricingCenter install directory.
7. To read the change set data from the change set file into memory, enter the following command. The file name must include the **.exp** extension.

   ```
   read change_set_file
   ```

8. (Optional) Enter the following command to view the names of the change sets you loaded into memory:

   ```
   list
   ```

9. To load the change set data from memory into the import database, enter the following command:

   ```
   import
   ```

Exporting and Importing Change Sets in Non-Interactive Mode

The following procedures describe exporting and importing change sets by using **loadchangesets** in non-interactive mode.

To export change sets by using **loadchangesets** in non-interactive mode:

1. Make sure the change sets that you want to export are complete and that they, and no others, are in Ready to Export state.
2. Go to the **/lib** directory in the Pricing Center installation directory.
3. Enter the following command, where *change_set_file* is the name of the file into which you want to export the change sets. The file name must include the .**exp** file name extension.

   ```
   loadchangesets -fx change_set_file
   ```

   The change sets are exported to the specified file in the **/export/done** subdirectory in the Pricing Center installation directory. This directory is created automatically if it doesn't exist when you run the utility.

To import change sets by using **loadchangesets** in non-interactive mode:

1. Move the change set file that you want to import into the **/import** subdirectory in the PricingCenter install directory.

2. Go to the **/lib** directory in the Pricing Center installation directory.
3. Enter the following command, where *change_set_file* is the name of the file that you want to import. The file name must include the **.exp** file name extension.

   ```
   loadchangesets -fi change_set_file
   ```

9

Transferring Data Between Pipeline Manager Databases

Learn how to transfer data from one Pipeline Manager database to another, such as from a test database to a production database, by using the Oracle Communications Billing and Revenue Management (BRM) **LoadIfwConfig** utility.

Topics in this document:

About Transferring Data

You transfer data by extracting data from a *source* Pipeline Manager database and then loading the data into a *destination* Pipeline Manager database. You specify which data to extract at the command line or by using an XML file. The **LoadIfwConfig** utility extracts the specified data from the source database to an output XML file. The utility can then load the output XML file directly into the destination database.

About Specifying the Data to Extract

You can specify to extract:

- All data in the Pipeline Manager database. You specify to extract all data by using only a utility command. For instructions, see "Extracting a Subset of Database Objects with LoadIfwConfig".
- All Pipeline Manager data that has been modified after a specified date and time. You specify to extract data based on the modification date and time by using utility commands. For instructions, see "Extracting All Database Objects Modified after a Specific Time".
- A subset of the data, based on the objects' attributes and modification time. You specify to extract a subset of the data by using an input XML file with the **LoadIfwConfig** utility. See "About Creating an Input XML File to Extract Data".

About Creating an Input XML File to Extract Data

If you are extracting a subset of data, you must create an input XML file that specifies the table from which to extract the data and, optionally, the criteria that the data must meet. The criteria consists of fields and their required values. For example, you can specify to extract from a specific table only those objects that have the SAMPLE field set to 100. The utility would then extract all objects with a SAMPLE field set to 100 as well as any child objects and any dependent objects. See "About Specifying to Extract Child and Dependent Objects".

When the input XML file specifies a table only, the utility extracts objects from the entire table as well as from any child and dependent objects. When the input XML file specifies a table and required field values, the utility extracts from the table only those objects that contain the matching field values plus any child and dependent objects.

The syntax for the input XML file is shown below:

```
<RecordSet>
  <TableName [FieldName1 ="Value1"] [FieldName2 ="Value2"] .../>
</RecordSet>
```

where:

- *TableName* is the name of the table from which to extract objects.
- *FieldNameX* is the name of the table field that must match the specified value. You can list multiple field-value pairs.
- *ValueX* specifies the required field value. To be able to list multiple values for each field, see "About Using Regular Expressions when Specifying the Data to Extract".

For example, the following input XML file specifies to retrieve all objects from the IFW_RUM table that match all of the criteria below:

- Have their NAME field set to **Duration**.
- Have their RUM field set to **DUR**.
- Have their TYPE field set to **D**.

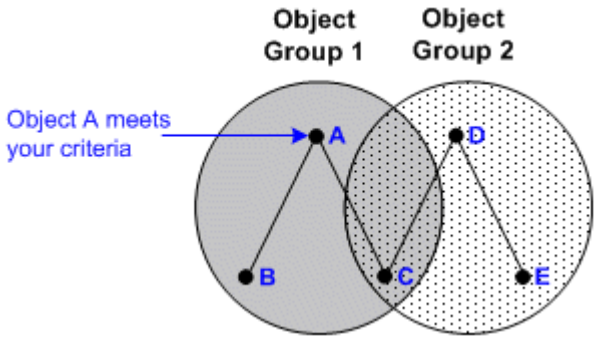
```
<RecordSet>
   <IFW_RUM NAME ="Duration" RUM ="DUR" TYPE ="D" />
</RecordSet>
```

About Specifying to Extract Child and Dependent Objects

When extracting a subset of data, the **LoadIfwConfig** utility automatically extracts all objects that meet your specified criteria as well as any related child objects. For example, when the utility extracts a charge object, the utility automatically extracts all child objects of the charge, such as the charge version and charge configuration objects.

If one of the child objects is also related to other objects, you can configure the utility to extract the other objects as well.

Figure 9-1 Object Relationships and Dependencies



For example, in Figure 9-1 above, object A meets your specified criteria. Because object A is a parent object, the utility automatically extracts object A and its two child objects, B and C. Because object C is also related to object D, extracting only the objects in object group 1 would break the relationships in object group 2.

To prevent this from happening, you can configure the utility to extract object D whenever object C is extracted. You do this by making object D a dependent of object C. In this case, whenever object C is extracted, the utility would also extract:

- Object D, because it is a dependent of object C.
- Object E, because it is a child of object D.

The utility determines the object dependencies by using the *Pipeline_home***/tools/XmlLoader/LoadIfwConfig.xsd** file, where *Pipeline_home* is the directory where you installed Pipeline Manager. You customize the dependencies by using the *Pipeline_home***/tools/XmlLoader/CustomConfig.xml** file.

> **Note**
>
> The settings in the **CustomConfig.xml** file override the settings in the **LoadIfwConfig.xsd** file.

The syntax for the **CustomConfig.xml** file is shown below:

```
<TableName AddDependingTable="AddTable" AddDependingTableMapping="Field1:Field2"/>
<TableName DependingTableNames="DependingTable"/>
```

where:

- *TableName* specifies the table that has dependent objects in another table.
- *AddTable* specifies the dependent table. The utility extracts data using the provided mapping whenever data from *TableName* is extracted.
- *Field1* is the field from *TableName* that is related to a field in *AddTable*.
- *Field2* is the relating field in *AddTable*. The utility extracts any objects from *AddTable* that have matching values whenever data from *TableName* is extracted.
- *DependingTable* specifies the list of dependent tables. Data from these tables must be extracted whenever data is extracted from *TableName*. You can list multiple table names by using the pipe symbol (**|**) as a delimiter.

Sample **CustomConfig.xml** entries are shown below:

```
<IFW_RATEPLAN_CNF AddDependingTable="IFW_TIMEMODEL" AddDependingTableMapping="TIMEMODEL:TIMEMODEL"/>
<IFW_RATEPLAN_CNF DependingTableNames="IFW_TIMEMODEL|IFW_PRICEMODEL"/>
```

- The first line specifies that the utility must extract dependent data from IFW_TIMEMODEL, relating the TIMEMODEL field of IFW_RATEPLAN_CNF to the TIMEMODEL field of IFW_TIMEMODEL.
- The second line specifies to extract dependent data from the IFW_TIMEMODEL and IFW_PRICEMODEL tables whenever data is extracted from IFW_RATEPLAN_CNF.

About Using Regular Expressions when Specifying the Data to Extract

By default, you cannot use regular expressions when specifying the data to extract. This means that the input XML file must include a separate line for each required field value, which impacts performance because multiple entries generate multiple SQL queries to the database. For example, to retrieve objects from the IFW_RATEPLAN_CNF table that have an IMPACT_CATEGORY value of FRANCE or SPAIN, the input XML file would contain these lines:

```
<IFW_RATEPLAN_CNF IMPACT_CATEGORY="FRANCE"/>
<IFW_RATEPLAN_CNF IMPACT_CATEGORY="SPAIN"/>
```

You can configure the utility to accept the following regular expressions when searching defined fields:

- The asterisk (*****) symbol for wildcard searches.
- The pipe (**|**) symbol for the logical OR operation.

You define which fields support regular expressions by configuring the **RegExFields** entry in the *Pipeline_home***/tools/XmlLoader/CustomConfig.xml** file.

The syntax for the **RegExFields** entry is shown below:

```
<TableName RegExFields="FieldName"/>
```

where:

- *TableName* is the name of the table that contains the specified field.
- *FieldName* is the name of the field that can be searched with regular expressions. You can list multiple fields by using the pipe symbol (**|**) as a delimiter.

For example, the following entry specifies that you can use regular expressions when searching for values in the IMPACT_CATEGORY field of the IFW_RATEPLAN_CNF table:

```
<IFW_RATEPLAN_CNF RegExFields="IMPACT_CATEGORY"/>
```

For the above example, you could retrieve records that have their IMPACT_CATEGORY field set to FRANCE or SPAIN by using this input XML entry:

```
<IFW_RATEPLAN_CNF IMPACT_CATEGORY="FRANCE|SPAIN"/>
```

About the LoadIfwConfig Error Messages

The **LoadIfwConfig** utility logs information about any errors it encounters to the log file you specified in the **ProcessLog** section of the **LoadIfwConfig.reg** registry file. Table 9-1 describes the utility's error messages.

> **Note**
>
> The utility will pass through any error messages thrown by the Xerces SAX parser and the Oracle database. For information about these error messages, see the appropriate vendor's documentation.

Table 9-1 LoadIfwConfig Error Messages utility

Error message	Description
ERROR: Connection is not Valid	The utility failed to connect to the database.
ERROR: DataBaseStatus is not Valid	The database credentials or database connect string is not correct.
ERROR: Couldn't get next sequence: *SQLString*	The utility could not generate a sequence number to insert into the database.
ERROR: during insert: *SQLString*	The utility encountered an error during the insert operation.
ERROR: during update: *SQLString*	The utility encountered an error during the update operation.
ERROR: during delete: *SQLString*	The utility encountered an error during the delete operation.
Exception occurred while running the *SQLString*	The utility encountered an error while running other SQL statements.
Exception from DB: *ErrorMessage*	The error message that was thrown by the Oracle database.
File Not parsed properly or Braces not matched properly	The utility's registry file (**LoadIfwConfig.reg**) contains incorrect entries or the entries are not framed correctly.
DependentFields structure provided in XML for depending table not proper	The **AddDependingTableMapping** entry in the **CustomConfig.xml** file is set incorrectly.
No rows for deletion because rows present in Child table	The utility could not delete the requested row because the row's associated child records still contain data.
Could not find valid Translation for: Table: *TableName* Referred Table: *TableName* Field: *FieldName* CODE Field Value: *FieldValue*	The required table dependencies are not provided in the input XML file.
FATAL ERROR at file: *FileName* line: *LineNumber* char: *Position* Message: *Message*	The input XML file contains mistakes, such as unparseable characters.

Using LoadIfwConfig to Transfer Data Between Databases

To transfer data from one Pipeline Manager database to another, perform these steps:

1. Connect **LoadIfwConfig** to the source Pipeline Manager database. See "Connecting LoadIfwConfig to the Pipeline Manager Database".
2. (Optional) Specify the regular expressions and table dependencies that are supported on the source Pipeline Manager system. See "Customizing the Regular Expression and Dependent Table Settings".
3. Extract data from the source Pipeline Manager database. See "Extracting Data from a Pipeline Manager Database".

4. Connect **LoadIfwConfig** to the destination Pipeline Manager database. See "Connecting LoadIfwConfig to the Pipeline Manager Database".
5. Load data into the destination Pipeline Manager database. See "Loading Data into Pipeline Manager Databases".

Connecting LoadIfwConfig to the Pipeline Manager Database

You connect the **LoadIfwConfig** utility to the Pipeline Manager database by using the **LoadIfwConfig.reg** registry file. You can edit this registry file manually, but it is also updated by the **pin_setup** utility during the **LoadIfwConfig** installation process.

To connect **LoadIfwConfig** to the Pipeline Manager database:

1. Open the *Pipeline_home*/**tools/XmlLoader/LoadIfwConfig.reg** registry file in a text editor.
2. **Edit the registry entries to match your system environment. In particular, pay attention to these entries:**
   - In the **XMLCustomizationFile** entry, specify the location of the optional customization XML file. **LoadIfwConfig** contains a sample customization file (**CustomConfig.xml**) and a schema file (**CustomConfig.xsd**) to which the XML should conform.
   - In the **LoadDataFromDB** entry, specify whether to increase performance by loading the sequence-to-code translation information into memory. If this entry is missing or blank, it defaults to **False**.
   - (Optional) In the **RowFetchSize** entry, specify the number of database rows to retrieve on each trip to the database. Increasing the number of rows can reduce the required number of database fetches and increase the utility's performance. The default is **100**.

   The other entries are standard logging and connection registry entries.
3. Save and close the file.

A sample **LoadIfwConfig.reg** file is shown below:

```
LoadIfwConfig
{
  LogMessageTable
  {
    MessageFilePath   = ./
    MessageFileSuffix = .msg
  }

  ProcessLog
  {
    FilePath          = ./log
    FileName          = LoadIfwConfig
    FileSuffix        = .log
    WriteMessageKey   = True
  }

  DataPool
  {
    Database
    {
      ModuleName = DbInterface
      Module
      {
        # Common
        DatabaseName = DatabaseName
        UserName     = UserName
```

```
        PassWord     = EncryptedPassword
        AccessLib    = oci231
      }
    }
  }
  XMLCustomizationFile = CustomConfig.xml
  LoadDataFromDB = True
  RowFetchSize = 200
}
```

Customizing the Regular Expression and Dependent Table Settings

If you want the **LoadIfwConfig** utility to support regular expressions for any fields, or if you want to add table dependencies for any objects, you must modify the **CustomConfig.xml** file.

To customize the regular expression and dependent table settings, perform these steps on the source Pipeline Manager:

1. Open the *Pipeline_home*/**tools/XmlLoader/CustomConfig.xml** file in a text editor.
2. Specify the table fields that support regular expressions by using the **RegExFields** entry. For example, the following entry specifies that the CODE and NAME fields in the IFW_RATEPLAN table support regular expressions:

   ```
   <IFW_RATEPLAN RegExFields="CODE|NAME"/>
   ```

   For more information, see "About Using Regular Expressions when Specifying the Data to Extract".
3. Specify any object dependencies by using the **AddDependingTable** and **AddDependingTableMapping** entries. For example, the following entry specifies to retrieve dependent data from IFW_TIMEMODEL, relating the TIMEMODEL field of IFW_RATEPLAN_CNF to the TIMEMODEL field of IFW_TIMEMODEL:

   ```
   <IFW_RATEPLAN_CNF AddDependingTable="IFW_TIMEMODEL"
   AddDependingTableMapping="TIMEMODEL:TIMEMODEL"/>
   ```

   For more information, see "About Specifying to Extract Child and Dependent Objects".
4. Specify any table dependencies by using the **DependingTableNames** entry. For example, the following entry specifies to extract dependent data from the IFW_TIMEMODEL table whenever data from IFW_RATEPLAN_CNF is extracted:

   ```
   <IFW_RATEPLAN_CNF DependingTableNames="IFW_TIMEMODEL"/>
   ```

   For more information, see "About Specifying to Extract Child and Dependent Objects".
5. Save and close the file.

Extracting Data from a Pipeline Manager Database

You can use the **LoadIfwConfig** utility to extract:

- All Pipeline Manager database objects. See "Extracting a Subset of Database Objects with LoadIfwConfig".
- All Pipeline Manager database objects modified after a certain date and time. See "Extracting All Database Objects with LoadIfwConfig".

- A subset of Pipeline Manager database objects. See "Extracting All Database Objects Modified after a Specific Time".

Extracting All Database Objects with LoadIfwConfig

The utility extracts all database objects by iterating through each table in the schema in order, selecting all of the rows, and dumping them directly into an XML file.

To extract all Pipeline Manager database objects:

1. Go to the *Pipeline_home*/**tools/XmlLoader** directory.
2. Enter this command:

   Interactive mode

   ```
   LoadIfwConfig
   write [OutputFile]
   retrieve_all
   ```

   where *OutputFile* specifies the name and location of the file to which to extract the pipeline data. By default, the utility writes the output to a file named **default.out** in the current directory.

   Non-interactive mode

   ```
   LoadIfwConfig  -rall [-o OutputFile]
   ```

   where *OutputFile* specifies the name and location of the file to which to extract the pipeline data. By default, the utility writes the output to a file named **default.out** in the current directory.

   > **Note**
   >
   > For more information about the utility's syntax, see "LoadIfwConfig" in *BRM Pipeline Manager Reference*.

The utility writes the extracted objects to the output file in XML format. You can now load the output XML file directly into the destination Pipeline Manager database.

Extracting All Database Objects Modified after a Specific Time

To extract all Pipeline Manager database objects that have been modified after a specified date and time:

1. Go to the *Pipeline_home*/**tools/XmlLoader** directory.
2. Enter this command:

   Interactive mode

   ```
   LoadIfwConfig
   write [OutputFile]
   retrieve_all [-t Modifydate]
   ```

   where:

   - *OutputFile* specifies the name and location of the file to which to extract the pipeline data. By default, the utility writes the output to a file named **default.out** in the current directory.

- *Modifydate* specifies the timestamp after which to retrieve pricing objects. Enter the timestamp in the ISO-8601 format: *YYYY*-*MM*-*DD*T*hh*:*mm*:*ss* or *YYYY*-*MM*-*DD*, with the server time zone as the default. For example:

  ```
  2025-07-16T19:20:30
  ```

Non-interactive mode

```
LoadIfwConfig  -rall [-t Modifydate] [-o OutputFile]
```

where:

- *Modifydate* specifies the timestamp after which to retrieve pricing objects. Enter the timestamp in the ISO-8601 format: *YYYY*-*MM*-*DD*T*hh*:*mm*:*ss* or *YYYY*-*MM*-*DD*, with the server time zone as the default. For example:

  ```
  2025-07-16T19:20:30
  ```

- *OutputFile* specifies the name and location of the file to which to extract the pipeline data. By default, the utility writes the output to a file named **default.out** in the current directory.

> **Note**
>
> For more information about the utility's syntax, see "LoadIfwConfig" in *BRM Pipeline Manager Reference*.

The utility writes the extracted objects to the output file in XML format. You can now load the output XML file directly into the destination Pipeline Manager database.

Extracting a Subset of Database Objects with LoadIfwConfig

To extract a subset of Pipeline Manager database objects:

1. Create an XML file that lists the objects to extract. The file specifies the table from which to extract the objects and, optionally, the criteria that the objects must meet. You can use the sample input XML file (*Pipeline_home*/**tools/XmlLoader/samples.xml**) as a starting point.

   See "About Specifying the Data to Extract" for more information.

2. Go to the *Pipeline_home*/**tools/XmlLoader** directory.

3. Enter the following command:

   Interactive mode

   ```
   LoadIfwConfig
   [-nodep]
   read InputFile
   fetch [-t Modifidate]
   write [OutputFile]
   ```

   where:

   - **-nodep** suppresses the object dependency relationships you configured in the *Pipeline_home*/**tools/XmlLoader/CustomConfig.xml** file. The utility extracts only the objects specified in *InputFile* and ignores all dependent objects.

> **Note**
>
> To suppress object dependency relationships in interactive mode, the utility session must start with the **-nodep** parameter.

- *InputFile* specifies the name and location of the file that lists which objects to retrieve. This is the file that you created in step 1.
- *Modifydate* specifies to retrieve objects that were modified after the specified timestamp. Enter the timestamp in the ISO-8601 format: *YYYY*-*MM*-*DD*T*hh*:*mm*:*ss* or *YYYY*-*MM*-*DD*, with the server time zone as the default. For example:

  ```
  2025-07-16T19:20:30
  ```

- *OutputFile* specifies the output file to which the Pipeline Manager data is extracted. By default, the utility writes the output to a file named **default.out** in the current directory.

Non-interactive mode

```
LoadIfwConfig  -i InputFile  -r [-nodep] [-t Modifydate] [-o OutputFile]
```

where:

- *InputFile* specifies the name and location of the file that lists which objects to retrieve. This is the file that you created in step 1.
- **-nodep** suppresses the object dependency relationships you configured in the *Pipeline_home*/**tools/XmlLoader/CustomConfig.xml** file. The utility extracts only the objects specified in *InputFile* and ignores all dependent objects.
- *Modifydate* specifies to retrieve objects that were modified after the specified timestamp. Enter the timestamp in the ISO-8601 format: *YYYY-MM-DDThh*:*mm*:*ss* or *YYYY-MM-DD*, with the server time zone as the default. For example:

  ```
  2025-07-16T19:20:30
  ```

- *OutputFile* specifies the output file to which the Pipeline Manager data is extracted. By default, the utility writes the output to a file named **default.out** in the current directory.

The utility writes the extracted objects to the output file in XML format. You can now load the output XML file directly into the destination Pipeline Manager database.

Loading Data into Pipeline Manager Databases

You load data into the destination Pipeline Manager database by using the **LoadIfwConfig** utility's update option or insert option.

- Update option: The utility verifies whether the data provided in the input XML file already exists in the database. If the data already exists, the utility updates the database record with the new information. If the data does not exist, the utility inserts the data into the database. To update data see, "Updating the Pipeline Manager Database".
- Insert option: The utility inserts data into the database without verifying whether the data already exists. To insert data, see "Inserting Data into the Pipeline Manager Database".

Updating the Pipeline Manager Database

To update the data in the Pipeline Manager database:

1. Go to the *Pipeline_home*/**tools/XmlLoader** directory.

2. Enter this command:

   Interactive mode

   ```
   LoadIfwConfig
   read InputFile
   update
   commit
   ```

   where *InputFile* specifies the name and location of the XML file that contains the extracted pipeline data from the source database. This is the output file you generated in "Extracting Data from a Pipeline Manager Database".

   Non-interactive mode

   ```
   LoadIfwConfig  -u  -c  -i InputFile
   ```

   where *InputFile* specifies the name and location of the XML file that contains the extracted pipeline data from the source database. This is the output file you generated in "Extracting Data from a Pipeline Manager Database".

   > **Note**
   >
   > For more information about the utility's syntax, see "LoadIfwConfig" in *BRM Pipeline Manager Reference*.

The utility loads the data from the XML file into the Pipeline Manager database and commits the data. If there is a failure, the utility rolls back the data and displays an error message.

Inserting Data into the Pipeline Manager Database

To insert data into a Pipeline Manager database:

1. Go to the *Pipeline_home*/**tools/XmlLoader** directory.
2. Enter this command:

   Interactive mode

   ```
   LoadIfwConfig
   read InputFile
   insert
   commit
   ```

   where *InputFile* specifies the name and location of the XML file that contains the extracted data from the source database. This is the output file you generated in "Extracting Data from a Pipeline Manager Database".

   Non-interactive mode

   ```
   LoadIfwConfig   -I  -c  -i InputFile
   ```

   where *InputFile* specifies the name and location of the XML file that contains the extracted data from the source database. This is the output file you generated in "Extracting Data from a Pipeline Manager Database".

> **Note**
>
> For more information about the utility's syntax, see "LoadIfwConfig" in *BRM Pipeline Manager Reference*.

The utility loads the data from the XML file into the Pipeline Manager database and commits the data. If there is a failure, the utility rolls back the data and displays an error message.

Deleting Data from a Pipeline Manager Database

> **Note**
>
> Make sure the utility is connected to the Pipeline Manager database. See "Connecting LoadIfwConfig to the Pipeline Manager Database".

To delete data from a Pipeline Manager database:

1. Create an XML file that specifies the data to delete. The file includes the table from which to delete the objects and, optionally, the criteria that the objects must meet. For information, see "About Specifying the Data to Extract".
2. Test the XML file by running the **LoadIfwConfig** utility with the **-r** or **fetch** parameter. Verify that the output file lists the correct objects to delete. See "Extracting a Subset of Database Objects with LoadIfwConfig" for more information.
3. Go to the *Pipeline_home*/**tools/XmlLoader** directory.
4. Enter the following command:

   Interactive mode

   ```
   LoadIfwConfig
   read InputFile
   delete
   ```

   where:

   - *InputFile* specifies the name and location of the file that lists the objects to delete. This is the file that you created in step 1.

   Non-interactive mode

   ```
   LoadIfwConfig  -p[f] -i InputFile
   ```

   where:

   - **f** turns off the delete confirmation message.
   - *InputFile* specifies the name and location of the file that lists the objects to delete. This is the file that you created in step 1.

   > **Note**
   >
   > For more information about the utility's syntax, see "LoadIfwConfig" in *BRM Pipeline Manager Reference*.

The utility deletes the specified database objects.

10

Creating iScripts and iRules

Learn how to use the Application Programming Interface (API) to create custom Oracle Communications Billing and Revenue Management (BRM) iScript and iRules modules to process event detail records (EDRs) in the pipeline.

Topics in this document:

For information about the iScript functions, see "About iScript Functions".

About iScripts

iScript is a script language for analyzing and manipulating usage events. You can access any field within the usage structure and modify it. Using an iScript, you can implement and configure new guiding rules, mapping scenarios, discounting structures, interfaces to user-defined database objects and to reference data, and operating report files.

iScript is the Pipeline Manager script programming language that can be used to customize Pipeline Manager. It can be used for:

- Mapping operations that are not covered by standard modules.
- Adjusting values, such as normalizing phone numbers and IP addresses.
- Evaluating additional statistical data.
- Firing events.
- Performing splitting operations.

- Performing customer-specific pre- or post-rating or enrichment functionality.

You use iScripts to perform the same operation on every EDR. For example, if you have to enrich or normalize data in EDRs or if you have to evaluate EDRs before splitting them.

iScript makes it easier than C or C++ programming languages to implement custom functionality and business policies.

About iRules

An iRule consists of rule items that contain a number of conditions and a script to run when a condition is fulfilled. Only the script for the first valid rule item is run during the rule evaluation.

You create the scripts to be run by using pattern-matching evaluation (for example, wildcards and logical expressions) as well as normal comparison operators (for example, equal to, less than, and greater than).

You use iRules when a function should be run only on certain EDRs or under certain conditions. The rule-based engine offers an optimized method for defining logical conditions and extensive string comparisons. Therefore, iRules offer a faster way of implementing rule-based conditions than writing an iScript, which can include many "if-else" constructs.

You group rules and rule items into rule sets. A rule set defines the set of rules to be used by a particular iRule module. You specify the rule set in the FCT_IRule module startup registry.

> **Note**
>
> You can refer to only one EDR extension block in an iRule. For example, you cannot use DETAIL.ASS_GSMW_EXT and DETAIL.ASS_SUSPENSE_EXT extension blocks in the same iRule. Use an iScript instead to compare or evaluate fields from multiple EDR extension blocks.

Creating Rule Sets via Description Files

In addition to the database interface and the file interface, you can also use a description file to define a rule set. This is useful when you store the rule data in separate database tables or ASCII files. A rule set defined by a description file can only have one rule.

Descriptions for Data from an ASCII File

In the following example description file:

- The rule name is **ClassTypeMap**.
- The source data is stored in an ASCII file called **/data/data.txt**.
- The variables, represented by **${*N*}**, are replaced by values in the source data file.

```
RULE: ClassTypeMap
SOURCE: File
FILE: /data/data.txt
INIT_SCRIPT:
String code = edrString( DETAIL.SERVICE_CODE ) + edrString( DETAIL.SERVICE_CLASS );
CONDITION:
code =~ "${1}";
edrString( DETAIL.CALL_CLASS ) =~ "${2}";
edrString( DETAIL.CALL_TYPE ) =~ "${3}";
```

```
RESULT:
edrString( DETAIL.CALL_TYPE ) = "${4}";
edrString( DETAIL.CALL_CLASS ) = "${5}";
```

The following example source data file contains the values that replace the variables (represented by **${*N*}**) in the definition file.

- The source data file must contain at least the number of columns, separated by semicolons, as there are variables in the definition file.
- The column position corresponds to the variable number. For example, ${2} in the description file is replaced with the value in column 2 of the source data file.

```
CODE1;CC1*;CT*;NewCC1;NewCT
CODE2;CC2*;CT*;NewCC2;NewCT
CODE3;CC3*;CT*;NewCC3;NewCT
CODE4;CC4*;CT*;NewCC4;NewCT
CODE5;CC5*;CT*;NewCC5;NewCT
CODE6;CC6*;CT*;NewCC6;NewCT
CODE7;CC7*;CT*;NewCC7;NewCT
CODE8;CC8*;CT*;NewCC8;NewCT
```

Descriptions for Data from a Database Table

In the following example description file:

- The rule name is **CZT_MapRule**.
- The source data is retrieved from the **IFW_CLASSTYPEZONE_MAP** database table.
- The data is ordered first by the ZONEMODEL column and then by the RANK column.
- Each variable, represented by **${*COLUMN_NAME*}**, is replaced by the value in that database table column.

```
SOURCE: Database
RULE: CZT_MapRule
TABLE: IFW_CLASSTYPEZONE_MAP
ORDERBY: ZONEMODEL
ORDERBY: RANK
INIT_SCRIPT:
String code = edrString( DETAIL.SERVICE_CODE ) + edrString( DETAIL.SERVICE_CLASS );
CONDITION:
code =~ "${CODE}"
edrString( DETAIL.CALL_TYPE ) =~ "${CALLTYPE}";
edrString( DETAIL.CALL_CLASS ) =~ "${CALLCLASS}";
edrString( DETAIL.SERVICE_CODE ) =~ "${SERVICECODE}";
edrString( DETAIL.AOC_ZONE ) =~ "${ZONE_WS}";
edrString( DETAIL.CHARGED_ZONE ) =~ "${ZONE_RT}";
RESULT:
if ( length( "${NEW_CALLTYPE}" ) > 0 )
{
edrString( DETAIL.CALL_TYPE ) = "${NEW_CALLTYPE}";
}
if ( length( "${NEW_ZONE_RT}" ) > 0 )
{
edrString( DETAIL.CHARGED_ZONE ) = "${NEW_ZONE_RT}";
}
if ( length( "${NEW_ZONE_WS}" ) > 0 )
{
edrString( DETAIL.AOC_ZONE ) = "${NEW_ZONE_WS}";
}
```

Importing and Exporting Validation Rules

You use the Database Storage and Extraction Tool for Validation Rules to extract validation rules from the Pipeline Manager database and to import them in the Pipeline Manager database. Pipeline Manager uses these validation rules for the roaming incollect and outcollect processes.

This tool uses DBI and DBD drivers which are not part of the Pipeline Manager installation. You download these drivers from www.cpan.org and compile and install them separately.

The Database Storage and Extraction Tool for Validation Rules consists of the following scripts:

- The **db2irules.pl** script extracts validation rules from the Pipeline Manager database.
- The **irules2db.pl** script adds validation rules to the Pipeline Manager database.

About the Rule Set XML File

The Database Storage and Extraction Tool for Validation Rules extracts the validation rules from the database to a Rule Set XML file. When creating the XML file, it maps the Structured Query Language (SQL) tables from the database to the appropriate XML tag names. When you import rules and rule items in the database, the rank for each of these is determined by the order in which they appear in the Rule Set XML file. For example, the first rule row or rule item row found in the XML file is inserted into the database with a rank of 1, the second one is inserted with a rank of 2 and so forth.

Table 10-1 shows a possible mapping.

Table 10-1 Possible Mappings for Validation

Pipeline Manager Database Table	Pipeline Manager Database Table Column	XML Tag	Parent XML Tag
N/A	N/A	<RULESET_ROW>	NONE
IFW_RULESET	RULESET	<RULESET_RULESET>	<RULESET_ROW>
IFW_RULELSETLIST	RULESET	<RULESET_RULESET>	<RULESET_ROW>
IFW_RULESET	NAME	<RULESET_NAME>	<RULESET_ROW>
IFW_RULESET	DESCRIPTION	<RULESET_DESCRIPTION>	<RULESET_ROW>
N/A	N/A	<RULE_ROW>	<RULESET_ROW>
IFW_RULESETLIST	RULE	<RULE_RULE>	<RULE_ROW>
IFW_RULE	RULE	<RULE_RULE>	<RULE_ROW>
IFW_RULEITEM	RULE	<RULE_RULE>	<RULE_ROW>
IFW_RULE	NAME	<RULE_NAME>	<RULE_ROW>
IFW_RULESETLIST	NAME	<RULE_NAME>	<RULE_ROW>
IFW_RULE	INIT_SCRIPT	<RULE_INIT_SCRIPT>	<RULE_ROW>
N/A	N/A	<RULEITEM_ROW>	<RULE_ROW>
IFW_RULEITEM	NAME	<RULEITEM_NAME>	<RULEITEM_ROW>
IFW_RULEITEM	CONDITION	<RULEITEM_CONDITION>	<RULEITEM_ROW>
IFW_RULEITEM	RESULT	<RULEITEM_RESULT>	<RULEITEM_ROW>

About the db2irules.pl Script

You use the **db2irules.pl** script to extract rule sets from the Pipeline Manager database to the Rule Set XML file. You can extract any number of rule sets at the same time. If you extract multiple rule sets, each rule set is written to a separate Rule Set XML file.

When **db2irules.pl** extracts rows from the database, the text of each database row is checked for the following XML reserved characters:

- Left angle bracket (<)
- Right angle bracket (>)
- Ampersand (&)

The XML parser uses these characters to find the start tags, end tags, and any references. It replaces them with the following HTML and XML flags:

- <
- >
- &

These flags make the file XML compliant, so that any XML parser can parse this file successfully.

About the irules2db.pl Script

Use the **irules2db.pl** script to insert a rule set from Validation Rules XML file into the Pipeline Manager database. You can insert only one rule set at a time. To load multiple rule sets into the database, run this script separately for each rule.

When **irules2db.pl** imports rows to the database, each rule and rule set row of the XML file is checked for the following HTML and XML flags:

- <
- >
- &

The XML parser uses these characters to find the start tags, end tags, and any references. It replaces the m with the following XML-reserved characters:

- Left angle bracket (<)
- Right angle bracket (>)
- Ampersand (&)

If you specify an invalid file name for the rule set, the **irules2db.pl** script displays an error and terminates. If the file exists and can be opened, the script opens a transaction to the database and starts inserting the rule sets. If any of the rule sets specified already exists in the database, the **irules2db.pl** script reports an error, rolls back the transaction, and terminates. The error message contains information on which row in the XML file caused the error.

Updating a Rule Set

You can update rule sets that already exist in the database in two ways:

- Extract, update, and import a rule set

- Replace a rule set with an updated version

Extract, update, and import a rule set

To update a rule set that already exists in the database, export, update, and reimport the XML file:

1. Export the rule set to a Rule Set XML file by using the **db2irules.pl** script. For more information, see "About the db2irules.pl Script".

   Example:

   ```
   db2irules.pl -u dbi:Oracle:orcl /home/data/CIBER_val
   ```

2. Delete the rule set from the database. Specify the name of the rule set and set the **-d** parameter.

   Example:

   ```
   db2irules.pl -d dbi:Oracle:orcl /home/data/CIBER_val
   ```

   The script deletes all rules of a rule set recursively.

3. Update the rule set in the XML file. For more information, see "About the Rule Set XML File".

4. Import the rule set into the database.

   Example:

   ```
   irules2db.pl dbi:Oracle:orcl /home/data/CIBER_val_2002_07_01_17-15-39.xml
   ```

Replace a rule set with an updated version

To replace an existing rule set with an updated version that you didn't extract from the database, you can import the rule set and create a backup of the old rule set by using the **-f** parameter.

Example:

```
irules2db.pl -f dbi:Oracle:orcl /home/data/CIBER_val /home/data/backup
```

The **-f** parameter causes the **db2irules** extraction script to be invoked and extract the original file from the database before loading the new file. A backup file of the original rule set is stored in the specified backup location.

Supported iScript Data Types

iScript supports the following data types in Table 10-2.

Table 10-2 Supported iScript Data Types

Data Type	Description
Bool	For Boolean values TRUE and FALSE.
String	For 8-bit character strings of an unlimited length.
Long	For signed integer values in the interval from - 9223372036854775808 (2^63) to 9223372036854775807 (2^63-1).
Date	For date/time values in the interval from the 1901/01/01 00:00:00 to the 2037/02/05 00:00:00.
Decimal	For floating pointer to numbers with a precision of 26 digits.

Table 10-2 (Cont.) Supported iScript Data Types

Data Type	Description
File	As a handle for files.

> **Note**
>
> Based on the data types listed above, you can also use hash and arrays.

Supported iScript Constants

iScript supports constants described in this section.

Constants for Normalizing National Access Codes

Table 10-3 lists the iScript constants used to normalize national access codes.

Table 10-3 National Access Code Normalizing iScript Constants

Constant	Type	Value	Description
NORM_NAC	String	"0"	"National access code"
NORM_IAC	String	"00"	"International access code array"
NORM_IAC_STRING	String	"00"	"International access code string"
NORM_CC	String	NA	Country code array
NORM_CC_STRING	String	"49"	"Country code string"
NORM_MCC	String	"262"	"Mobile country code"
NORM_IAC_SIGN	String	"+"	"International country code sign"
NORM_NDC	String	"172"	"Network destination code"

Date Constants

Table 10-4 lists the Date constants.

Table 10-4 Date Constants

Constant	Type	Value	Description
MAX_DATE	Date	05.02.2037 23:59:59	Maximum value for the date. The date is stored as the number of seconds since 00:00:00. The MAX_DATE value is the last date that can be represented with a four-byte unsigned long, which is February 5, 2037. The internal representation is adjusted to the time zone of the system.

Table 10-4 (Cont.) Date Constants

Constant	Type	Value	Description
MIN_DATE	Date	01.01.1901 00:00:00	Minimum value for the date. The date is stored as the number of seconds since 00:00:00. The MIN_DATE value is January 1, 1901. The internal representation is adjusted to the time zone of the system.

Database Connection Constants

Table 10-5 describes the database connection constants.

Table 10-5 Database Connection Constants

Constant	Type	Value	Description
INVALID_CONNECTION	Long	-1	Invalid database connection handle
INVALID_RESULT	Long	-1	Invalid result handle
NO_MORE_RESULTS	Long	0	No more results
NO_MORE_ROWS	Long	0	No more rows
NEXT_ROW	Long	1	Next row to be retrieved from the database table
NEXT_RESULT	Long	1	Next result from db query

Decimal Constants

Table 10-6 lists the decimal constants.

Table 10-6 Decimal Constants

Constant	Type	Value	Description
INVALID_DECIMAL	Decimal	Decimal::initInvalidDecimal();	Invalid Decimal value

Decimal Rounding Constants

Table 10-7 lists the decimal rounding constants.

Table 10-7 Decimal Rounding Constants

Constant	Type	Value	Description
ROUND_PLAIN	Long	0	If the digit to the right of the specified decimal place is equal to or greater than 5, add one to the digit at the specified decimal place, then truncate any digits to the right. Otherwise, truncate all digits to the right of the specified decimal place.

Table 10-7 (Cont.) Decimal Rounding Constants

Constant	Type	Value	Description
ROUND_UP	Long	1	If the digits to the right of the specified decimal place are non-zero, add one to the digit at the specified decimal place and truncate any digits to the right.
ROUND_DOWN	Long	2	Truncate all digits to the right of the specified decimal place.
ROUND_BANKERS	Long	3	If incrementing the digit at the specified decimal place results in an even number, increment it and truncate the digits to its right. Otherwise, truncate the digits to the right of the specified decimal place.

EDR Container Content Constants

Table 10-8 lists the EDR container content constants.

Table 10-8 EDR Container Constants

Constant	Type	Value	Description
EDR_UNKNOWN_CONT	Long	1	Unknown EDR content type
EDR_HEADER	Long	2	EDR header record
EDR_DETAIL	Long	3	EDR detail record
EDR_TRAILER	Long	4	EDR trailer record
EDR_START	Long	5	Service container that tells pipeline starts
EDR_STOP	Long	6	Service container that tells pipeline shutting down
EDR_BEGIN	Long	7	Service container that tells EDR processing begins
EDR_END	Long	8	Ser vice container that tells EDR processing ends
EDR_BEGIN_TRANSACTION	Long	9	Service container that tells transaction begins
EDR_END_TRANSACTION	Long	10	Service container that tells transaction ends

EDR Container Characters Deletion Constants

Table 10-9 lists the EDR container content deletion constants.

Table 10-9 EDR Container Content Deletion Constants

Constant	Type	Value	Description
STRIP_LEADING	Long	True	Roguewave constant used to delete the special leading characters in an EDR string, for example, white spaces at the beginning of the EDR container.
STRIP_TRAILING	Long	True	Roguewave constant used to delete the special trailing characters in an EDR string, for example, white spaces at the end of the EDR container.
STRIP_BOTH	Long	True	Roguewave constant used to delete both the special leading and trailing characters in an EDR string, for example, white spaces at the beginning of the EDR container.

EDR Input State Constants

Table 10-10 lists the EDR input state constants.

Table 10-10 EDR Input State Constants

Constant	Type	Value	Description
EDR_INPUT_MISSING	Long	0	Not supplied in input data
EDR_INPUT_EMPTY	Long	1	Supplied with no value
EDR_INPUT_OTHER	Long	2	Other "uninteresting" input state

EDR Internal State Constants

Table 10-11 lists the EDR internal state constants.

Table 10-11 EDR Internal State Constants

Constant	Type	Value	Description
STATE_CLEARED	Long	0	EDR value is cleared
STATE_CONNECTED	long	1	EDR value is connected with a field from an input record
STATE_INITIALZED	long	2	EDR value is initialized
STATE_SET	long	3	EDR value is set
STATE_RESTORED	long	4	EDR value is restored
STATE_RESTOREDASSET	long	5	EDR value is restored as set
CONTAINER_HEADER	Long	2	Header record descriptor in container
CONTAINER_DETAIL	Long	3	Detail record descriptor in container
CONTAINER_TRAILER	Long	4	Trailer record descriptor in container
CONTAINER_UNKNOWN	Long	1	Unknown record type in container

POID Constants

Table 10-12 lists the Portal object ID (POID) constants.

Table 10-12 POID Constants

Constant	Type	Value	Description
NULL_PPOID	POID	BAS::Identifier(0, "", 0);	Null POID pointer
INVALID_PPOID	POID	BAS::Identifier	Invalid POID pointer

TAM Transaction Constants

Table 10-13 lists the TAM transaction constants.

Table 10-13 TAM Transaction Constants

Constant	Type	Value	Description
TAM_NORMAL	Long	0	Normal transaction
TAM_RECYCLE	Long	1	Recycle transaction
TAM_RECYCLE_TEST	Long	2	Recycle transaction for testing
TAM_UNKNOWN	Long	-1	Unknown transaction type

Supported Regular Expressions

iScript supports the regular expressions listed in Table 10-14.

Table 10-14 Regular Expressions Supported by iScript

Expression	Description
.	Matches any single character except the newline character **\n**.
****	Introduces metacharacters, and as part of the escape sequences. For example, **\n**, a newline character, and *****, is a literal asterisk. You can use it with three digits representing a number between 0 and 255 (ASCII code) to get the corresponding character. For example, **\065** is matched to the character **A**.
[ ]	A character class which matches any character within the brackets. If the first character is a circumflex (^), it changes the meaning to match any character except the ones within the brackets. A dash inside indicates a character range. For example, [0-9] means the same thing as [0123456789].
{ }	Indicates how many times to match the previous pattern when the pattern has one or two numbers. For example, A{1,3} matches one to three occurrences of the letter A.
*****	Matches zero or more copies of the preceding expression.
+	Matches one or more copies of the preceding expression. For example, [0-9]+ matches 1, 111, or 123456 but not an empty string.
?	Matches zero or one copy of the preceding expression. For example, -?[0-9]+ matches a signed number including an optional leading minus.
|	Matches either the preceding expression or the following expression.
( )	Groups a series of expressions into a new expression. Useful when building up complex patterns with *, +, and |.
!	Negates the complete expression. It must be the first character in the expression.

iScript Variable Declarations

The syntax of the variable declaration is similar to the C/C++ syntax:

```
Type Name [=Value];
```

As an option, you can declare variables to be constants:

```
const Type Name = Value;
```

For example:

```
Long x;
String serviceCode = "Tel";
const Decimal pi = 3.1415927
```

iScript Arrays and Hashes

All data types except File can be used in arrays or hashes:

```
Long a[ ]; // A normal array declaration
String dist [ ] [ ]; // A 2-dimensional string array
String ndc { } [ ]; // An associative array
String cli { } [ ]; // An associative array or arrays
```

You do not have to specify the dimension of the arrays and hashes. The data structures are resized automatically and are initialized by default values. For numerical values, this is 0; for strings, it is an empty string and dates become invalid.

You can access arrays and associative arrays in the following way:

```
a [3] = 4711;
ndc { "040" } = Hamburg;
cli { "Max Mueller" } [0] = "0171123456789";
cli { "Max Mueller" } [0] = "0177471111245";
```

> **Note**
>
> If you use arrays and hashes in functions, clear them at the start of the functions, as in this example:
>
> ```
> function myFunction
> {
>     String str[];
>     arrayClear(str);
>      // ...
> }
> ```
>
> Arrays and hashes aren't initialized at the start of functions. They behave like static variables. If they are not cleared in a function, they retain values from the last time the function was run.

iScript Function Declarations

A function declaration has the following syntax:

```
function [returnType] identifier [( parameter [,parameter...])]
```

where *returnType* is optional. If you do not specify any return type, the default is VOID. A function can have an unlimited number of arguments. You can use the basic types as return and parameter types.

For example:

```
function Long square (Long x)
{
  return x·*;
}
```

> **Note**
>
> Avoid nesting functions. Nested functions can create unexpected results, as in this example:
>
> ```
> function myFunction
> {
>     Long i = 5;
>     // ...
>     myFunction();
>         // Here the variable i is assigned the value set in the nested fun
> ction.
>     // ...
> }
> ```

iScript Control Structures

The syntax of the control structures is similar to the C++ syntax, but because there are no implicit type casts, the following expression is *not* valid in iScript:

```
if ( i ) ...
```

You must use explicit type casts:

```
if ( i != 0 )...
```

There are AND and OR operators for Boolean expressions. Empty statements in FOR loops are not valid in iScript and there is no increment operator.

For example:

```
for ( ; i<100; i++ )
```

in C++ has to be replaced with

```
for ( i ; i<100; i=i+1)
```

in iScript.

iScript Function Blocks

A function block that is followed by a control structure must be enclosed in curly braces { }. This is also true if only one statement is in the function block.

For example:

```
if ( (edrString ( DETAIL.RECORD_TYPE) = = "H"))
  {
    logStdout ("Header detected\n");
  }
else
  {

  }
```

Using iScript Switch Statements

iScript provides switch statements for String values, Long values, and regular expressions. The syntax of the switch statement is similar to the C syntax. Follow these rules when including switch statements:

- Specify only one statement per **case** label.
- Use a statement block and enclose several statements between curly braces ({ }).
- Terminate every **case** label by a **break** statement. Otherwise the statement of the following **case** label is also run.
- For *regular expressions*, use the **regExprSwitchCase** statement instead of **switch**.

Examples for Switch Statements

Switch statements (Long)

```
switch ( edrLong ( DETAIL.RECORD_LENGTH ) )
{
  case 104:
    logStdout ( "Header record!" );
    break;
  case 685:
    {
           detail = detail + 1;
           logStdout ( "Detail record" );
    }
    break;
  default:
    logStdout ( "unknown record type" );
}
```

Switch statements (String)

```
switch ( edrString ( DETAIL.ERROR_TEXT) )
{
   case "ERR_DATETIME":
      logStdout ("invalid date/time" );
      break;
   case "ERR_NOIMSI":
```

```
      logStdout ("IMSI not specified");
   case "":
      logStdout ("no error detected");
      break;
...
}
```

Switch statements for regular expressions

```
regExprSwitch ( edrString ( DETAIL.A_NUMBER ) )
{
   case "0049171.*":
      logStdout ( "D1-call\n" );
      break;
   case "0049172.*":
      logStdout ( "D2-call\n" );
      break;
}
```

Including Other iScript Source Files in Your iScript

You can use the **include** statement in your iScript to include other iScript source files.

Use the following syntax and specify each **include** statement in a separate line:

```
include "iScriptFile.isc";
```

Before an iScript is compiled, a preprocessor evaluates and processes the **include** statements. If the included iScript file has an absolute path, the preprocessor tries to include the file with the absolute path. Otherwise, the preprocessor uses a semicolon-separated list of include directories specified in the **ISCRIPT_INCLUDE** environment variable.

If the **ISCRIPT_INCLUDE** environment variable is not set, the preprocessor uses only the current directory as the input directory. If the environment variable is set, it does not contain the current working directory by default. You must explicitly add the current working directory to the list by using a dot (**.**) as the path. For example:

bash and **sh:**

```
export ISCRIPT_INCLUDE="/home/integRate/iscript include;/usr/iscript;."
```

csh and **tsch:**

```
setenv ISCRIPT_INCLUDE "/home/integRate/iscript_include;/usr/iscript;."
```

About iScript Functions

You use the iScript functions to perform a variety of operations. See "Pipeline Manager iScript Functions" in *BRM Pipeline Manager Reference*.

About Special iScript Functions

Pipeline Manager includes a basic iScript interpreter and an extended interpreter that calls special iScript functions on external events, such as **onHeaderEdr** or **onDetaiEdr.** These special functions are function hooks that you can use to implement any actions you want to

perform at specific situations, such as when a transaction is rolled back, during EDR processing.

For example, to perform custom actions when a transaction rolls back, you can include the following function block in an iScript:

```
Function onRollback
{
  logStdout("onRollback() \n");
  /* Define rollback-related actions here. */
}
```

Pipeline-Related Function Hooks

Use the functions listed in Table 10-15 to perform the actions you want at various stages of the pipeline process:

Table 10-15 Pipeline-Related Functions

Function	Description
BEGIN	Called when Pipeline Manager starts and after the iScript is compiled.
END	Called when Pipeline Manager is shut down.
onMessageReceived	Deprecated.
onStartEdr	Called when a pipeline starts. It is called once in the life of a pipeline process.
onStopEdr	Called when a pipeline stops. It is called once in the life of a pipeline process.

EDR Processing-Related Function Hooks

Use the functions listed in Table 10-16 to perform the actions you want during various stages of EDR processing:

Table 10-16 EDR-Processing Related Functions

Function	Description
onBeginEdr	Called when a BEGIN EDR container for each file opened in a transaction passes through a module.
onBeginTransaction	Called when a BEGIN_TRANSACTION EDR container passes through a module.
onDetailEdr	Called when a DETAIL EDR container passes through a module.
onEndEdr	Called when an END EDR container for each file opened in a transaction passes through a module.
onEndTransaction	Called when an END_TRANSACTION EDR container passes through a module.
onHeaderEdr	Called when a HEADER EDR container passes through a module.
onInvalidDetailEdr	Called when an EDR with invalid detail record is received by a module.
onTrailerEdr	Called when a trailer container passes through a module.

Input Grammar-Related Function Hooks

Use the function hooks listed in Table 10-17 in the input grammar to run the actions you want when the input module parses a stream:

Table 10-17 Input Grammar-Related Functions

Function	Description
onParseEnd	Called when the input module finishes parsing a stream.
onParseError	Called when the input module encounters an error while parsing a stream.
onParseStart	Called when the input module starts processing a stream.
streamIsEmpty	Called when an empty stream is encountered.

Transaction-Manager Related Function Hooks

Use these functions listed in Table 10-18 to perform the actions you want during a transaction.

Table 10-18 Transaction Manager Related Functions

Function	Description
onCancel	Called when a transaction needs to be cancelled.
onCommit	Called when the transaction manager notifies that a transaction is committed.
onPrepareCommit	Called when the transaction manager sends a request to prepare to commit a transaction.
onRollback	Called when a transaction needs to be rolled back.

About iScript Flist Extension Functions

You use the iScript flist extension functions to manipulate data in flists. For example, you can use the functions to retrieve information from an flist so that you can add it to the EDR container. You can also use the functions to add data taken from an EDR to an flist.

There are functions for creating flists, retrieving data from fields, setting field values, setting and unsetting the current array, deleting fields and arrays, and retrieving error text.

Here is a simple example of moving data from an EDR to an flist. Suppose you have the following EDR data block:

```
DETAIL.ASS_DATA
   String NAME
   Decimal VALUE
   Long QUANTITY
```

You can convert that EDR block to an flist in this format:

```
0 PIN_FLD_ARRAY   ARRAY
1   PIN_FLD_STRING   STRING
1   PIN_FLD_DECIMAL   DECIMAL
```

```
1   PIN_FLD_INT   INT
```

The following is an example of iScript code to convert the EDR block to an flist:

```
fListPushElem( "PIN_FLD_ARRAY", 0 );
fListSetString( "PIN_FLD_STRING", edrString( DETAIL.ASS_DATA.NAME, 1 ) );
fListSetDecimal( "PIN_FLD_DECIMAL",edrDecimal( DETAIL.ASS_DATA.VALUE, 1 ) );
fListSetLong( "PIN_FLD_INT", edrLong( DETAIL.ASS_DATA.QUANTITY, 1 ) );
fListPopElem();
```

Improving Pipeline Performance in Custom iScripts and iRules

This section provides some guidelines to reduce pipeline startup time and memory usage in your custom iScripts and iRules.

When the pipeline framework loads an iRule, a finite state machine (FSM) is built in memory. The number of objects the FSM creates in memory affects the pipeline startup and memory usage. FSMs create decision trees in memory at startup-time which affect pipeline startup, but decision trees help the pipeline work efficiently at run-time.

To reduce pipeline startup time and memory usage, follow these guidelines while creating iRules:

- Split the database entries because the number of rows in the loaded table affects memory usage and startup time of Rules.
- When writing rules, review the condition and reduce the number of compare patterns.
- Keep the regular expressions as simple as possible.
- If you have complex rules, place them at the beginning of an iRule.
- To improve processing performance, split big iRules if you have CPUs to allocate for new threads.

To improve pipeline processing performance, follow these guidelines when creating iScripts:

- If you can use an iRule instead of an iScript, use an iRule.
- Keep iScripts simple.
- Whenever possible, use standard modules instead of creating custom modules.
- Avoid database read access from the iScript when processing EDRs. Instead, load data during startup into hashes, and use that data when processing EDRs.
- Avoid functions that duplicate EDRs. They are performance intensive.
- Avoid writing to a database within an iScript. If you do, make sure you handle transactions properly.

Part II

Configuring Pipeline Discounting

This part describes how to configure pipeline discounting in Oracle Communications Billing and Revenue Management (BRM). It includes these chapters:

- Configuring Discounting Modules and Components
- Discounting Utilities

11

Configuring Discounting Modules and Components

Learn how to configure discounting in the Oracle Communications Billing and Revenue Management (BRM) Pipeline Manager.

Topics in this document:

Configuring a Batch Discounting Pipeline

Batch discounting is typically performed in a separate discounting pipeline.

Configure the following data modules:

- **DAT_AccountBatch**. This module provides account data to Pipeline Manager. This module loads account data into memory when you start Pipeline Manager, and updates it when it is changed in the BRM database.
- **DAT_BalanceBatch**. This module provides balance data for the FCT_Discount module when discounting is run in batch. This module loads balance data into memory when you start Pipeline Manager, and keeps the balance data synchronized between the Pipeline Manager database and the BRM database. See "DAT_BalanceBatch" in *BRM Pipeline Manager Reference*.
- **DAT_ModelSelector**. This module provides discount selector data to the FCT_DiscountAnalysis module. See "DAT_ModelSelector" in *BRM Pipeline Manager Reference*.
- **DAT_Discount**. This module supplies discount information to the discount function modules. See "DAT_Discount" in *BRM Pipeline Manager Reference*.

Configure the following function modules:

- **FCT_Discount**. This module performs the discount calculations and adds discounting data to the event data record (EDR). See "FCT_Discount" in *BRM Pipeline Manager Reference*.
- **FCT_DiscountAnalysis**. This module selects applicable discounts and prioritizes them. See "FCT_DiscountAnalysis" in *BRM Pipeline Manager Reference*.
- **FCT_Rounding**. This module rounds the balance impacts of discounting. Use the **Mode** registry entry to specify **Discounting**. See "FCT_Rounding" in *BRM Pipeline Manager Reference*.

- **FCT_ApplyBalance**. This module is used only for batch discounting. This module adds the discount balance impact to the EDR and updates the Pipeline Manager memory. See "FCT_ApplyBalance" in *BRM Pipeline Manager Reference*.

About Setting the Validity of Balance Elements Impacted by Discounts

The effective period of a granted balance element can start when a subscriber first consumes the balance element balance.

The following modules are used to set the validity period of balance elements that start on first usage when they are impacted for the first time:

- **DAT_BalanceBatch**. This module calculates the balance element validity period based on the EDR timestamp and initializes the validity period in memory.

  If the validity periods of all first-usage balance elements in the bundle should be synchronized, DAT_BalanceBatch adds information about those balance elements to the EDR.

- **FCT_ApplyBalance**. This module sets the validity period information in the EDR for all first-usage balance elements whose validity needs to be set. It then sends the entire EDR to an output stream.

  You specify the first-usage validity output stream in the FCT_ApplyBalance registry. See "FCT_ApplyBalance" in *BRM Pipeline Manager Reference*.

  You configure the output stream in the batch rating pipeline. See "Configuring Pipeline Output for First-Usage Products, Discounts, and Balance Elements".

To set the validity period of first-usage balance elements sent to the output stream, you configure Universal Event (UE) Loader. See "About Updating Validity Period Information in the BRM Database".

Configuring Batch Discounting to Restrict Balance Element Validity End Time

When the validity period of a granted balance element starts on first usage and ends relative to the start time, you can restrict the balance element end time to ensure the balance element balance cannot continue to be consumed after the charge offer or discount offer expires.

To restrict the validity end time of first-usage balance elements, configure "DAT_BalanceBatch" in *BRM Pipeline Manager Reference* to use the **RestrictResourceValidityToOffer** business parameter setting from the BRM database.

You configure DAT_BalanceBatch to use business parameter settings from the BRM database by performing the following:

- Configuring the "DAT_PortalConfig" in *BRM Pipeline Manager Reference* module in your registry file. This module must be listed before all other data modules in the registry file.
- Connecting the "DAT_BalanceBatch" in *BRM Pipeline Manager Reference* module to DAT_PortalConfig by using the **PortalConfigDataModule** registry entry.

When you restrict balance element validity end time, DAT_BalanceBatch sets the end time of the balance element validity period to the end time of the charge offer or discount offer that grants the balance element if it is earlier than the balance element validity end time.

Calculating the Match Factor of Parallel and Sequential Discounts

The *match factor* is the percentage of usage a single discount reduces when multiple discounts apply. The match factor is used with cascading discounts, in which each discount applies only to the portion of the usage that is not yet discounted.

For example, if an account owns two cascading discounts and the first one discounts 75% of the usage, the match factor is .75. The second discount, therefore, can be applied only to 25% of the usage.

By default, discounting does not calculate the match factor for parallel and sequential discounts. To enable this calculation, set the **AvoidMatchFactorCalculation** entry to **False** in the FCT_Discount module registry. See "FCT_Discount" in *BRM Pipeline Manager Reference*.

Configuring a Real-Time Discounting Pipeline

You can use a real-time discounting pipeline to calculate discounts for events that are rated by real-time rating. This allows you to discount all events in real time, so customer discount balances are always current.

To configure real-time discounting:

1. Configure a real-time discounting pipeline. See "Configuring a Real-Time Discounting Pipeline".
2. Configure the Input registry section. See "Configuring the Input Registry Section".
3. Configure the Connection Manager (CM) to send discounting requests to the NET_EM module.

Configuring a Real-Time Discounting Pipeline

Configure a real-time discounting pipeline that includes the following real-time discounting modules:

- **INP_Realtime**. This module handles flist-to-EDR format translation for a real-time pipeline. See "INP_Realtime" in *BRM Pipeline Manager Reference*.

  Use this entry for the **OpcodeMapping** entry:

  ```
  OpcodeMapping = ./formatDesc/Formats/Realtime/discount_event.xml
  ```

- **OUT_Realtime**. This module handles EDR-to-flist format translation for a real-time pipeline. See "OUT_Realtime" in *BRM Pipeline Manager Reference*.
- **NET_EM**. This module provides an interface to the CM for the INP_Realtime and OUT_Realtime modules and an interface to the BRM database for the DAT_AccountRealtime and DAT_BalanceRealtime modules.
- **DAT_AccountRealtime**. This module provides account cycle data to the FCT_Discount module. The DAT_AccountRealtime module gets data from the BRM database by connecting with the NET_EM module. It does not store data in memory, so it does not load data when you start Pipeline Manager. See "DAT_AccountRealtime" in *BRM Pipeline Manager Reference*.
- **DAT_BalanceRealtime**. This module provides balance data for the FCT_Discount module for real-time discounting. The DAT_BalanceRealtime module gets data from the BRM database by connecting with the NET_EM module. It does not store data in memory, so it

does not load data when you start Pipeline Manager. See "DAT_BalanceRealtime" in *BRM Pipeline Manager Reference*.

- **FCT_CreditLimitCheck**. This module is used only for real-time discounting. This module is used during the prepaid authorization process to determine whether event owners have enough balance elements in their account balance to use a requested service. For more information, see "FCT_CreditLimitCheck" in *BRM Pipeline Manager Reference*.

In addition, configure the following standard discounting modules:

- **FCT_Discount**. This module performs the discount calculations and adds discounting data to the EDR. See "FCT_Discount" in *BRM Pipeline Manager Reference*.
- **FCT_DiscountAnalysis**. This module selects applicable discounts and prioritizes them. See "FCT_DiscountAnalysis" in *BRM Pipeline Manager Reference*.

Configuring the Input Registry Section

To manage real-time discounting efficiently, you must enable the Input registry **UnitsPerTransaction** entry. Use the following entry in the Input registry section:

```
Input
{
   UnitsPerTransaction = 1
   NoThread = true
...
```

About Dumping Discount Information During Runtime

You can dump discount configuration information during runtime. Dumping the information writes it to a file or to the terminal. This is useful should you need to verify, compare, or provide discount configuration information during runtime for troubleshooting purposes.

You can use the DAT_Discount and DAT_ModelSelector modules to write discount configuration information. You can write discount configuration information for one or all discounts in your system by using the **DiscountModel** and **DataFileName** semaphores. See "DAT_Discount" and "DAT_ModelSelector" in *BRM Pipeline Manager Reference*.

About Discount Transaction Management

To maintain data integrity in a pipeline, discounting uses transactional processing on two levels:

- Standard pipeline transactions, managed by the Transaction Manager (TAM). If a transaction fails, the input is stopped and all open transactions are rolled back. After rolling back the data, the input is restarted.

  > **Note**
  >
  > To enable transaction management, the **redoEnable** registry parameter of the TAM must be set to **True**. If the redo mechanism is not enabled, discounting will block any other sequential transactions.

- EDR transactions, managed by the discounting modules. An EDR might contain multiple charge packets that are manipulated by the FCT_Discount module. The module might find errors in some charge packets and not be able to finish processing the EDR. Therefore, all

changes made in one EDR are logged. If there are no errors, the data is committed, otherwise the changes made by the module are rolled back and the EDR is not committed.

About Processing Balance Groups Locked by Other Transactions

During discount calculations in BRM, when discounting data is added to the event data record (EDR), the associated balance group is locked by the transaction. By default, there is a dependency between concurrent transactions involving the same balance group. If transaction A locks a balance group, then, by default, transaction B waits for transaction A to commit or roll back before it locks the same balance group.

You can enhance pipeline throughput performance in BRM by configuring the **IgnoreEDROnLock** entry in the FCT_Discount module to ignore an event data record, if the associated balance group object is locked by another transaction.

For more information on the **IgnoreEDROnLock** registry entry in the FCT_Discount module, see "FCT_Discount" in *BRM Pipeline Manager Reference*.

To configure the FCT_Discount module for:

- Batch operations, set the **IgnoreEDROnLock** registry entry to **True**.
- Run-time operations, set the **IgnoreEDROnLock** semaphore entry to **True**.

When the FCT_Discount module processes concurrent transactions involving the same locked balance group objects, if **IgnoreEDROnLock** entry is set to **True**, it places the ignored or rejected EDRs with the locked balance group in the **discountError** directory.

Configuring Custom Business Events for Pipeline Discounting

A business event is a BRM operation that you define in the Payload Generator EM configuration file (**payloadconfig.xml**).

To enable custom business events in Pipeline Manager, you need to list them in the DAT_BalanceBatch **CustomEvents** registry entry. For example, this entry enables an event named CycleRollover20days:

```
CustomEvents
{
   CycleRollover20days
 }
```

12

Discounting Utilities

Learn about the Oracle Communications Billing and Revenue Management (BRM) Pipeline Manager discounting utilities.

Topics in this document:

pin_discount_cleanup

Use this utility to change the status of expired discounts from **active** to **canceled** and to delete canceled discounts.

You use this utility to close or delete the discounts that are canceled in the middle of a cycle and the discount's validity rule is set to Full discount.

You can run this utility daily or add it to the **pin_bill_day** script to be run automatically.

You can also use this utility to cancel discount offers marked as tied but not linked with any charge offer. This is done using two modes:

- **close_tied**: Changes the status of discount offers marked as tied but not linked with any charge offer to **canceled**.
- **delete_tied**: Deletes closed discounts marked as tied.

Location

BRM_home/**bin**

Syntax

```
pin_discount_cleanup -m [close|close_inactive|delete|close_tied|delete_tied] [-n days] [-d date] [-v] [-t] [-help]
```

Parameters

-m close|close_inactive|delete|close_tied|delete_tied
Specifies whether to delete discounts when they are canceled:

- **close**

  Changes the status of all active, expired discounts to **canceled** without deleting the discounts.

- **close_inactive**

  Changes the status of inactive discounts to canceled without deleting them.

- **delete**

  Deletes all expired discounts.

- **close_tied**

  Changes the status of discount offers marked as tied but not linked with any charge offer to canceled.

- **delete_tied**

  Deletes closed discounts marked as tied.

-n *days*
The number of days prior to **-d** *date* for which the expired discounts are retained. The utility changes the status to **canceled** for the discounts that expired more than **-n** *days* prior to **-d** *date*.

- d *date*
The end date (in the format *MM/DD/YYYY*) of the period in which discounts that expired are retained.
For example, if **-n** is **5** and **-d** is **07/15/2025**, the status of the discounts that expired before 7/10/2025 are changed to canceled.

> **Note**
>
> - The expiry date cannot be greater than the current date. For instance, in the example above, if 7/10/2025 is greater than the current date, **pin_discount_cleanup** returns an error. Similarly, if only **-d** is specified, and is greater than the current date, **pin_discount_cleanup** returns an error.
> - If neither -**n** nor **-d** parameter is specified, the current date is used.

-v
Displays information about successful or failed processing as the utility runs.

> **Note**
>
> This parameter is always used in conjunction with other parameters and commands. It is not position dependent. For example, you can enter -v at the beginning or end of a command to initiate the verbose parameter. To redirect the output to a log file, use the following syntax with the verbose parameter. Replace filename.log with the name of the log file:
> **pin_discount_cleanup** *other_parameters* **–v** > *filename*.**log**

-t
Displays the number of records processed (the number of discounts that were canceled.

-help
Displays the syntax and parameters for this utility.

Results

To check results of running this utility, look in the log file (normally **default.pinlog**) for error messages. The log file is located in the directory from which the utility was started or in a directory specified in the utility's configuration file (**pin.conf**).

load_pin_snowball_distribution

Use this utility to load snowball discount distribution rules into the **/config/snowball_distribution** object in the BRM database. You define how snowball discounts are distributed in the **pin_snowball_distribution** file in *BRM_home***/sys/data/pricing/example**.

> **Note**
>
> This utility overwrites existing distribution rules. If you are updating distribution rules, you cannot load new distribution rules only. You must load a complete set of distribution rules each time you run this utility.

Location

*BRM_home***/bin**

Syntax

```
load_pin_snowball_distribution pin_snowball_distribution_file [-d] [-v]
```

Parameters

pin_snowball_distribution_file
The name and location of the file that defines the snowball distribution rules. The default **pin_snowball_distribution** file is in *BRM_home***/sys/data/pricing/example**.
If you do not run the utility from the directory in which the file is located, you must include the complete path to the file. For example:

```
load_pin_snowball_distribution BRM_home/sys/data/pricing/example
```

-d
Creates a log file for debugging purposes. Use this parameter for debugging when the utility appears to have run with no errors, but the data has not been loaded into the database.

-v
Displays information about successful or failed processing as the utility runs.

> **Note**
>
> This parameter is always used in conjunction with other parameters. To redirect the output to a log file, use the following syntax:
> **load_pin_snowball_distribution** *other_parameter* **–v >** *filename***.log**

Part III

Customer and Financial Management

This part describes how to configure customer management and financial functionality in Oracle Communications Billing and Revenue Management (BRM) when using Pipeline Manager. It contains the following chapters:

13

Configuring Balance Monitoring in Pipeline Manager

Learn about the Oracle Communications Billing and Revenue Management (BRM) Balance Monitoring Manager.

Topics in this document:

About Balance Monitoring

You use balance monitoring to enable your customers to monitor the balance of a group of accounts. Balance monitoring collects the balance impacts for a specified group of accounts and services, and notifies customers when their balance is too high. Customers and CSRs can also access the group's total balance at any time by using a custom client interface.

For information about balance monitoring, see "Managing Balance Monitoring Groups" in *BRM Managing Customers*.

Balance Monitoring with Pipeline Rating

Pipeline Manager processes monitored balances as follows:

1. Pipeline modules rate the event and apply any discounts.
2. The FCT_Account module determines whether the event owner belongs to any monitor groups and, if it does, enriches the MONITOR_LIST section of the EDR container with information about each monitor group.

> **Note**
>
> DAT_AccountBatch stores in-memory the list of monitor groups to which an account or service belongs. The module retrieves this information from the BRM database when you first start Pipeline Manager and when you create, modify, or delete a monitor group. For more information, see "About Synchronizing Monitor Data Between Cycle-fee and Pipeline Batch Rating".

3. The FCT_BillingRecord module generates a monitor packet for each monitor group.
   - When a balance packet impacts the event balance group, FCT_BillingRecord creates a MONITOR_PACKET for each monitor group in the monitor list.
   - When a balance packet does not impact the event balance group (for example, the discount share or charge share event balance groups), FCT_BillingRecord calls

DAT_AccountBatch to retrieve the monitor group list for the discount share or charge share owner's balance group.

The MONITOR_PACKET contains an account POID of the balance monitor, the balance group ID from the monitor group, balance element ID, amount, sub-balance impact, and the sub-balance block from the balance packet.

4. The pipeline generates an output file that includes any monitor balance impacts.
5. Rated Event (RE) Loader retrieves the pipeline output file and publishes any monitored balance impact data to the monitor queue.

About Synchronizing Monitor Data Between Cycle-fee and Pipeline Batch Rating

Pipeline batch rating and cycle-fee rating determine where to apply balance monitor impacts by accessing each member's list of monitor groups. For cycle-fee rating, this information is stored in each member's **/ordered_balgrp** object in the BRM database. For pipeline batch rating, this information is stored in memory by the DAT_AccountBatch module.

When you create, modify, or delete a balance monitor, BRM updates each member's **/ordered_balgrp** object and then uses the EAI framework and Oracle DM to send the updated information to Pipeline Manager, as follows:

1. BRM updates the member's **/ordered_balgrp** object and generates one of the business events listed in Table 13-1.

Table 13-1 Monitor-Related Actions

Action	Event
Create a new monitor group	**/event/group/sharing/monitor/create**
Modify a monitor group by adding or removing members	**/event/group/sharing/monitor/modify**
Delete a monitor group	**/event/group/sharing/monitor/delete**

2. The EAI framework and Oracle DM processes these business events and publishes them to the AQ database queue.
3. Pipeline Manager retrieves the data from the AQ database queue and updates its internal memory:
   - The DAT_Listener module retrieves the data from the queue and passes it to DAT_AccountBatch.
   - The DAT_AccountBatch module refreshes the monitor group list for all accounts.

Enabling Balance Monitoring in Pipeline Manager

Pipeline Manager determines whether balance monitoring is enabled by using the **BalanceMonitoring** entry from the multibalance instance of the **/config/business_params** object.

You must configure Pipeline Manager to use business parameter settings from the BRM database by performing the following:

- Configuring the DAT_PortalConfig module in your registry file. This module must be listed before all other data modules in the registry file.
- Connecting the DAT_AccountBatch module to DAT_PortalConfig by using the **PortalConfigDataModule** registry entry.

> **Note**
>
> You must enable balance monitoring in Pipeline Manager to have notification events generated during the batch rating process.

14

Setting Up Pipeline-Triggered Billing

Learn how to set up Oracle Communications Billing and Revenue Management (BRM) pipeline-triggered billing.

Topics in this document:

About Pipeline-Triggered Billing

When you use pipeline batch rating, if an account is currently in the process of being billed, incoming call records are suspended (not rated) for that account until its billing is complete. The number of accounts being billed affects the time it takes to complete the billing process. When you need accounts to be billed quickly so that their new usage can be rated, you can set up Pipeline Manager to trigger billing. This will reduce the number of call records that might need suspending. When Pipeline Manager triggers billing for an account, it is billed in a separate billing process.

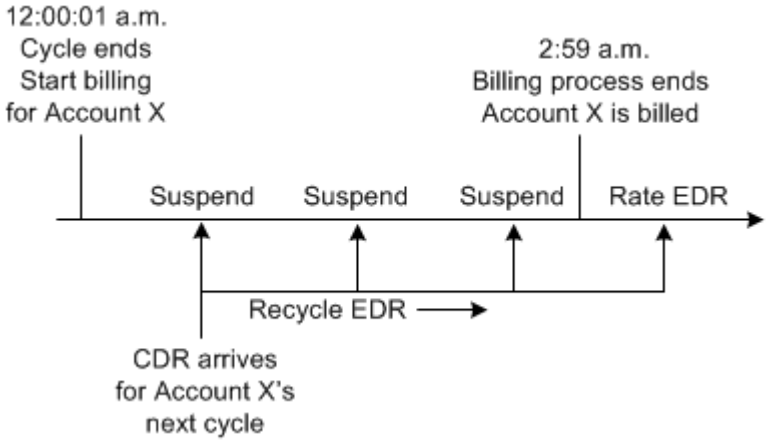
You use pipeline-triggered billing when event data records (EDRs) arrive for the next accounting cycle before the associated accounts have been billed. Pipeline Manager triggers billing for these accounts, enabling the new usage to be rated sooner and reducing the number of EDRs that might need suspending or rerating.

When customers use their services while their accounts are being billed, Pipeline Manager does not rate their usage until the accounts' billing is complete. The BRM billing process can sometimes take several hours. When there is account activity during the billing process, there is a greater possibility that call detail records (CDRs) will need recycling or rerating. When you use pipeline-triggered billing, the accounts are billed in a separate billing process, reducing the billing processing time.

When Pipeline Manager suspends EDRs because it is waiting for the account to be billed, those EDRs must later be recycled for rating. Each time an EDR is recycled, it is suspended until Pipeline Manager receives notification that the billing process for the account is complete.

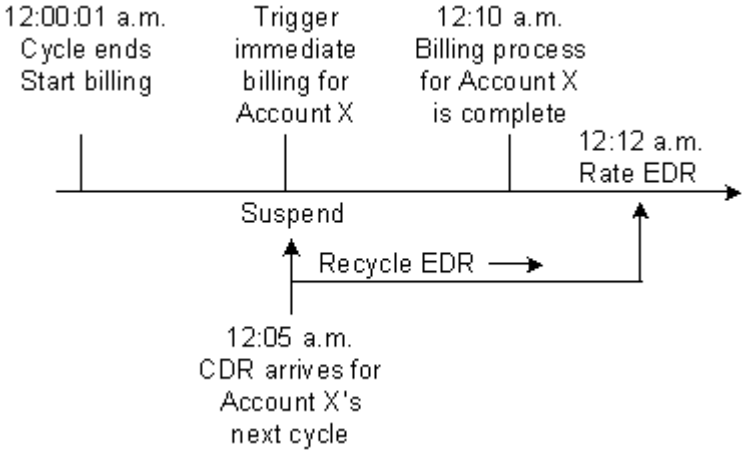
Figure 14-1 shows how a CDR is repeatedly suspended for Account X until the account's billing is complete. In this example, the billing process takes almost three hours.

Figure 14-1 Suspension of CDR Rating During Billing



With pipeline-triggered billing, billing is triggered for Account X when the first new CDR arrives, as shown in Figure 14-2. The account is billed in a separate billing process, reducing the processing time. Pipeline Manager can then rate the recycled CDR and new CDRs that arrive for the next accounting cycle for that account:

Figure 14-2 Pipeline-Triggered Billing



> **Note**
>
> Performance is affected by the number of accounts that need pipeline-triggered billing. If too many accounts are triggered for billing by Pipeline Manager, there is no performance advantage.

Pipeline-Triggered Billing Components

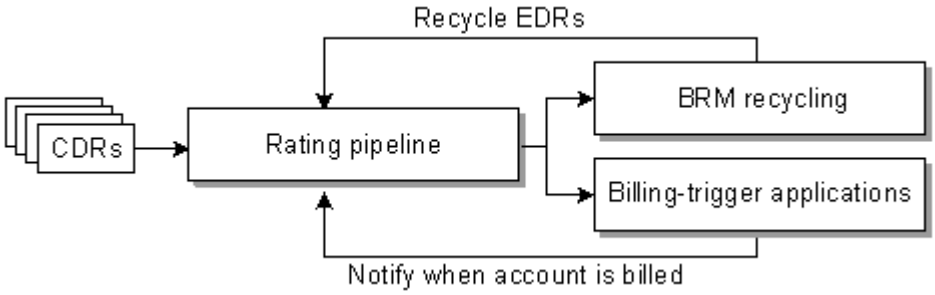
Pipeline-triggered billing comprises several components that work together to bill accounts:

- **Pipeline Manager modules** flag and route EDRs for accounts that require billing. See "About the Pipeline-Triggered Billing Modules".

- **BRM recycling** suspends EDRs flagged to trigger billing until the accounts are billed. The EDRs are periodically recycled. When the accounts are billed, the recycled EDRs can be rated and are no longer suspended. See "About Suspending Billing-Trigger EDRs".
- **Billing-trigger applications** bill the triggered accounts. These applications include a billing batch handler (BillHandler) and the BRM billing utility. When the accounts are billed, BRM notifies Pipeline Manager that billing for the accounts is complete. See "About BillHandler".

Figure 14-3 shows the relationships among Pipeline Manager, BRM standard recycling, and the billing-trigger applications.

Figure 14-3 Pipeline Manager, Recycling, and Billing Applications Relationship



How Pipeline Manager Triggers Billing

The following steps describe the entire process of pipeline-triggered billing:

1. BRM runs billing when the accounting cycle ends.
2. An EDR enters a pipeline for processing.
3. The FCT_TriggerBill module checks if the EDR belongs to the next accounting cycle and if partial billing has not yet been triggered for the account. When both of these conditions are true, it sends the EDR to the billing-trigger output stream.
4. The FCT_Reject module sends the EDR to the suspense output stream.

   For more information, see "About the Pipeline-Triggered Billing Modules".
5. When the EDR reaches the output modules, the data in the EDR takes two routes:

   Suspend EDR:

   a. The suspense output module suspends the EDR by sending it to Suspended Event (SE) Loader. SE Loader stores the EDR in the BRM database.

   b. The **pin_recycle** utility retrieves the EDR from the BRM database and sends it back through the rating pipeline. The EDR begins the cycle again: If billing for the account is complete, the EDR is rated. If billing is not complete, the EDR is again suspended.

      For more information, see "About Suspending Billing-Trigger EDRs".

   Trigger billing:

   a. The pipeline billing-trigger output module creates a file containing the account and bill units associated with the EDR.

   b. The billing-trigger applications retrieve the file and bill the account associated with the EDR.

      For more information, see "About BillHandler".

c. BRM uses the Oracle Data Manager (DM) to notify Pipeline Manager that the account is billed.

d. When the recycled EDR is sent back through a rating pipeline (through the Suspend EDR route), the EDR is rated because the account has been billed.

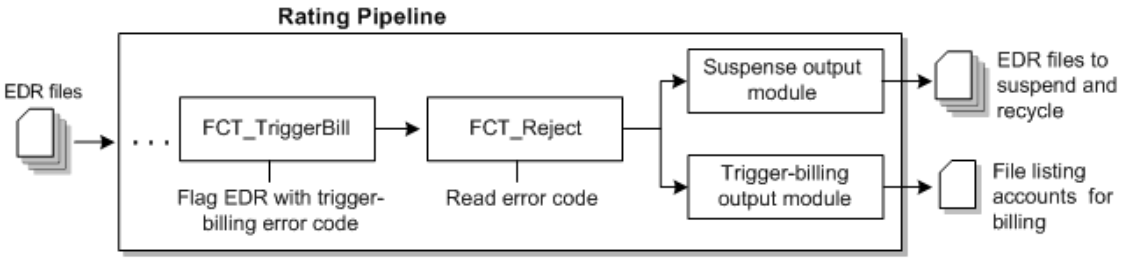
About the Pipeline-Triggered Billing Modules

Pipeline Manager flags EDRs associated with accounts that require immediate billing and sends them to the appropriate output streams. Pipeline-triggered billing uses the following modules:

- The FCT_TriggerBill module, which determines whether EDRs should trigger billing based on the accounting cycle date and billing state. To trigger billing, it sets a billing-trigger error code (**Awaiting billing of account**) in the EDRs. To flag the EDRs for recycling, it sets a billing-trigger recycle key value (**Trigger_Billing**).
- The FCT_Reject module, which detects the billing-trigger error code and sends the EDR to the suspense output stream.
- The billing-trigger output module, which creates a file containing the accounts and bill units for billing and sends the file to a separate directory.
- The suspense output module, which adds the EDRs that trigger billing to an output file. The EDRs are loaded into the BRM database to be suspended and recycled.

Figure 14-4 shows the path that EDRs take when they are flagged to trigger billing.

Figure 14-4 EDR Path for Triggered Billing



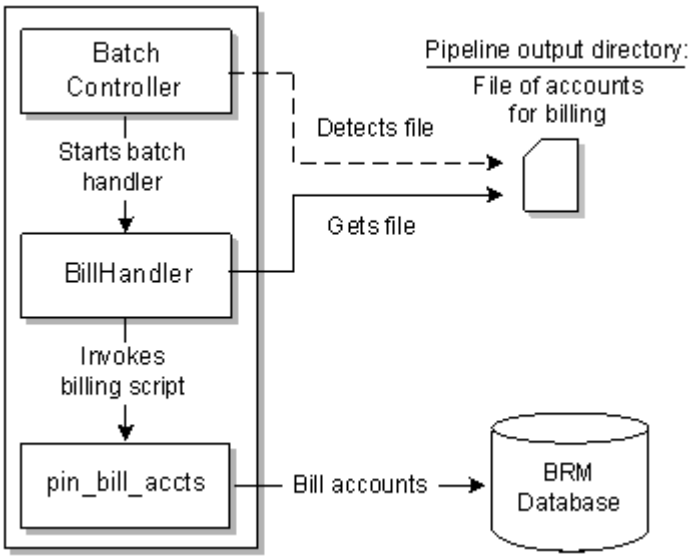
About BillHandler

BillHandler bills the accounts whose EDRs are flagged to trigger billing. BillHandler is used with the following applications:

- **Batch Controller,** which watches for billing-trigger files output by a pipeline. When a file is present, Batch Controller starts BillHandler.
- **The pin_bill_accts billing utility,** which bills the accounts and bill units. BillHandler starts the billing utility.

Figure 14-5 shows the batch handler billing process:

Figure 14-5 Batch Handler Billing Process



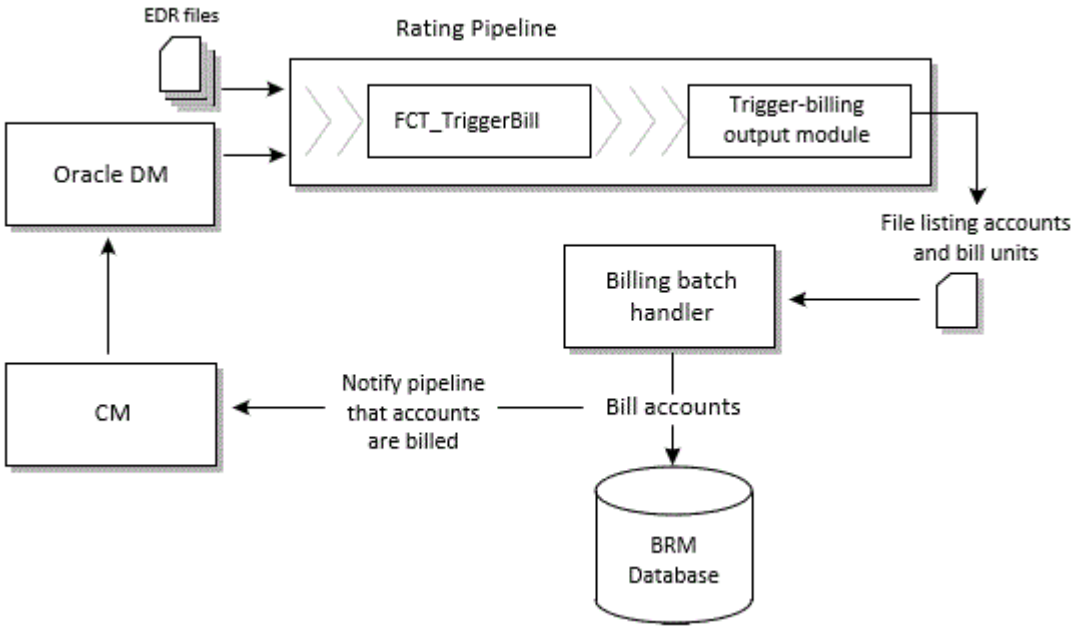
Overview of the Immediate Billing Process

To bill accounts identified for immediate billing, the following actions are performed:

1. Batch Controller starts BillHandler when a file is present in the pipeline billing-trigger output directory.
2. BillHandler reads the accounts and bill units in the file and passes them to the **pin_bill_accts** billing utility to be billed.
3. The **pin_bill_accts** billing utility creates the bills and updates the account information in the BRM database.
4. When the accounts are billed, BRM notifies Pipeline Manager by sending a business event to the Oracle DM.

Figure 14-6 shows an overview of the processes required to bill accounts that are identified for immediate billing.

Figure 14-6 Immediate Accounts Billing Process



About Suspending Billing-Trigger EDRs

EDRs that are flagged to trigger billing are not rated and must be periodically recycled until the accounts are billed. Pipeline-triggered billing uses BRM standard recycling to suspend and recycle these EDRs.

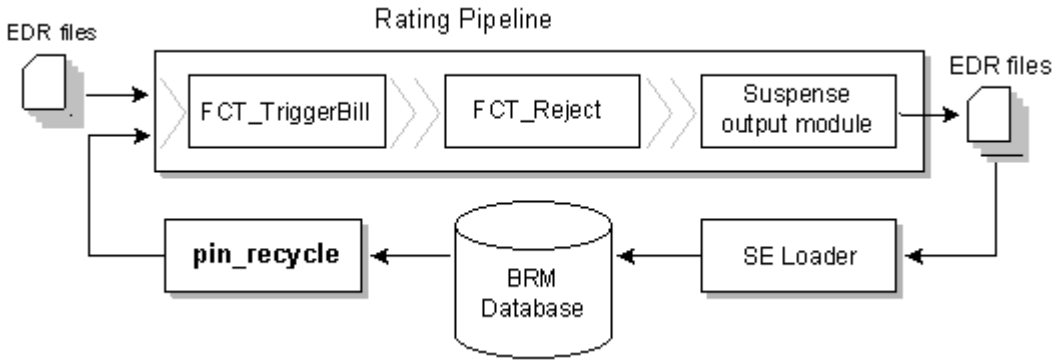
You perform the following actions to suspend and recycle EDRs:

1. When EDRs contain the **Awaiting billing of account** error code, the FCT_Reject module sends them to the suspense output stream.
2. SE Loader retrieves the EDRs from the output stream and stores them in the BRM database.
3. The **pin_recycle** utility retrieves the EDRs from the BRM database and sends them back through the rating pipeline. To continuously recycle any waiting EDRs, schedule the utility to run periodically.

EDRs are suspended each time they are recycled until the accounts are billed. After the accounts are billed, the EDRs are rated and are no longer sent to the suspense output stream.

Figure 14-7 shows an overview of the processes required to suspend and recycle EDRs flagged to triggered billing.

Figure 14-7 Suspension and Recycling of EDRs Flagged to Trigger Billing



Configuring Pipeline-Triggered Billing

To configure pipeline-triggered billing:

- Configure Pipeline Manager. See "Setting Up Pipeline Manager to Trigger Billing".
- Configure Batch Controller. See "Setting Up the Billing Batch Applications".
- Configure Pipeline Manager recycling. You can use either standard recycling or Suspense Manager with pipeline-triggered billing.

Setting Up Pipeline Manager to Trigger Billing

To set up Pipeline Manager to trigger billing:

1. Configure the FCT_TriggerBill module.

   Use the **TriggerBillCreateStream** entry in the module registry to specify the billing-trigger output module.

2. Configure the billing-trigger output stream in the OUT_GenericStream module.

   The default output grammar file is **TriggerBilling_OutGrammar.dsc**.

   The following is an example of the registry entries for the billing-trigger instance of the OUT_GenericStream and EXT_OutFileManager modules:

```
TriggerBillCreateOutput
{
   ModuleName = OUT_GenericStream
   ProcessType = RATING_PIPELINE
   EventType = /event/delayed/session/telco/gsm
   Module
   {
      Grammar = ./formatDesc/Formats/TriggerBill/TriggerBilling_OutGrammar.dsc
      DeleteEmptyStream = True
      OutputStream
      {
         ModuleName = EXT_OutFileManager
         Module
         {
            OutputPath            = ./data/TriggerBill
            OutputPrefix          = trigger_billing
            OutputSuffix          = .tb
            TempPrefix            = .
```

```
            TempDataPath          = ./data/TriggerBill
            TempDataPrefix        = trigger.billing.tmp.
            TempDataSuffix        = .data
            AppendSequenceNumber  = True
            Replace               = True
         }
      }
   }
}
```

> **Note**
>
> To ensure output file integrity, specify a unique combination of OutputPath, OutputSuffix, and OutputPrefix values for each output stream defined in the registry.

3. Add the format and mapping files to the DataDescription registry.
   - The default **StreamFormats** file is **TriggerBilling.dsc**.
   - The default **OutputMapping** file is **TriggerBilling_OutMap.dsc**.

   The following is an example of the billing-trigger entries in the DataDescription registry:

```
DataDescription
{
   Standard
   {
   ModuleName = Standard
   Module
   {
   StreamFormats
   {
      TRIGGERBILL_CREATE_OUTPUT =./formatDesc/Formats/TriggerBill/TriggerBilling.dsc
   }
   . . .
   OutputMapping
   {
      TRIGGERBILL_CREATE_OUTPUT =                              ./formatDesc/Formats/TriggerBill/
TriggerBilling_OutMap.dsc
   }
   }
   }
}
```

Setting Up the Billing Batch Applications

To set up the billing batch applications for pipeline-triggered billing:

- Configure batch controller to start BillHandler. See "Configuring Batch Controller to Start BillHandler".
- Configure BillHandler. See "Configuring BillHandler".

Configuring Batch Controller to Start BillHandler

Use Batch Controller to start BillHandler. You configure Batch Controller to start BillHandler when it detects a file in the pipeline billing-trigger output directory.

To configure Batch Controller:

1. Open the *BRM_home*/**apps/batch_controller/Infranet.properties** file, where *BRM_home* is the directory in which BRM is installed.
2. Add the entries shown in Table 14-1 for BillHandler.

Table 14-1 BillHandler Entries

Entry	Description
batch.random.events	Specify the event identifier. For example: **batch.random.events = Bill_Handler_file** For pipeline-triggered billing, this event is the appearance of a billing-trigger file that is output by a pipeline.
event_name.**name**	Specify a description for the event identifier. For example: **Bill_Handler_file.name = File passed to BillHandler**
event_name.**file.location**	Specify the full path to the pipeline billing-trigger output directory. This is the directory where the billing-trigger output module deposits the file of accounts to be billed.
event_name.**file.pattern**	Specify the billing-trigger output file name. When Batch Controller detects a file with this name, it starts BillHandler. **Tip:** You can use an asterisk (*****) to represent zero or more characters in the file name. No other wildcards are supported. For example: **Bill_Handler_file.pattern = *.out**
event_name.**handlers**	Specify BillHandler identifier. For example: **Bill_Handler_file.handlers = BillHandler**
handler_name.**name**	Specify a description for the BillHandler identifier. For example: **BillHandler.name = Bill Handler that executes pin_bill_accounts**
handler_name.**max.at.lowload.time** *handler_name*.**max.at.highload.time**	Specify the number of BillHandler instances that can run concurrently during periods of low-load and high-load usage. Typical default settings are **6** at low load and **3** at high load.
handler_name.**start.string**	Specify the command that starts BillHandler. The default is *BRM_home*/**apps/pin_bill_handler/BillHandler.pl**.

3. Save and close the file.

Configuring BillHandler

BillHandler retrieves the pipeline-triggered billing output file and sends the account and bill units to the billing utility. After the accounts and bill units are billed, BillHandler deposits the input file to a separate directory.

To configure BillHandler:

1. Open the *BRM_home*/**apps/pin_bill_handler/BillHandler_config.values** file.
2. Edit the entries listed in Table 14-2.

Table 14-2 Configuration Values for BillHandler

Entry	Description
$FILETYPE	Specify the EDR file-name pattern. For example, ***.txt.bc**. **Note:** The asterisk (*****) represents zero or more characters in the file name. No other wildcards are supported. Batch Controller runs BillHandler for each file with a name that matches this pattern.
$HANDLER_DIR	Specify the full path to the directory containing BillHandler, log, input, output, and other files. The default is *BRM_home*/**apps/pin_bill_handler**.
$pinBillActDir	Specify the full path to the directory containing the **pin_bill_accts** billing utility.
$STAGING	Specify the full path to the BillHandler input file location. **Note:** This is typically the same location specified for **$HANDLER_DIR**. You configure this location as the output directory.
$PROCESSING	Specify the full path to the directory from which the billing-trigger files are processed. The default is **$pinBillActDir**.

For information about other entries, see the **BillHandler_config.values** file.

3. Save and close the file.

Running the pin_bill_accts Utility

When Batch Handler runs the **pin_bill_accts** utility, it uses the **-from_file** parameter. This parameter specifies that the accounts to bill will be read from a file, and the associated file name. The input file is passed to the **pin_bill_accts** utility by the BillHandler billing batch handler to trigger billing for the accounts specified in the file.

15

Setting Up Revenue Assurance Manager for Pipeline Batch Rating

Learn how to configure Oracle Communications Billing and Revenue Management (BRM) Revenue Assurance Manager to collect revenue assurance data from pipeline batch rating.

Topics in this document:

About Revenue Assurance

You use Revenue Assurance Manager to verify the end-to-end completeness, accuracy, and integrity of BRM billing and pipeline batch rating results. You can analyze revenue assurance data to find revenue leakage in your system.

You obtain revenue assurance data by auditing the results of these processes:

- **Billing:** Revenue Assurance Manager provides statistics such as the number of accounts billed or invoiced, the total revenue, and the number of records that were successfully billed or failed to be billed.
- **Pipeline batch rating:** Revenue Assurance Manager provides statistics such as total duration and charges, retail and wholesale amounts, and total discount amounts.

About Collecting Revenue Assurance Data from Pipeline Batch Rating

You can collect revenue assurance data to analyze the effect of pipeline rating on Event Data Records (EDRs). You configure Revenue Assurance Manager to collect statistics on EDRs at various points in your pipelines, and then compare those statistics to see how a batch of EDRs changes as it is processed.

The statistics collected can include:

- The number of EDRs in the batch
- Retail charged amount
- Event wholesale value
- Discounts applied

- Total time usage
- Amount of data transferred
- When a call started
- When a call ended

Revenue assurance data is collected and aggregated at control points that you configure in pipelines. You establish control points by adding the FCT_AggreGate module at appropriate pipeline locations. You determine the data to be collected by specifying aggregation scenarios used by the module.

You can configure related control points into flows that enable you to examine data for a batch sequentially. You can also link pairs of control points to see original and current values.

FCT_AggreGate outputs aggregated data into flat files. You configure the Batch Controller to send these flat files to the Universal Event (UE) Loader as they are created. UE Loader parses the flat files and then calls opcodes to load the information into the database as ***/process_audit/batchstat*** objects.

Collecting revenue assurance data from pipeline batch rating involves these tasks:

- Configuring event notification to capture data on written-off EDRs and set up revenue assurance alerts.

  See "About Using Event Notification to Generate Revenue Assurance Data" for more information.

- Configuring control points in your pipelines to determine where revenue assurance data is captured.

  See "About Control Points" for more information.

- Associating an aggregation scenario with each of your control points to determine how revenue assurance data is organized. Revenue Assurance Manager includes preconfigured aggregation scenarios that group revenue assurance statistics by using different fields.

  See "About Aggregation Scenarios" for more information.

- Linking pairs of control points related to rating, rerating, and written-off EDRs. Revenue Assurance Center uses these linked pairs to establish the original and current values for a set of EDRs.

  See "About Linking Pairs of Rating, Rerating, and Written-Off Control Points" for more information.

- Defining flows, which are ordered lists of control points that Revenue Assurance Center uses to display revenue assurance data. Flows allow you to track revenue assurance data for EDR batches at various stages during pipeline processing.
- Configuring Universal Event (UE) Loader to load revenue assurance data into the database.
- Configuring alerts to be sent when revenue assurance data passes a threshold that you set.

About Using Event Notification to Generate Revenue Assurance Data

Revenue Assurance Manager uses event notification to collect data on written-off EDRs and set up revenue assurance alerts.

The following events are generated specifically to facilitate the revenue assurance event notification process:

- **/event/notification/suspense/writeoff**: When suspended EDRs are written off, Suspense Manager generates this event. By default, when this event occurs, the PCM_OP_PROCESS_AUDIT_CREATE_WRITEOFF_SUMMARY opcode is called.
- **/event/notification/ra_threshold**: When specified conditions for producing revenue leakage alerts occur, the **load_pin_config_ra_thresholds** utility generates this event. By default, when this event occurs, the PCM_OP_PROCESS_AUDIT_POL_ALERT policy opcode is called.

About Control Points

You establish control points in batch pipelines to determine where Revenue Assurance Manager collects data. You can configure control points in locations that enable you to compare data from different stages in the rating processes.

You define a control point by adding the FCT_AggreGate module to the pipeline in the desired location and specifying the control point name in the module registry. Each control point must have a unique name that describes its purpose. For example, in a rating pipeline, you could have control points named Rating and CP_After_Rating.

Each control point is associated with an aggregation scenario that specifies the data to be collected and how it should be organized.

You add related control points to *flows*. In Revenue Assurance Center, you can view data from all the control points in a flow. This enables you to follow the progress of EDR batches of EDRs through the pipeline. See "About Flows" for more information.

About Aggregation Scenarios

Each control point requires an aggregation scenario to specify the data that the control point collects and how the data will be organized.

An aggregation scenario specifies:

- The EDR fields to collect data from. For example, a scenario can collect information from the discount amount or duration fields.
- Aggregation functions to apply to the data. For example, you can add data together or average it.
- Fields by which to group the data collected. For example, you can group data by service type. In this case, the data for each batch is grouped by the type of service, such as TEL, SMS, or GPRS.

You can also collect revenue assurance data on more than one grouping field. For example, in the BatchStat_SvcType_Status scenario, the grouping fields are Service Type and EDR Status. The revenue assurance data collected is the EDR status such as Duplicate, Rejected, or Successful for each service type.

You specify the scenario to use when you define a control point in the FCT_AggreGate module registry. Scenarios can be used by any number of control points.

Revenue Assurance Manager provides a number of preconfigured scenarios. These scenarios are suitable for use in a production system. You can also create new scenarios if necessary.

About Linking Pairs of Rating, Rerating, and Written-Off Control Points

Certain types of control points must be linked to establish *original* and *current* values displayed in Revenue Assurance Center. You should link these types of control points:

- A rating control point to a rerating control point.
- A rating control point to a writeoff control point.
- A rerating control point to a writeoff control point.

When pairs of control points are linked, Revenue Assurance Center shows the linked control points in the same area. Values for the first control point are marked as *original* values, and the last control point as *current* values.

About Flows

A flow is an ordered set of related control points that you group together for convenient searching and viewing in Revenue Assurance Center. You can add any number of control points to a flow, from any pipelines that are relevant.

You use Revenue Assurance Center to display data for flows. Each control point appears in its own area. In this example, you can see two control points (P1_END and CP_AfterRating) from Flow12. The second control point shows the effect that rating had on a small batch of EDRs.

About Using UE Loader to Load Revenue Assurance Data

In order to view batch pipeline revenue assurance data, you must first load that data into **/process_audit** objects in the BRM database. You use Universal Event (UE) Loader to load the data.

The UE Loader loads revenue assurance data into the following **/process_audit** objects:

- **/process_audit/pipeline** objects store information about processed EDRs.
- **/process_audit/batchstat** objects store information about the revenue assurance data collected for specific scenarios.

You can configure the BRM Batch Controller to probe automatically for revenue assurance data files and then call UE Loader to load them into the database. You can also decide to load the data periodically by using **cron** or a similar program.

About the Revenue Assurance Data Collected in Rated Event Loader

Revenue Assurance Manager collects data from Rated Event (RE) Loader. BRM uses RE Loader to load events that have been rated or rerated with pipeline batch rating.

The revenue assurance data collected from RE Loader include:

- Total revenue generated by the batch.
- Total number of EDRs loaded.

You can view the RE Loader data in Revenue Assurance Center or in Revenue Assurance reports.

When the rated EDRs are loaded in RE Loader, either all the EDRs are loaded, or they all fail. There is no possibility of some records getting loaded and some records failing. In Revenue

Assurance Manager, the data collected for RE Loader is not grouped by status or any other fields.

You must load the **CollectProcessAuditForIREL.sql** file to enable the collection of revenue assurance data from RE Loader.

About Collecting Revenue Assurance Data on Written-Off EDRs

You can use Revenue Assurance Manager to view statistics about the number of EDRs that have been written off.

When an EDR is written off through Suspense Manager, an **/event/notification/suspense/writeoff** event is generated. You configure event notification so that Revenue Assurance Manager collects data every time such an event occurs. The data collected from a written-off EDR includes the original batch ID and the number of EDRs that were written off in that batch of EDRs.

Configuring Revenue Assurance Manager

To set up Revenue Assurance Manager for pipeline batch rating, you need to complete the following tasks:

- Configure event notification to notify Revenue Assurance Manager when events occur.

  See "Configuring Event Notification" for more information.
- Choose scenarios that determine how the revenue assurance data is grouped.

  See "Selecting Aggregation Scenarios" for more information.
- Identify control points for pipeline data collection.

  See "Identifying Control Point Locations for Revenue Assurance Data" for more information.
- Configure FCT_Aggregate to use your aggregation scenarios and control points.

  See "Configuring the FCT_AggreGate Module to Collect Revenue Assurance Data" for more information.
- Configure SimpleSample files to map the batchIDs of the EDRs.

  See "Configuring SimpleSample Files" for more information.
- Add related sets of control points to flows.

  See "Adding Control Points to Flows" for more information.
- Link control points for rating, rerating, and written-off EDRs.

  See "Linking Rating, Rerating, and Write-Off Control Points " for more information.
- Configure Universal Event (UE) Loader to load revenue assurance data into the database.

  See "Setting Up UE Loader to Load Revenue Assurance Data into the Database" for more information.
- Configure Batch Controller to call UE Loader.

  See "Setting Up Batch Controller to Call UE Loader" for more information.

Configuring Event Notification

Revenue Assurance Manager uses event notification to collect data on written-off EDRs and set up revenue assurance alerts.

Before you can use Revenue Assurance Manager, you must configure the event notification feature as follows:

1. If your system has multiple configuration files for event notification, merge them.
2. Ensure that the merged file includes the entire event notification list in the *BRM_home***/sys/data/config/pin_notify_ra** file.
3. (Optional) If necessary to accommodate your business needs, add, modify, or delete entries in your final event notification list.
4. (Optional) If necessary to accommodate your business needs, create custom opcodes for event notification to trigger.
5. Load your final event notification list into the BRM database.

> **Note**
>
> In addition, your business needs may require that you use event notification to call opcodes when other objects are created in the BRM database.

Selecting Aggregation Scenarios

Aggregation scenarios determine what revenue assurance data is collected from pipelines and how that data is grouped.

You may be able to use one or more of the preconfigured scenarios supplied with Revenue Assurance Manager. Each scenario groups data differently.

If none of the preconfigured aggregation scenarios satisfies your business requirements, you can create your own.

You must associate aggregation scenarios with control points in the registry of the FCT_Aggregate module.

Loading Scenarios into the Pipeline Manager Database

You must load aggregation scenarios into the Pipeline Manager database before you can use them. Scenarios that you create are loaded into the database by Pricing Center at the time of creation.

To load scenarios into an Oracle Pipeline Manager database, run the following command against the Pipeline Manager database from the *Pipeline_home***/database/oracle/scripts** directory:

```
% sqlplus user@databaseAlias
Enter password: password

SQL> RevenueAssurance_Scenarios.sql
```

where:

- *Pipeline_home* is the directory in which you installed Pipeline Manager.
- *user* is the Pipeline Manager user ID.
- *password* is the Pipeline Manager user password.
- *database* is the Pipeline Manager database alias.

Identifying Control Point Locations for Revenue Assurance Data

You configure control points in pipeline locations where you want to collect revenue assurance data. You associate an aggregation scenario with the control point to determine the data that is collected.

The exact locations where you place control points depend on the data you are collecting. It is often useful to insert control points before and after a critical point in the pipeline. For example, you can insert a control point before and after rating so that you can analyze the impact on EDRs.

You place a control point in a pipeline by inserting an instance of the FCT_AggreGate module at the desired point in the pipeline. You specify the control point ID and aggregation scenario in the module registry. See "Configuring the FCT_AggreGate Module to Collect Revenue Assurance Data" for more information. Each control point ID must be unique system-wide.

Configuring the FCT_AggreGate Module to Collect Revenue Assurance Data

You insert the FCT_AggreGate module at locations that you identify to collect revenue assurance data. You define a control point and associated aggregation scenario in the registry of each FCT_AggreGate module that you insert.

> **Note**
>
> If an aggregation scenario requires one or more iScripts, you must specify them in the Function Pool section of the pipeline registry. See "Using iScripts to Derive Grouping Fields" for more information.

You also configure other options in the module registry, including details about control and result files. These are standard options unrelated to revenue assurance.

To configure an FCT_AggreGate module for revenue assurance:

1. Add the FCT_AggreGate module to the pipeline registry at the desired location.
2. Configure the module for revenue assurance data:
   - In the Scenarios section of the registry, create a block for the scenario by entering the scenario name. For example, enter **RA_03** to use the Service Type scenario.
   - For the **ControlPointId** parameter, enter an ID for the control point you are defining. The control point ID must be unique and have maximum of 30 characters. For example, if you use the Service Type scenario, you might define the **CP_PostRatingBatchStat_Svctype** control point.
   - For the **TableName** registry entry, enter the name of the scenario that you defined earlier.

> **Note**
>
> The value of the **TableName** entry is used as the name of the output files for the scenario. Using the scenario name for this entry makes it easier to associate files with the scenarios from which they were created.

- Add the registry parameter **IncludeProcessingTimestamps** and set it to **TRUE**.
- Enter values for the standard FCT_AggreGate parameters, such as **Threshold**, **TempDir**, **DoneDir**, **CtlDir**.
- Enter a semicolon (;) for the FieldDelimiter.
- Add the parameters **IncludeErrorEDRs** and **IncludeInvalidDetailEDRs** and set them to **TRUE**.

> **Note**
>
> When configuring scenarios that do not use grouping based on the field **EDRStatus**, do not specify the **IncludeErrorEDRs** and **IncludeInvalidDetailEDRs** parameters.

The following example shows FCT_AggreGate configured for control point **CP_PostRatingBatchStat_Svctype** using the Service Type aggregation scenario (**RA_03**):

```
# Aggregation
------------------------------------------------
# AggreGate
{
 ModuleName = FCT_AggreGate
 Module
 {
  Active                    = TRUE
  ScenarioReaderDataModule = ifw.DataPool.ScenarioReader
  Scenarios
  {
   RA_03
   {
    TableName       = RA_03
    Threshold       = 10000
    TempDir         = ./data/aggregate
    DoneDir         = ./data/aggregate
    CtlDir          = ./data/aggregate
    FieldDelimiter  = ;
    ControlPointId  = CP_PostRatingBatchStat_Svctype
    IncludeErrorEDRs            = TRUE
    IncludeInvalidDetailEDRs    = TRUE
    IncludeProcessingTimestamps = TRUE

  ResultFile
  {
   IncludeFormat  = FALSE
   TempSuffix     = .tmp
   DoneSuffix     = .dat
   WriteEmptyFile = FALSE
  }
  ControlFile
  {
```

```
    IncludeFormat  = FALSE
    Suffix         = .ctl
    DataFilePath   = TRUE
   }
  }
 }
}
```

Using iScripts to Derive Grouping Fields

Review the description of the scenario you are using to see if it requires any iScripts. Scenarios use iScripts to derive grouping fields such as **EDR Status**, **Revenue Stream**, and **Output Stream**.

If required, you need to specify the iScripts in the FunctionPool section of the registry file, before the FCT_AggreGate module.

- ISC_SetEDRStatus
- ISC_SetOutputStream
- ISC_SetRevenueStream
- ISC_SetRevenueFigures

The following iScript must be placed at the beginning of the pipeline to ensure that the batch ID is inserted before any further processing of the mediation batches.

- ISC_SetAndValidateBatchInfo

Configuring SimpleSample Files

Configure the SimpleSample files to map the batchIDs of the EDRs. This is a mandatory step for Revenue Assurance Manager to work correctly.

You can find the SimpleSample files at: *Pipeline_home***/ifw/formatDesc/Formats/Sample**.

To configure the SimpleSample files:

1. Open the **SIMPLESAMPLE_v1.dsc** file using a text editor such as Notepad.
2. Find the following line:

   ```
   SERVICE                    AscString();
   ```

3. Add the following lines before the aforementioned line:

   ```
   BATCH_ID                   AscString();
   ORIGINAL_BATCH_ID          AscString();
   SUSPENDED_FROM_BATCH_ID    AscString();
   EVENT_ID                   AscString();
   ```

4. Save and close the file.
5. Open the **SIMPLESAMPLE_v1_InMap.dsc** file using a text editor such as Notepad.
6. Find the following line:

   ```
   "020"                     -> DETAIL.RECORD_TYPE;
   ```

7. Add the following lines before the aforementioned line:

   ```
   BATCH_ID                -> DETAIL.BATCH_ID;
   ORIGINAL_BATCH_ID       -> DETAIL.ORIGINAL_BATCH_ID;
   SUSPENDED_FROM_BATCH_ID  -> DETAIL.ASS_SUSPENSEEXT.SUSPENDED_FROM_BATCH_ID;
   EVENT_ID                 -> DETAIL.EVENT_ID;
   ```

8. Save and close the file.
9. Open the **SIMPLESAMPLE_v1_InGrammar.dsc** file using a text editor such as Notepad.
10. Find the following line:

```
edrInputMap( "SIMPLESAMPLE_V1.DETAIL.STD_MAPPING" );
```

11. Add the following lines before the aforementioned line:

```
edrAddDatablock( DETAIL.ASS_SUSPENSE_EXT );
```

12. Save and close the file.

Adding Control Points to Flows

Flows are ordered sets of related control points. You group control points into flows so that you can search for and view them conveniently in Revenue Assurance Center. When you view a flow in Revenue Assurance Center, its control points are displayed in the order in which they were defined.

You define flows in the **pin_config_ra_flows** file and load the flows into the database by using the **load_pin_config_ra_flows** utility. Flows are stored as **/config/ra_flows** objects.

The **load_pin_config_ra_flows** utility overwrites existing flows. You must load a complete set of flows each time you run the utility.

To add control points into flows and load them into the database:

1. Open the *BRM_home***/sys/data/config/pin_config_ra_flows** file in a text editor.

   This file includes instructions about the syntax to use to add control points to flows.
2. Save and close the file.
3. Use the following command to load your control points into the database:

```
% load_pin_config_ra_flows pin_config_ra_flows
```

   If you do not run the utility from the directory in which the file is located, include the complete path to the file; for example:

```
% load_pin_config_ra_flows BRM_home/sys/data/config/pin_config_ra_flows
```

> **Tip**
>
> If you copy the **pin_config_ra_flows** file to the directory from which you run the **load_pin_config_ra_flows** utility, you do not have to specify the path or file name. The file must be named **load_pin_config_ra_flows**.

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

Linking Rating, Rerating, and Write-Off Control Points

For Revenue Assurance Center to recognize original and current values for certain types of control points, you must link them and add the links to the BRM database. The control points that require linking include:

- Rating to rerating.
- Rerating to written-off.

- Rating to written-off.

To link control points, you define the links in the **pin_config_controlpoint_link** file and then load them into the BRM database by using the **load_pin_config_controlpoint_link** utility. Links are stored in the **/config/link_controlpoint** object in the BRM database. The PCM_OP_PROCESS_AUDIT_CREATE_AND_LINK opcode links **/process_audit/batchstat** objects based on the data in the **/config/link_controlpoint** object.

> **Caution**
>
> The **load_pin_config_controlpoint_link** utility overwrites existing links. You must load a complete set of links each time you run the utility.

> **Note**
>
> The **load_pin_config_controlpoint_link** utility needs a configuration (**pin.conf**) file in the directory from which you run the utility.

To configure your control points into flows and load them into the database:

1. Open the *BRM_home***/sys/data/config/pin_config_controlpoint_link** file in a text editor. This file includes instructions on how to add control points into flows.
2. Save and close the file.
3. Use the following command to load your control points into the database:

   ```
   % load_pin_config_controlpoint_link pin_config_controlpoint_link
   ```

   If you do not run the utility from the directory in which the file is located, include the complete path to the file; for example:

   ```
   % load_pin_config_controlpoint_link BRM_home/sys/data/config/
   pin_config_controlpoint_link
   ```

   > **Tip**
   >
   > If you copy the **pin_config_controlpoint_link** file to the directory from which you run the **load_pin_config_controlpoint_link** utility, you do not have to specify the path or file name. The file must be named **pin_config_controlpoint_link**.

To verify that the control point links were loaded, you can display the **/config/controlpoint_link** object by using Object Browser, or use the **robj** command with the **testnap** utility.

Setting Up UE Loader to Load Revenue Assurance Data into the Database

UE Loader takes the revenue assurance data from the FCT_AggreGate output files and loads it into the BRM database. The following sections explain how to configure UE Loader.

Setting Up UE Loader Templates

UE Loader requires that each aggregation scenario have a separate UE Loader event input template in XML format to map the data generated by FCT_AggreGate to the input flist for the opcodes that data. Revenue Assurance Manager provides XML templates for each of the preconfigured scenarios.

Setting Up Batch Controller to Call UE Loader

Batch Controller runs programs such as UE Loader either when a specific event occurs or automatically at timed intervals.

Most implementations use the occurrence-driven processing feature of Batch Controller to run UE Loader whenever FCT_AggreGate creates output files with revenue assurance data.

> **Note**
>
> If you set up Batch Controller to load scenario data files periodically rather than by occurrence, make sure the files are loaded frequently. Revenue Assurance Center uses the time when records were loaded into the database when it searches for data in time range. For the revenue assurance data to be meaningful, it should be loaded into the database soon after creation.

Setting Up Revenue Assurance Manager to Collect Data on Written-Off EDRs

For written-off EDRs, Revenue Assurance Manager collects the original batch ID and the number of EDRs that were written off in the batch.

To collect revenue assurance data on written-off EDRs, do the following:

- Enable event notification for Revenue Assurance Manager.
- Set a parameter in the Connection Manager **pin.conf** file to configure the control point ID to collect the revenue assurance data on written-off EDRs. You can change this control point ID.

About Aggregation Scenarios

Aggregation scenarios specify the data that is collected by Revenue Assurance Manager from pipeline batch rating. You must specify an aggregation scenario at every control point you set up in the pipeline.

Revenue Assurance Manager includes scenarios that are suitable for use in a production system. These scenarios group revenue assurance data in different ways. See "Data Fields Collected by All Scenarios" for a list of the fields from which the scenarios collect data. See "Fields Used to Group Scenario Data" for a list of the fields by which the scenarios can group data.

For example, the EDR statistics for scenario **RA_03** are grouped by service type, such as TEL, SMS, or GPRS. Data, such as event count, retail charged amount, and so on, is aggregated for each service type.

Most of the scenarios group by combinations of grouping fields. For example, scenario **RA_04** groups fields by both service type (TEL, SMS, and so on) and EDR status (duplicate, rejected, written-off or successful). Data is aggregated separately for each combination; duplicate EDRs for the SMS service, for example.

If the preconfigured scenarios do not meet your business needs, you can also create your own.

> **Note**
>
> Custom scenarios require custom UEL templates and cannot be used with Revenue Assurance Center.

Before you can use the preconfigured scenarios, you must load them into the pipeline database.

Data Fields Collected by All Scenarios

All of the preconfigured aggregation scenarios collect statistics from the EDR fields listed in Table 15-1.

Table 15-1 All Scenarios Data Fields

Field	Description
Event count	Number of EDRs.
Retail charged amount	Retail charged amount collected.
Event wholesale value	Total wholesale amount charged for the EDR, if appropriate.
Discount amount	Discount amount applied.
Duration	Total usage time, in seconds, if appropriate (for example, for a voice call).
Volume sent	Data transferred in bytes, if appropriate (for example, for a GPRS event).
Volume received	Data received in bytes, if appropriate (for example, for a GPRS event).
Earliest call made	The earliest call start timestamp.
Latest call made	The latest call start timestamp.

Fields Used to Group Scenario Data

The EDR data fields in Table 15-2 are available to use for grouping data.

Table 15-2 Grouping Data Fields

Grouping Field	Description
Batch ID	Batch ID of the batch being processed.
Original batch ID	Mediation batch ID of the rerating or recycling EDR batches.
Suspended from batch ID	The batch ID from which the EDR was suspended.
Service type	Type of service, such as TEL, SMS, or GPRS.

Table 15-2 (Cont.) Grouping Data Fields

Grouping Field	Description
Revenue stream	Retail, Wholesale, or Roaming.
Output stream	The output stream of the EDR. This is based on the service type; for example, if the service type is SMS, the output stream is SMS.
EDR status	Success, Suspense, Duplicate, Discard, Rejected, or Skipped.
Suspense code	Suspense reason code assigned to the EDR.

Preconfigured Aggregation Scenario Details

Table 15-3 provides information about the Revenue Assurance Manager preconfigured aggregation scenarios.

Table 15-3 RA Manager Preconfigured Aggregation Scenarios

Scenario/ File Name	Collects Data for an EDR Batch Based On	Grouping Fields (Listed in the Grouping Order)	Storable Class	Installation Point	iScripts Required
RA_01, Batchstatat_simple	An EDR batch.	Batch ID, Original batch ID, Suspended from batch ID	**/process_audit/ batchstat/simple**	Anywhere in a pipeline	None
RA_02, BatchStat_status	EDR Status.	Batch ID, Original batch ID, Suspended from batch ID, EDR status	**/process_audit/ batchstat/status**	Anywhere in a pipeline	ISC_SetEDR Status
RA_03, BatchStat_Svc Type	Service type.	Batch ID, Original batch ID, Suspended from batch ID, Service type	**/process_audit/ batchstat/svctype**	After FCT_Service CodeMap	None
RA_04, BatchStat_Svc TypeStatus	Service type and EDR status. Using this scenario, you can find the number of records that are duplicate, rejected, or successful for each service type.	Batch ID, Original batch ID, Suspended from batch ID, Service type, EDR status	**/process_audit/ batchstat/ svctype_status**	After FCT_Service codeMap	ISC_SetEDR Status
RA_05, BatchStat_RevenueStream	Revenue stream. The revenue streams are: Retail, Wholesale, and Roaming.	Batch ID, Original batch ID, Suspended from batch ID, Revenue stream	**/process_audit/ batchstat/ revstream**	After Post Rating	ISC_SetRevenueStream
RA_06, BatchStat_Revenue Stream_Status	Revenue stream and EDR status.	Batch ID, Original batch ID, Suspended from batch ID, Revenue stream, EDR status	**/process_audit/ batchstat/ revstream_status**	After Post Rating	ISC_SetRevenueStream, ISC_SetEDR Status
RA_07, BatchStat_RevenueStream_SvcType	Revenue stream and service type.	Batch ID, Original batch ID, Suspended from batch ID, Revenue stream, Service type	**/process_audit/ batchstat/ revstream_svctype**	After Post Rating	ISC_SetRevenueStream

Table 15-3 (Cont.) RA Manager Preconfigured Aggregation Scenarios

Scenario/ File Name	Collects Data for an EDR Batch Based On	Grouping Fields (Listed in the Grouping Order)	Storable Class	Installation Point	iScripts Required
RA_08, BatchStat_RevenueStream_ServiceType_Status	Revenue stream, service type, and EDR status.	Batch ID, Original batch ID, Suspended from batch ID, Revenue stream, Service type, EDR status	**/process_audit/batchstat/revstream_svctype_status**	After Post Rating	ISC_SetRevenueStream, ISC_SetEDRStatus
RA_09, BatchStat_SvcType_RevenueStream	Service type and the revenue stream.	Batch ID, Original batch ID, Suspended from batch ID, Revenue stream, Service type	**/process_audit/batchstat/revstream_status_svctype**	After Post Rating	ISC_SetRevenueStream
RA_10, BatchStat_ServiceType_RevenueStream_Status	Service type, revenue stream, and EDR status.	Batch ID, Original batch ID, Suspended from batch ID, Revenue stream, Service type, EDR status	**/process_audit/batchstat/svctype_revstream_status**	After Post Rating	ISC_SetRevenueStream, ISC_SetEDRStatus
RA_11, BatchStat_Outputstream	Output stream.	Batch ID, Original batch ID, Suspended from batch ID, Output stream	**/process_audit/batchstat/outputstream**	Just before output	ISC_SetOutputStream
RA_12, BatchStat_ServiceType_RevenueStream_Outputstream	Service type, revenue stream, and output stream.	Batch ID, Original batch ID, Suspended from batch ID, Output stream, Service type, Revenue stream	**/process_audit/batchstat/svctype_revstream_outputstream**	Just before output	ISC_SetRevenueStream and ISC_SetOutputStream
RA_13, BatchStat_Suspense	Suspense reason code.	Batch ID, Original batch ID, Suspended from batch ID, Suspense code	**/process_audit/batchstat/suspense**	After FCT_Suspense	None
RA_14, BatchStat_ServiceType_RevenueStream_Status_Outputstream	Service type, revenue stream, EDR status, and output stream.	Batch ID, Original batch ID, Suspended from batch ID, Service type, Revenue stream, EDR status, Output stream	**/process_audit/batchstat/svctype_revstream_status_outputstream**	Just before output	ISC_SetRevenueStream, ISC_SetOutputStream, and ISC_SetEDRStatus

Creating New Aggregation Scenarios for Revenue Assurance

Revenue Assurance Manager uses aggregation scenarios to group the data that it collects from the pipeline. If the aggregation scenarios provided with Revenue Assurance Manager do not meet you needs, you can create your own.

> **Note**
>
> You cannot view data from custom scenarios in Revenue Assurance Center.

To create your own aggregation scenarios:

1. Create an aggregation scenario by using Pricing Center or PCC.
2. Create the appropriate **/process_audit/batchstat** subclasses.

> **Note**
>
> The new subclasses created must contain an array field of type PIN_FLD_GROUP_DETAILS. The PIN_FLD_ORIGINAL_BATCHID field must also be created in the array.

3. Create the XML templates used by Universal Event (UE) Loader to load the aggregation data generated by the new scenario.
4. Modify the PCM_OP_PROCESS_AUDIT_POL_CREATE_AND_LINK policy opcode to check for duplication of records in the newly created **/process_audit** object.

Tracking EDRs by Using Batch IDs

In pipeline batch rating, each batch file received from the mediation system is assigned a batch ID, which is stored in every EDR derived from the file. Each EDR also receives a unique event ID.

> **Note**
>
> The **KeepExistingBatchIds** registry entry in FCT_PreSuspense module controls the way batch IDs are set. For details, see "Setting the Default Batch ID Behavior".

During rerating and recycling, the EDR receives a new batch ID, but the original batch ID is retained in a different field. Retaining the original batch ID in the EDR makes it possible to determine the revenue impact of EDRs for each batch that is received from mediation, even if some EDRs are rerated or recycled.

Revenue Assurance Manager uses the following fields to track EDRs as they are processed by pipelines and as they are rerated or recycled:

- DETAIL.BATCH_ID
- DETAIL.ORIGINAL_BATCH_ID
- DETAIL.ASS_SUSPENSE_EXT.SUSPENDED_FROM_BATCH_ID

For example, BRM receives a batch file with batch ID **Mediation087**. All EDRs for events in the file are assigned this batch ID. The batch rating pipeline processes EDRs from this batch loads their data into the BRM database.

Later, some of the EDRs from this batch and a second batch, **Mediation099**, are rerated. During rerating, the two sets of EDRs from different batches are given the new batch ID **ReratingBatch007**. When the individual EDRs are given the new batch ID, their original batch IDs are moved to the ORIGINAL_BATCH_ID field.

Table 15-4 contains selected data from an EDR in the batch after rating.

Table 15-4 Rating EDR Data

Event ID	Duration	Charge	Batch ID	Original Batch ID
189	180	3	Mediation087	Mediation087

Table 15-5 contains the data for the EDR after rerating.

Table 15-5 Rerating EDR Data

Event ID	Duration	Charge	Batch ID	Original Batch ID
189	180	5	ReratingBatch007	Mediation087

Keeping Track of Rejected EDRs by Using Batch IDs

When a batch file is processed, some of its EDRs may not be able to be rated because of missing data or another reason. These rejected EDRs can be processed by Suspense Manager and recycled back into the rating pipeline. EDRs that are being rerated can also be rejected and sent to recycling.

When Suspense Manager recycles the rerated EDRs back through the pipeline, they receive new batch IDs based on the recycling batch. Their original batch IDs remain to reflect the mediation batch they started in. The suspended-from batch ID field (DETAIL.ASS_SUSPENSE_EXT.SUSPENDED_FROM_BATCH_ID) stores the ID of the batch in which the EDR was rejected. This could be the original batch or a rerating batch.

For example, two batches (MED1 and MED2) are received from the mediation system and processed by the batch rating pipeline. Some EDRs from each of the two batches are rejected and then recycled as part of batch RCL1. In addition, some EDRs from the original two batches are rerated as part of batch RRT1. Some of the EDRs in that rerating batch are rejected and then recycled as part of batch RCL2.

Table 15-6 summarizes how the three different batch ID fields change as EDRs are rated, rerated, and recycled.

Table 15-6 Batch ID Changed Fields

Pipeline Process	Value for BATCH_ID	Value for ORIGINAL_BATCH_ID	Value for SUSPENDED_FROM_BATCH_ID
Rating Batch MED1	MED1	MED1	MED1(1)
Rating Batch MED2	MED2	MED2	MED2(1)
Recycle Batch RCL1 (containing suspended EDRs from MED1 and MED2)	RCL1	MED1/MED2	MED1/MED2
Rerating Batch RRT1 (containing EDRs from MED1 and MED2)	RRT1	MED1/MED2	RRT1(1)
Recycle Batch RCL2 (containing suspended EDRs from RRT1)	RCL2	MED1/MED2	RRT1

> **Note**
>
> The value for SUSPENDED_FROM_BATCH_ID is ignored in rating and rerating. Because it is left blank, it is assigned the batch ID value.

By linking the control point in the original mediation pipeline to the control point in the recycle pipeline that processed the rerated EDRs, you can determine the revenue impact for each of the mediation batches and identify the revenue leakage in your system.

Setting the Default Batch ID Behavior

The **KeepExistingBatchIds** registry entry in FCT_PreSuspense module controls whether the values for batch IDs in EDR records are preserved as originally set by the input module.

- A value of **True** preserves the batch ID in each detail record of the batch input file.
- A value of **False** (the default) sets the batch ID of each record to the batch ID contained in the header record of the batch input file.

16

Setting Up Pipeline Manager Taxation

Learn how to configure your Oracle Communications Billing and Revenue Management (BRM) system to tax events rated by a batch pipeline.

Topics in this document:

About Pipeline Taxation

BRM can apply taxes to events rated by Pipeline Manager. You can configure your system to tax pipeline events in one of the following ways:

- **During the pipeline rating process**. Taxing events during pipeline rating allows you to generate pipeline output files that include both usage and tax charges. This is useful when output files are not sent through the billing process. For example, when you rate outcollect roaming calls, you rate calls from another carrier's customer and then send the pipeline output files directly to the other carrier for rating and verification.

  > **Note**
  >
  > You can apply only flat taxes during pipeline rating. For complex taxation, you must defer taxation to the billing process.

- **During the billing process**. Deferring taxation to the BRM billing process allows you to apply complex tax rules using third-party tax software or customized BRM policy opcodes. It also reduces rounding errors because BRM simultaneously taxes all events of the same type. This configuration works best for most wireless applications because it allows complex taxation.
- **During both pipeline rating and billing**. Taxing events during both pipeline rating and BRM billing allows you to:
  - Obtain itemized tax charges for each usage event.
  - Tax non-usage events, such as one-time purchases and cycle-time events that are rated in real-time rather than by Pipeline Manager.

Setting Up Pipelines to Tax Events

You can configure a pipeline to apply flat taxes to events. To use flat taxes during pipeline batch rating, you configure ISC_TaxCalc in your pipelines.

> **Note**
>
> A pipeline can apply only flat taxes. To use more complex taxation, you must defer taxation to the billing cycle.

ISC_TaxCalc must run *after* the FCT_MainRating module and *before* the FCT_BillingRecord module. If you want to round the tax balance impact, you can optionally run the FCT_Rounding module immediately after ISC_TaxCalc.

Pipeline Manager applies taxes as follows:

1. The pipeline input and function modules process call records.
2. The FCT_MainRating module rates the call record and adds charge packets to the associated charge breakdown record.
3. The ISC_TaxCalc iScript module applies a flat tax.

   See "About Applying a Flat Tax by Using the ISC_TaxCalc iScript " for more information.
4. The FCT_BillingRecord module consolidates the billing charges and flags whether an event was taxed in a pipeline.

   See "About Consolidating Tax Data by Using the FCT_BillingRecord Module" for more information.
5. (Optional) The FCT_Rounding module rounds tax balance impacts.
6. The remaining function and output modules finish processing the call records and generating output files.

See "Configuring Pipelines to Apply Flat Taxes" for information about configuring the pipeline for taxation.

Configuring Pipelines to Apply Flat Taxes

To configure your pipelines to apply flat taxes:

1. Use ISC_TaxCalc in your pipelines.

   See "About Applying a Flat Tax by Using the ISC_TaxCalc iScript " and "About Consolidating Tax Data by Using the FCT_BillingRecord Module" for more information.
2. Configure ISC_TaxCalc.

About Applying a Flat Tax by Using the ISC_TaxCalc iScript

You use the ISC_TaxCalc iScript to apply flat taxes to charges flagged for taxation.

When ISC_TaxCalc processes an event data record (EDR), it determines whether the EDR should be taxed by checking whether the **Tax Treatment** flag was applied to the pipeline charge that applies to the event. If the flag isn't set, ISC_TaxCalc ignores the EDR and passes it on to the next module in the pipeline. If the Tax Treatment flag is set, ISC_TaxCalc:

- Determines which tax rate to use by checking the **taxcodes_map** data stored in the cache, if the **taxcodes_map** information is provided:
  - If the event meets one of the tax criteria, ISC_TaxCalc uses the appropriate tax rate defined in the **taxcodes_map** file.

  - If the event does not meet any tax criteria, ISC_TaxCalc uses the default tax rate specified in the registry file.
- If the **TaxCodeMapFilePath** parameter is not defined, the module retrieves tax code map data from the **/config/taxcodes_map** object.
- Calculates the flat tax for the pre-tax amount.

> **Note**
>
> The pre-tax amount is the charged amount minus any discounts.

- Populates the tax packet data block in the EDR with the pre-tax amount, tax percentage, and tax balance impact.

About Consolidating Tax Data by Using the FCT_BillingRecord Module

You use the FCT_BillingRecord module to consolidate charge packets, discount packets, and tax packets into an associated BRM billing record in the EDR.

When an EDR is flagged for taxation, FCT_BillingRecord:

- Creates a balance impact packet that consolidates the charge packet and any tax packet data.
- Flags whether ISC_TaxCalc applied taxes to the EDR by using the PIN_DEFERRED_AMOUNT field. It sets this field to **0** when an EDR contains a tax packet and to PIN_AMOUNT when an EDR does not contain a tax packet.

Sample Flat Tax Configurations

This section shows how to set up flat taxes for two sample applications:

- Applying Flat Taxes to Outcollect Roaming Calls
- Applying Flat Taxes to Third-Party Content Usage

Applying Flat Taxes to Outcollect Roaming Calls

Roaming allows customers of one network operator to use their mobile phones in foreign networks. When customers travel to other regions, they can use the services of any network operator with a roaming agreement with their home network operator.

When customers from another network use your network (outcollect roaming), you rate those calls and bill the customer's network operator for this usage. Applying a flat tax during the rating process allows you to pass on local taxes to the other network operator.

To apply a local flat tax to outcollect roaming calls:

1. In PDC, ensure that the tax calculation is enabled in charge offers.
2. Set up a custom flat tax.
3. Configure the ISC_TaxCalc iScript in your outcollect processing pipeline.

   See "Configuring Pipelines to Apply Flat Taxes" for more information.

After the pipeline generates output files that include both usage and tax charges:

- Send one output file to the other network operator for verification and rating.
- Load settlement data into the BRM database and generate invoices for billing the other network operator.

Applying Flat Taxes to Third-Party Content Usage

Wireless carriers allow customers to access third-party content, such as news and sports scores, through mobile phones. Customers are often charged monthly subscription fees to access the content and usage fees when downloading files. Carriers bill their customers for this usage and then remit some of the fees to the third-party provider.

In this scenario, taxes are applied to:

- Usage events during the pipeline batch rating process.
- Cycle and one-time purchase events during the billing process.

To configure your system to apply flat taxes to third-party content usage:

1. In PDC, ensure that tax calculation is enabled in charge offers.
2. Set up your custom tax rates.
3. Configure the ISC_TaxCalc iScript in your pipelines.
4. Use Rated Event (RE) Loader to load your pipeline output files into the BRM database.
5. Configure BRM to perform deferred taxation.

When BRM completes the billing process, you can run the remittance utility and generate a report summarizing the amount owed to each third-party provider. See "Remitting Funds to Third Parties" in *BRM Configuring and Running Billing*.

17

Credit Limit and Threshold Checking During Batch Rating

Learn how to manage credit limits when using Oracle Communications Billing and Revenue Management Pipeline Manager.

Topics in this document:

About Credit Limit and Threshold Checking During Batch Rating

During batch rating, Pipeline Manager determines whether a customer's *noncurrency* balance has breached a credit threshold. Pipeline Manager modules compare a customer's balance against noncurrency credit threshold values and then compile a list of events that crossed thresholds. A utility loads the list from Pipeline Manager into BRM.

> **Note**
>
> Pipeline rating calculates balance impacts and credit threshold breaches immediately. However, the utility that loads the balance impacts into the BRM database is different from the utility that loads credit threshold breaches into BRM. This means that a notification for a credit threshold breach might be sent before the customer's balance is updated in the BRM database. Therefore, the customer balance shown on the threshold breach notification might temporarily differ from the balance in the customer's account.

The following pipeline modules are responsible for checking credit thresholds during the batch rating process:

- DAT_PortalConfig retrieves credit profile information from the BRM database and stores it in its internal memory.
- DAT_BalanceBatch retrieves the customer's current credit profile from the DAT_PortalConfig module, compares the customer's updated balance to the credit profile, and notifies FCT_ApplyBalance if a credit threshold or credit limit has been breached.
- FCT_ApplyBalance updates the customer's current balance in the DAT_BalanceBatch internal memory and, if a credit threshold has been breached, writes the event detail record (EDR) into an internal list.

  To help you identify the exact event that caused the threshold breach, FCT_ApplyBalance adds the following event attributes to each threshold breach notification:

  - Event type (DETAIL.EVENTTYPE)
  - Call originator (DETAIL.A_NUMBER)
  - Call destination (DETAIL.B_NUMBER)

At the end of the transaction, FCT_ApplyBalance writes all of the EDRs from the internal list to an XML output file. See "About the Format of the XML Output File Name".

After the XML output file is generated, Batch Controller uses the **load_notification_event** utility to load the notification events into BRM. See "About Loading Notifications from Pipeline Manager to BRM ".

About the Format of the XML Output File Name

Pipeline Manager generates output XML files using the following file name format:

OutputPrefix_PipelineName_StreamName_TransactionID_SequenceNumber

where:

- *OutputPrefix* is the prefix name specified in the **OutputPrefix** registry entry.
- *PipelineName* is the name of the individual pipeline, such as ALL_RATE.
- *StreamName* is based on the **unitsPerTransaction** parameter maintained at the input level of the pipeline. A value of **1** means that one input file generates one output file. A value greater than **1** means that multiple input files are merged into one output file. *StreamName* is replaced with the name of the last file merged into the output file. For example, if **unitsPerTransaction** was **3** and the files merged into the output file were **file01**, **file02**, and **file03**, *StreamName* would be replaced with **file03**.
- *TransactionID* is the system-generated transaction ID number.
- *SequenceNumber* signifies the order in which FCT_ApplyBalance created the XML output file within a transaction. For example, the first XML output file has a sequence number of 1, the second output file has a sequence number of 2, and so on.

For example:

```
balancenotification_ALL_RATE_file01_776_1
```

About Loading Notifications from Pipeline Manager to BRM

Notifications from Pipeline Manager are loaded into the BRM database by the **load_notification_event** utility. This utility is launched automatically by Batch Controller whenever Pipeline Manager writes an output XML file to a specified directory.

When Pipeline Manager writes an output XML file to the specified directory, notifications are loaded as follows:

1. Batch Controller launches the SampleLoadHandler batch handler.
2. SampleLoadHandler moves the output XML file to a processing directory and runs the **load_notification_event** utility.
3. **load_notification_event** converts the notification events in the output XML file into flist format and loads them into BRM.
4. The BRM event notification system sends an alert to the customer.
5. SampleLoadHandler determines whether the utility successfully loaded the notification events into BRM:
   - If the utility was successful, SampleLoadHandler moves the output XML file from the processing directory to the archive directory.
   - If the utility failed, SampleLoadHandler moves the output XML file from the processing directory to the reject directory.

You must configure Batch Controller, the batch handler, and the **load_notification_event** utility to process the output XML files. See "Configuring Batch Controller to Run load_notification_event".

Setting Up the Batch Rating Process to Perform Threshold Checking

To set up your system to perform threshold checking during the batch rating process:

- Enable threshold checking. See "Enabling Threshold Checking in Pipeline Manager".
- Configure Batch Controller to run the **load_notification_event** utility. See "Configuring Batch Controller to Run load_notification_event".
- Configure Pipeline Manager to check thresholds. See "Configuring Pipeline Manager to Perform Threshold Checking".

Enabling Threshold Checking in Pipeline Manager

By default, threshold checking is disabled for batch rating because it decreases pipeline rating performance.

You can enable threshold checking in batch rating by modifying a field in the **multi_bal** instance of the **/config/business_params** object.

To enable credit threshold checking during batch rating:

1. Go to the *BRM_home***/sys/data/config** directory and run the following command:

   ```
   pin_bus_params -r BusParamsMultiBal bus_params_multi_bal.xml
   ```

   This command creates an editable XML file named **bus_params_multi_bal.xml.out** in your working directory. If you do not want this file in your working directory, specify the path as part of the file name.
2. Search the XML file for the following line:

```
<CreditThresholdChecking>disabled</CreditThresholdChecking>
```

3. Set the **CreditThresholdChecking** entry to the appropriate value:
   - **enabledOffline** specifies to check credit thresholds during the batch rating process.
   - **disabled** specifies to skip credit threshold checking during the batch rating process and thus increase pipeline processing performance. This is the default setting.
4. Save and exit the file.
5. Rename the **bus_params_multi_bal.xml.out** file to **bus_params_multi_bal.xml**.
6. Go to the *BRM_home***/sys/data/config** directory and run the following command:

   ```
   pin_bus_params bus_params_multi_bal.xml
   ```

   This command loads your changes into the **multi_bal** instance of the **/config/business_params** object.

> **Note**
>
> BRM uses the XML in this file to overwrite the existing **multi_bal** instance of the **/config/business_params** object. If you delete or modify any other parameters in the file, these changes affect the associated aspects of the BRM **multi_bal** configuration.

7. Read the object with the **testnap** utility or Object Browser to verify that all fields are correct.

You do not need to restart the Connection Manager (CM) to update this entry.

Configuring Batch Controller to Run load_notification_event

Batch Controller is a **cronjob** application that monitors a processing directory and starts batch handlers. When configured for threshold checking, Batch Controller waits for output XML files to appear in the processing directory and then launches the SampleLoadHandler handler. The batch handler moves the output XML files to a separate directory and then calls the **load_notification_event** utility to load the notifications into the BRM database.

You configure Batch Controller by using the following configuration files:

- The **batch_controller.properties** file connects the Batch Controller to the CM, specifies run-time parameters, identifies the batch handlers to run, and specifies when and how to run the batch handlers.
- The **load_notification_event pin.conf** file connects the **load_notification_event** utility to the CM.
- The **SampleLoadHandler_config.values** file specifies where the batch handler retrieves and deposits the XML files and which utility to run.

To configure Batch Controller for threshold checking:

1. Open the *BRM_home***/apps/batch_controller/batch_controller.properties** file in a text editor.
2. Connect Batch Controller to the CM, if you have not already done so.
3. Make sure the entries for the **load_notification_event** batch handler (*BRM_home***/apps/load_notification_event/SampleLoadUtilityHandler.pl**) are correct. For example:

```
SampleHandler.name = SampleHandler
SampleHandler.max.at.lowload.time = 1
SampleHandler.start.string = $PIN_HOME/apps/load_notification_event/
SampleLoadUtilityHandler.pl
```

4. Specify the handler's polling directory and the file pattern to look for. For example:

```
event1.name = eventlabel1
event1.handlers = SampleHandler
event1.at = 115700
event1.file.location = $HOME/ifw/data/out/notifybalancebreach/
event1.file.pattern = *.xml
```

5. Save and close the file.
6. Connect the **load_notification_event** utility to the BRM database by editing the *BRM_home***/apps/load_notification_event/pin.conf** file.
7. Open the *BRM_home***/apps/load_notification_event/SampleLoadHandler_config.values** file in a text editor.

8. Specify the directory from which to retrieve the XML files in the $STAGING entry. For example:

   ```
   $STAGING = "$HOME/ifw/data/out/notifybalancebreach/"
   ```

9. Specify the directory in which the handler archives the output XML files that loaded successfully into BRM in the $ARCHIVE entry. For example:

   ```
   $ARCHIVE = "$HANDLER_DIR/archive"
   ```

10. Specify the directory in which the handler stores the output XML files that failed to load into BRM in the $REJECT entry. For example:

    ```
    $REJECT = "$HANDLER_DIR/reject"
    ```

11. Save and close the file.
12. Start Batch Controller.

Configuring Pipeline Manager to Perform Threshold Checking

To configure threshold checking, perform the following:

1. Configure the DAT_BalanceBatch module.
2. Configure the DAT_PortalConfig module.
3. Configure the FCT_ApplyBalance module.

   When you configure the FCT_ApplyBalance module, you specify the following:

   - The maximum number of notifications in the output file. See "Configuring the Maximum Number of Notifications".
   - The output file prefix and directory location. See "Configuring the Output XML File".

Configuring the Maximum Number of Notifications

You specify the maximum number of notification events that FCT_ApplyBalance writes into the XML output file by using the **NumberOfNotificationLimit** registry entry. When the file breaches the maximum number, FCT_ApplyBalance closes the file and begins writing notification events into a new XML output file.

> **Note**
>
> You can identify the order in which XML output files were created by the sequence number used in the file name. See "About the Format of the XML Output File Name".

Configuring the Output XML File

After FCT_ApplyBalance identifies that an EDR caused a balance to breach a credit threshold, it writes the EDR to an output XML file. You can specify the prefix for the output file name by using the **OutputPrefix** registry entry and the directory in which to save the file by using the **OutputDirectory** registry entry.