Oracle® Communications Billing and Revenue Management

Configuring and Running Billing

Release 15.2

G35232-01

January 2026

Oracle Communications Billing and Revenue Management Configuring and Running Billing, Release 15.2

G35232-01

Contents

About This Content

This guide describes how to configure and run billing in Oracle Communications Billing and Revenue Management (BRM).

Audience

This guide is intended for operations personnel and system administrators.

1 Running Billing Scripts

Learn how to run the Oracle Communications Billing and Revenue Management (BRM) billing utilities.

For an overview of the BRM billing process, see "Overview of Billing" in *BRM Concepts*.

Topics in this document:

> **Note**
>
> Before running billing, you must set the billing configuration defaults, as described in "Configuring Billing".

Running the Billing Scripts

Billing scripts run one or more billing utilities on a daily, weekly, or monthly basis. By default, billing scripts are located in *BRM_home*/**bin**.

You can customize the scripts.

Note the following:

- Use billing scripts for regular bills only, not for corrective bills.
- Run billing as **pin_user**, not as **root**. Running billing as **pin_user** provides the read and write permissions for billing.
- Since billing generates a lot of system activity, it's best to run billing scripts at night.
- Use a different time for all three scripts so you do not run billing utilities simultaneously.
- Check for billing errors daily. The log file for the billing utilities is **pin_billd.pinlog**.

To run the billing scripts, use a **cron** job with a **crontab** entry.

- The following **crontab** entry runs **pin_bill_day** at 1:00 a.m. daily:

  ```
  0 1 * * * BRM_home/bin/pin_bill_day &
  ```

- The following **crontab** entry runs **pin_bill_week** every Sunday at 12:05 a.m.

  ```
  5 0 * * 0 BRM_home/bin/pin_bill_week &
  ```

> **Note**
>
> By default, the **pin_bill_week** script runs the **pin_collect** utility with the utility's **end** parameter set to **7**. If you modify the script to run the utility with the **end** parameter set to **1** or **0**, do not run the script at the same time that you run the **pin_bill_day** script. If you do, accounts whose payment collection date is on the day or the day before the utility runs may be double charged.

- The following **crontab** entry runs **pin_bill_month** at 12:05 a.m. on the first day of the month:

```
5 0 1 * * BRM_home/bin/pin_bill_month &
```

Table 1-1 shows the billing utilities in each script.

Table 1-1 Utilities in Billing Scripts

Billing Script	Description
pin_bill_day	Run daily. Runs the following billing utilities: • **pin_deferred_act**: Runs deferred actions. See "pin_deferred_act" in *BRM Managing Customers*. • **pin_bill_accts**: Bills accounts. See "pin_bill_accts". • **pin_collect**: Collects BRM-initiated payments. See "pin_collect" in *BRM Configuring and Collecting Payments*. • **pin_refund**: Grants refunds. See "pin_refund" in *BRM Managing Accounts Receivable*. • **pin_inv_accts**: Generates invoices. See "pin_inv_accts" in *BRM Designing and Generating Invoices*. • **pin_deposit**: Deposits pre-authorized credit card payments, such as payments authorized when creating an account. See "pin_deposit" in *BRM Configuring and Collecting Payments*. • **pin_cycle_fees**: Prorates balance impacts for cycle forward fees. See "pin_cycle_fees".
pin_bill_week	Run weekly. Runs the **pin_collect** utility with the **-rebill** option on all active accounts with a payment collection date of at least 8 days old. Collects outstanding balances from active credit card or direct debit accounts that could not be collected during regular daily billing. For example, a daily billing run might return a soft decline on a BRM-initiated payment. In that case, the payment is not collected, but the bill is left open so that the **pin_collect** utility can attempt to collect the payment again when you run the **pin_bill_week** script.
pin_bill_month	Run monthly. Runs the **pin_collect** utility with the **-rebill** option on all closed and inactive accounts with a payment collection date that is at least 31 days old. Collects outstanding balances from closed or inactive accounts.

Changing the Path in the Billing Scripts

You can change the path to the billing utilities in the billing scripts. For example, to change the path for the **pin_bill_day** script, edit these lines:

```
PINDIR=/opt/portal/${VERSION}
CNFDIR=${PINDIR}/apps/pin_billd
```

```
NVDIR=${PINDIR}/apps/pin_inv
OGDIR=/var/portal/${VERSION}/pin_billd
PATH=/usr/bin:/bin:${PINDIR}/bin
cd ${CNFDIR}
```

Running the pin_bill_day Script for Bill Run Management

You must run the **pin_bill_day** script manually instead of automatically to do the following:

- Reduce the load and duration of a large daily billing run. See "Configuring a Split Billing Run".
- Add days to the due dates of bills in a daily billing run at run-time. See "Specifying Due Date Adjustments in a Billing Run".

When running **pin_bill_day** manually with bill run management, use this syntax:

```
pin_bill_day -file file_name
```

Where *file_name* is the name and location of a billing run configuration file.

The **-file** parameter when used with **pin_bill_day**, affects only the **pin_bill_accts** utility.

Note

When you use a **cron** job to run **pin_bill_day**, *do not* include the configuration file name. If you do, some bill units might never be billed.

Handling Billing Failures

Billing can fail in the following cases:

- When an internal BRM component, such as a CM or DM, goes offline.
- When the online payment processor goes offline.
- When a connection between BRM components is broken.

If the **pin_collect** utility or the **pin_deposit** utility is interrupted while it is in progress, you can run it again. However, you might need to resolve failed BRM-initiated payment transactions. See "Resolving Failed BRM-Initiated Payment Transactions" in *BRM Configuring and Collecting Payments*.

All other billing utilities, **pin_cycle_fees**, **pin_deferred_act**, **pin_inv_accts**, and **pin_bill_accts**, can be run again. You do not need to resolve failed transactions.

If the billing utilities were not run at all, for example, if the database was offline, you could run all of the billing utilities with no problems. The **pin_bill_accts** utility bills all accounts that are due for billing, not just those that are due on the day that you run the utility.

The **pin_collect** and **pin_inv_accts** utilities act on accounts that were billed by the **pin_bill_accts** utility, so if you run the **pin_bill_accts** utility first, payments for all accounts that are due are collected or invoiced.

2 About the Billing Utilities

Learn about the main billing utilities in Oracle Communications Billing and Revenue Management (BRM), **pin_bill_accts** and **pin_cycle_fees**.

Topics in this document:

- Billing Accounts By Using the pin_bill_accts Utility
- Using the pin_cycle_fees Utility

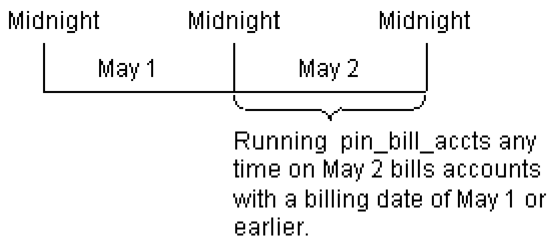
For information about all billing utilities, see "Billing Utilities".

Billing Accounts By Using the pin_bill_accts Utility

Use the **pin_bill_accts** utility to generate regular bills (not corrective bills).

The **pin_bill_accts** utility calculates the balance due for each account bill unit, and creates a bill for the balance due. It creates bills for accounts whose billing date is any day before midnight of the day that you run the **pin_bill_accts** utility as shown in Figure 2-1.

Figure 2-1 Accounts Included when Running pin_bill_accts


Midnight
Midnight
Midnight
May 1
May 2
Running pin_bill_accts any time on May 2 bills accounts with a billing date of May 1 or earlier.

The balance due amount is requested as a payment by the **pin_collect** utility, and is shown on the invoice.

The **pin_bill_accts** utility also performs the accounting cycle activity, such as creating new bill items.

You can use the **pin_bill_accts** utility to generate, accept, and reject a proforma invoice. For more information on proforma invoices, see "Generating Proforma Invoices" in *BRM Designing and Generating Invoices*.

For information about the **pin_bill_accts** utility syntax, see "pin_bill_accts".

When to Run the pin_bill_accts Utility

Use the **pin_bill_day** script to run the **pin_bill_accts** utility daily.

If you use bill unit hierarchies, you must run the **pin_bill_accts** utility to bill nonpaying child bill units before their paying parent bill units. The correct order is set in the **pin_bill_day** script.

You must run the **pin_bill_accts** utility before you run **pin_collect** because **pin_collect** needs the balance due amount collected by the **pin_bill_accts** utility.

About Bill Unit States and the pin_bill_accts Utility

The bill unit stores the state of a bill:

- **0** to indicate that the bill has been finalized.
- **1** to indicate that partial billing has been completed.
- **2** to indicate that all cycle charges and billing-time discounts have been applied at the end of the billing period and the bill needs to be finalized. A bill advances to this state only after billing is run for its account with the **pin_bill_accts -cycle_charge_only** parameter.

Running the **pin_bill_accts** utility with the **-cycle_charge_only** parameter runs billing on accounts with bill states 0 and 1. All cycle fees and billing-time discounts are applied, but totals are not calculated, and the bill is not finalized. At the end of the billing run, the bill state is set to 2.

To finalize a bill, run the **pin_bill_accts** utility with the **-finalize_bill** parameter. This runs billing on accounts with bill state 2. Totals are calculated, and items, bills, and so on are updated. At the end of the billing run, the bill state is set to 0.

Note

These bill unit states, stored in the **/billinfo** object, are different from the informational bill states stored in the **/bill** object. See "About Bill States" in *BRM Concepts* for more information about bill-level states.

Using the pin_cycle_fees Utility

The **pin_cycle_fees** utility can perform the following tasks at the account level or the service level:

- Apply charges for cycle-forward fees that have reached the end of free billing periods. For example, if a customer signs up for one month of free service, the **pin_cycle_fees** utility finds when the free period is over, and applies the cycle-forward fee balance impact to the customer's account balance group.
- Cancel charge offers when a pending cancellation comes due. For example, if a charge offer is set to cancel at a future date, the **pin_cycle_fees** utility cancels the charge offer on that date.

  Note

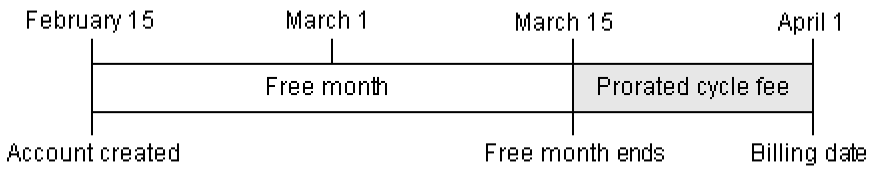
  You determine which of these two tasks **pin_cycle_fees** should perform by running it with different parameters.

- Split charges for backdated operations. If the backdate window spans multiple event cycles, the utility splits the events for each cycle and applies the charges accordingly.

If a free period ends before the customer's billing date, the **pin_cycle_fees** utility calculates the prorated fees for the time between the end of the free period and the start of the customer's next accounting cycle.

For example, a customer opens an account on February 15 and is given one free month, but the customer's billing date is on the 1st of the month. When you run **pin_cycle_fees** on March 15, it finds that the customer's free time period has ended. The utility then assesses the prorated fee due for March 15 through March 31 and impacts the customer's balance with the prorated amount. The result is that the system makes no charges to the customer on March 1, but charges the prorated fee and the cycle fee on April 1 as shown in Figure 2-2.

Figure 2-2 Proration of Cycle Forward Fees by pin_cycle_fees



For information about the **pin_cycle_fees** utility syntax, see "pin_cycle_fees".

When to Run the pin_cycle_fees Utility

Use the **pin_bill_day** script to run the **pin_cycle_fees** utility daily. This applies the prorated balance impacts as soon as they are due. If you do *not* run the **pin_cycle_fees** utility daily, the **pin_bill_accts** utility applies the balance impacts for the expired cycle forward fees. The only difference is that the balance impacts are not calculated by the **pin_cycle_fees** utility on the day that the cycle-forward fee expires.

Improving Performance of the pin_cycle_fees Utility

If system performance slows unacceptably when running the **pin_cycle_fees** utility, edit the **pin_bill_day** script and change the default **start** and **end** parameters for the **pin_cycle_fees** utility to every other day or every third day.

Canceling an Account’s Charge Offer Using the pin_cycle_fees Utility

When you run **pin_cycle_fees -defer_cancel** to cancel products for a specified account, it internally calls PCM_OP_SUBSCRIPTION_CANCEL_PRODUCT and then PCM_OP_SUBSCRIPTION_CYCLE_FORWARD, which cancels the account’s charge offers without applying cycle-forward fees to all other charge offers.

If you want BRM to apply cycle-forward fees to all charge offers before canceling an account’s charge offer, run **pin_cycle_fees -regular_cycle_fees** followed by **pin_cycle_fees -defer_cancel**.

Normally, **pin_bill_day** runs **pin_cycle_fees -regular_cycle_fees** followed by **pin_cycle_fees -defer_cancel**. If you run these utilities on days other than the billing day, you need to ensure that **pin_cycle_fees -regular_cycle_fees** is run first, followed by **pin_cycle_fees -defer_cancel**.

Processing a Subset of Accounts or Services Using XML Files

To run the **pin_cycle_fees** utility against a subset of account or services using an XML file:

1. (Optional) If you intend to process a subset of services, set the **StagedBillingFeeProcessing** business parameter is set to **1**, **2**, or **3**. See "Applying Cycle Forward Fees in Parallel" in *PDC Creating Product Offerings*.
2. Create an XML file:
   - For account-level processing, list the account POIDs in the following format:

     ```
     <CycleFeeList>
                  <Account>12345</Account>
     </CycleFeeList>
     ```

   - For service-level processing, list the account and service POIDs in the following format:

     ```
     <CycleFeeList>
                  <Account>12345</Account>
                  <Service>12345</Service>
     </CycleFeeList>
     ```

3. Run the **pin_cycle_fees** utility with the **-file** *filename***.xml** option. For example, to bill charge offers for a subset of accounts or services with a delayed purchase start time:

   ```
   pin_cycle_fees -defer_purchase -file filename.xml
   ```

Processing a Subset of Accounts or Services Using Files in Flist Format

To run the **pin_cycle_fees** utility against a subset of account or services using a file in an flist format with a **.txt** extension:

1. (Optional) If you intend to process a subset of services, set the **StagedBillingFeeProcessing** business parameter is set to **1**, **2**, or **3**. See "Applying Cycle Forward Fees in Parallel" in *PDC Creating Product Offerings*.
2. Create a TXT file in flist format:
   - For account-level processing, list the account POIDs in the following format:

     ```
     0 PIN_FLD_RESULTS            ARRAY [0] allocated 20, used 9
     1   PIN_FLD_POID              POID  [0] 0.0.0.1 /account 1808980
     0 PIN_FLD_RESULTS            ARRAY [1] allocated 20, used 9
     1   PIN_FLD_POID              POID  [0] 0.0.0.1 /account 203813 0
     ```

   - For service-level processing, list the account and service POIDs in the following format:

     ```
     0 PIN_FLD_RESULTS              ARRAY [0] allocated20, used 9
     1   PIN_FLD_POID                POID [0] 0.0.0.1 /account 180898 0
     1   PIN_FLD_SERVICE_OBJ         POID [0] 0.0.0.1 /service/email 184226 0
     0 PIN_FLD_RESULTS              ARRAY [1] allocated 20, used 9
     1   PIN_FLD_POID                POID [0] 0.0.0.1 /account 203813 0
     1   PIN_FLD_SERVICE_OBJ         POID [0] 0.0.0.1 /service/ip 201893 0
     ```

3. Run the **pin_cycle_fees** utility with the **-f** *filename***.txt** option. For example, to charge cycle forward fees for a subset of accounts or services:

   ```
   pin_cycle_fees -regular_cycle_fees -f filename.txt
   ```

3

Setting Default Billing Properties for Account Creation

Learn how to set up system-wide billing defaults, such as the default billing-cycle length, in Oracle Communications Billing and Revenue Management (BRM).

Topics in this document:

Setting the Default Accounting Day of Month

Note

It is a good idea to leave the accounting DOM set to the date the account was created. This distributes the load for the billing utilities throughout the month.

To set the default accounting day of month (DOM):

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

   ```
   pin_bus_params -r BusParamsCustomer bus_params_customer.xml
   ```

   This command creates an XML file named **bus_params_customer.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_customer.xml.out**, set **ActgDom** to the day of the month when the customer should be billed. Ordinarily, this can be between 1 and 28, but if you are using 31-day billing, it can be between 1 and 31. See "Configuring 31-Day Billing" for more information about 31-day billing.

   ```
   <ActgDom>value</ActgDom>
   ```

   The default value is **0**.

> ⚠ **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_customer.xml.out** file to **bus_params_customer.xml**.
6. Use the following command to load your changes into the **/config/business_params** object:

   ```
   pin_bus_params bus_params_customer.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*.

The new value becomes effective immediately and applies to the next account created.

Setting the Default Billing-Cycle Length

To set the default billing-cycle length:

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

   ```
   pin_bus_params -r BusParamsCustomer bus_params_customer.xml
   ```

   This command creates an XML file named **bus_params_customer.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_customer.xml.out**, set **BillWhen** to the number of months in one billing cycle:

   ```
   <BillWhen>value</BillWhen>
   ```

   The default value is **1**.

> ⚠ **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_customer.xml.out** file to **bus_params_customer.xml**.

6. Use the following command to load your changes into the **/config/business_params** object:

   ```
   pin_bus_params bus_params_customer.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*.

The new value becomes effective immediately and applies to the next account created. You do not need to restart the Connection Manager (CM) to enable this entry.

Specifying the Maximum Number of Months Allowed in Billing Cycles

A billing cycle defines the time period between bills, which corresponds to the billing frequency (how often a bill is generated). For example, if the billing cycle is three months, a bill is generated every three months.

The default maximum number of months allowed in billing cycles is 24.

To change the maximum number of months allowed in billing cycles, modify the value of the **bill_when** field.

Setting the Default Accounting Type

You can set the default accounting type for all bill units (**/billinfo** objects) using the business parameter **ActgType**. BRM uses this setting during account creation only when an accounting type is not passed in the input flist of the Customer FM standard opcode.

To set the default accounting type:

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

   ```
   pin_bus_params -r BusParamsBilling bus_params_billing.xml
   ```

   This command creates an XML file named **bus_params_billing.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_billing.xml.out**, set the following entry:

   ```
   <ActgType>value</ActgType>
   ```

   where *value* is one of the following:

   - **balance_forward**: Specifies that BRM uses balance-forward accounting.
   - **open_item**: Specifies that BRM uses open-item accounting.

The default value is **balance_forward**.

> **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_billing.xml.out** file to **bus_params_billing.xml**.
6. Use the following command to load your changes into the **/config/business_params** object:

   ```
   pin_bus_params bus_params_billing.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*.

The new value becomes effective immediately and applies to the next account created.

Setting the First Billing Cycle to the Day after Account Creation

Normally, an account is billed after one month on the day on which the account is created. For example, if an account is created on January 10, the account is billed on February 10, then on March 10, April 10, and so on. However, you can set the first billing date to be the day after account creation. For example, if an account is created on December 16, the account is billed on December 17. After the first billing run, all remaining bills for the account are generated normally. In this example, the account is billed on January 17, February 17, and so on. This option is called *advance billing cycle*.

To set the first billing cycle to the day after account creation:

1. Open the CM configuration file (*BRM_home*/**sys/cm/pin.conf**) in a text editor.
2. Add the following line and set the value to **1**.

   ```
   - fm_bill advance_bill_cycle 1
   ```

   > **Note**
   >
   > To set the first billing cycle to one month after the account is created, comment out the line by using the pound (#) symbol.

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

4

Configuring Billing

Learn how to configure Oracle Communications Billing and Revenue Management (BRM) billing.

Topics in this document:

- Specifying the Minimum Payment to Collect
- Customizing the Format of Bill and Invoice Numbers
- Specifying When to Apply Custom Bill Numbers
- Configuring Auto-Triggered Billing
- Setting the Bill Unit Status When Billing Errors Occur

Specifying the Minimum Payment to Collect

You can specify the minimum payment for billing. The **pin_collect** billing utility retrieves only those account bill units with an amount due greater than the minimum you specify. Charges accrue in the account balances associated with the bill unit until they reach the minimum amount, and then the amount due is collected.

The minimum value is expressed in terms of the account currency.

By default, the minimum amount is **2**.

1. Open the billing utility configuration file (*BRM_home*/**apps/pin_billd/pin.conf**) in a text editor.
2. Change the value of the **minimum** entry.
3. Save the file.

Setting the Minimum Amount to Charge

To set the default minimum amount to charge to a credit card:

- Customize the PCM_OP_PYMT_POL_PRE_COLLECT policy opcode. See "Customizing the Minimum Amount to Charge" in *BRM Opcode Guide*.
- Specify the amount in the **MinimumPayment** business parameter entry in the Connection Manager (CM).

To set the minimum charge:

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

   ```
   pin_bus_params -r BusParamsAR bus_params_AR.xml
   ```

   This command creates an XML file named **bus_params_AR.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_AR.xml.out**, set **MinimumPayment** to the desired minimum amount to charge:

   ```
   <MinimumPayment>value</MinimumPayment>
   ```

   The default value is **2.00**.

   > **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_AR.xml.out** file to **bus_params_AR.xml**.
6. Use the following command to load your changes into the **/config/business_params** object:

   ```
   pin_bus_params bus_params_AR.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*.

You do not need to restart the CM to update this entry.

There is no BRM configuration entry to set the minimum charge for accounts that pay their bills by using the invoice payment method.

> **Note**
>
> Ensure that the minimum credit card charge does not conflict with the minimum amount to collect.

Customizing the Format of Bill and Invoice Numbers

Bill numbering for regular bills is **B1-1**, **B1-2**, **B1-3**, and so on. For corrective bills, the default numbering is **CB1-1**, **CB1-2**, **CB1-3**, and so on. The default numbering takes the format:

*PrefixDB_num***-***Seq_Number*

where:

- *Prefix* represents the prefix to be used for the bill. **B** is used for regular bills, and **C** is used for corrective bills.
- *DB_num* indicates the number of the database in which the original bill is stored. For example, when *DB_num* is 1, 3, or 7, the prefixes for regular bills are **B1**, **B3**, or **B7**.

- *Seq_Number* is the sequence number.

To customize how BRM formats bill numbers and invoice numbers, create a custom application that calls the PCM_OP_BILL_POL_SPEC_BILLNO opcode. See "Customizing the Format of Bill and Invoice Numbers" in *BRM Opcode Guide*.

Specifying When to Apply Custom Bill Numbers

For corrective bills, BRM assigns a new bill number when it generates the corrective bill only. It does not support applying bill numbers at any other time in the corrective billing process.

For regular bills, you can control when custom bill numbers must be assigned to account bill units, which can be useful for revenue and expense accounting purposes. Custom bill numbers can be applied at the beginning of the first accounting cycle or at the end of the previous accounting cycle for multimonth billing cycles.

To specify when BRM assigns custom bill numbers:

1. Open the CM configuration file (*BRM_home*/**sys/cm/pin.conf**) in a text editor.
2. Change the value of the **custom_bill_no** entry.
   - **0** assigns custom bill numbers at the end of the previous accounting cycle. This is the default.
   - **1** assigns custom bill numbers at the beginning of the first accounting cycle.
3. Stop and restart the CM.

Configuring Auto-Triggered Billing

If you perform no offline usage charging, you might not need auto-triggered billing. You can enable or disable auto-triggered billing by setting the following business parameters:

- **ConfigBillingDelay**: By default, the **ConfigBillingDelay** parameter is set to **0**. If you are using delayed billing, you must set it to a value greater than **0**.

  When billing delay is specified, the system maintains an internal list of bill items for both the previous billing cycle and the current billing cycle so that new events impact bill items of the current billing cycle and old events impact bill items of the previous billing cycle.
- **AutoTriggeringLimit**: By default, **AutoTriggeringLimit** is set to **2**. For example, if billing for the previous cycle has not occurred and the billing for the current cycle is due in the next two days, then billing for the previous cycle is auto-triggered within these two days.

You set these parameters by running the **pin_bus_params** utility. For information on this utility, see "pin_bus_params" in *BRM Developer's Guide*.

To set the **ConfigBillingDelay** business parameter, see "Configuring Delayed Billing".

Note

If you do not use delayed billing but want to disable auto-triggered billing, set **ConfigBillingDelay** to **0**.

To set the **AutoTriggeringLimit** business parameter:

1. Go to *BRM_home***/sys/data/config**.

2. Create an XML file from the **/config/business_params** object:

   ```
   pin_bus_params -r BusParamsBilling bus_params_billing.xml
   ```

3. In the XML file, enter a value greater than **0**.

   For example, if you change the value to **5**, auto-triggered billing is enabled only for the last 5 days of each billing cycle.

   ```
   <AutoTriggeringLimit>5</AutoTriggeringLimit>
   ```

4. Load the XML file into the BRM database:

   ```
   pin_bus_params bus_params_billing.xml
   ```

5. Stop and restart the CM.

Setting the Bill Unit Status When Billing Errors Occur

When the billing utility (**pin_bill_accts**) encounters an error while generating a bill for a bill unit, the utility sets the billing status of the bill unit to PIN_BILL_ERROR. Bill units with an error status are not selected when billing is run.

Note

When billing fails for a nonpaying child bill unit, the status of the nonpaying bill unit and the paying parent bill unit are both set to PIN_BILL_ERROR. This ensures that when billing fails for a nonpaying bill unit, the paying bill unit is also not billed. Otherwise, the parent bill might not include charges from the nonpaying bill unit, resulting in incorrect billing.

After you have resolved the billing errors, you can rerun billing for the failed bill units by running the billing utility with the **-retry** option. See "pin_bill_accts".

To set the bill unit status when billing errors occur:

1. Open the billing utility configuration file (*BRM_home***/apps/pin_billd/pin.conf**) in a text editor.
2. Add the following line and enter the appropriate value.
   - **0** sets the billing status in the **/billinfo** object. This is the default.
   - **1** does not set the billing status in the **/billinfo** object.

   ```
   - pin_bill_accts unset_error_status 1
   ```

3. Save the file.
4. Run the billing utility.

5

Managing Billing and Accounting Cycles

Learn how to manage billing and accounting cycles in Oracle Communications Billing and Revenue Management (BRM).

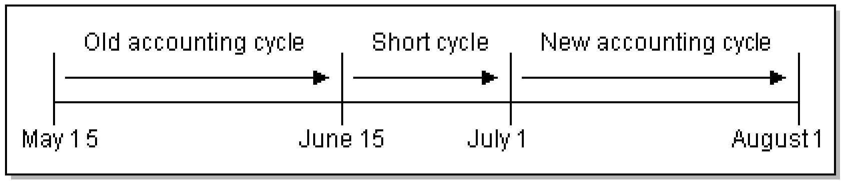
Topics in this document:

Specifying How to Handle Partial Accounting Cycles

When you change the accounting cycle date in the middle of an accounting cycle, the new date does not take effect until after the current accounting cycle is over. This results in a gap of time between the end of the old accounting cycle and the start of the new accounting cycle.

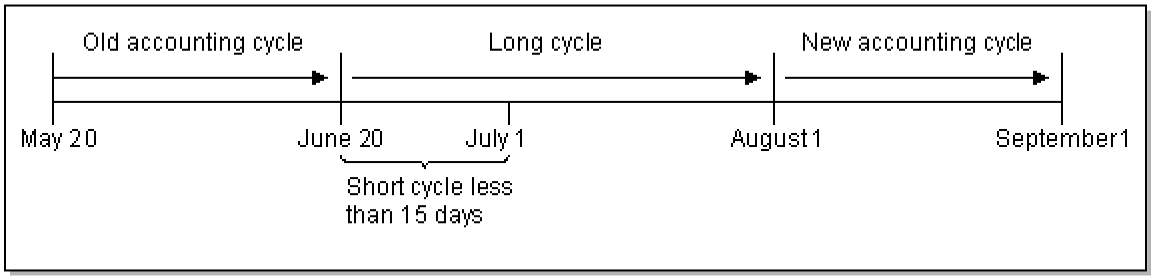
For example, for a 30-day month, if the current accounting cycle ends on the 15th and the new cycle starts on the 1st, there is a gap of 15 days between the end of the old cycle and the start of the new cycle. By default, the BRM system treats those 15 days as a short, but complete accounting cycle. At the end of that *short cycle*, the accounting cycle resumes its normal monthly cycle. A timeline for this scenario is displayed in Figure 5-1.

Figure 5-1   Short Accounting Cycle



If the short cycle is less than 15 days, a *long cycle* is created instead. In that case, the extra days are added to the next one-month accounting cycle. This results in a long cycle with the start date of the old cycle and the end date of the new cycle as seen in Figure 5-2.

Figure 5-2   Long Accounting Cycle



Monthly charges are prorated for accounting cycles less than or greater than one month.

A short or long cycle can also occur when a customer creates an account and the billing DOM is different from the day of month when the the account is created. For example, your company might require that all customers be billed on the first day of the month. If a customer creates an account on January 26, by default the first bill is created on March 1. To bill the customer on February 1, you must change the default partial billing cycle to short.

By default, BRM creates a short cycle and a one-month cycle. You can configure BRM to always create short cycles by setting the **ShortCycle** business parameter using the **pin_bus_params** utility. By default, the **ShortCycle** parameter is **disabled**. When this parameter is disabled, if the number of days in the current accounting cycle is less than 15 days, BRM creates a long cycle and adds the remaining days to the next month's accounting cycle. If the number of days in the current accounting cycle is greater than 15 days, BRM creates a short cycle.

To create short cycles:

1. Go to *BRM_home***/sys/data/config**.
2. Create an XML file from the **/config/business_params** object:

   ```
   pin_bus_params -r BusParamsBilling bus_params_billing.xml
   ```

3. In the XML file, set the value for the following entry to **enabled**:

   ```
   <ShortCycle>enabled</ShortCycle>
   ```

4. Load the XML file into the BRM database:

   ```
   pin_bus_params bus_params_billing.xml
   ```

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

You can also configure BRM to round up a long cycle so that the scale for the long cycle equals 2. This enables you to charge your customers for two full cycles.

To round up long billing cycles:

1. Open the CM configuration file (*BRM_home***/sys/cm/pin.conf**) in a text editor.
2. Change the value of the following entry to **1**:

   ```
   - fm_rate rating_longcycle_roundup_flag  1
   ```

3. Set the value of rounding precision to **0**:

   ```
   - fm_rate rating_quantity_rounding_scale 0
   ```

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

To change how BRM handles short and long cycles, customize the PCM_OP_CUST_POL_PREP_BILLINFO policy opcode source code. See *BRM Opcode Guide*.

Configuring Time-Stamp Rounding

By default, BRM rounds time stamps to midnight. You can configure BRM to use exact time stamps by changing the **TimestampRounding** business parameter.

The following features are affected by time stamp rounding. Before disabling time stamp rounding, consider how that change might impact these features:

- **Billing cutoff time.** To use a billing cutoff time other than the default, time stamp rounding must be enabled. Changing the cutoff time also changes the time to which time stamps are rounded throughout BRM. See "Configuring the Billing Cutoff Time" and "How Timestamp Fields Are Affected by Changing the Cutoff Time".
- **Unit interval used to calculate prorated cycle fees.** If time-stamp rounding is enabled, the unit interval is calculated in days because time stamps are rounded to midnight. If time-stamp rounding is disabled, the unit interval is calculated in seconds, and proration begins from the time indicated by the time stamp. See "Calculating the Unit Interval".
- **Validity period start time of resources granted by cycle events.** Use the **TimestampRounding** business parameter to specify whether this time stamp is rounded.
- **Validity period start time of resources granted by purchase events.** Use the **TimestampRounding** and **TimeStampRoundingForPurchaseGrant** business parameters to specify whether this time stamp is rounded.

Note

When a charge offer's purchase, cycle, and usage start and end units are set to 1 (seconds), 2 (minutes), or 3 (hours), and the validity period is less than 24 hours, time stamps are not rounded, regardless of your system configuration. If the validity is greater than 24 hours, the cycle end time stamp is rounded for the purpose of calculating the scale to determine the cycle fee amount to charge.

To configure BRM to round timestamps to midnight:

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

   ```
   pin_bus_params -r BusParamsSubscription bus_params_subscription.xml
   ```

   This command creates an XML file named **bus_params_subscription.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_subscription.xml.out**, set **TimestampRounding** to **enabled**:

   ```
   <TimestampRounding>enabled</TimestampRounding>
   ```

> ⚠ **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_subscription.xml.out** file to **bus_params_subscription.xml**.
6. Use the following command to load your changes into the **/config/business_params** object:

   ```
   pin_bus_params bus_params_subscription.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*.

Aligning Account and Cycle Start and End Times

You can align purchase, cycle, and usage start and end times to the accounting cycle, but only if the following are true:

- You configure delayed purchase, cycle, or usage start and end times when you set up your product offerings or when you create an account.
- The delayed start and end time is a whole number, not a fraction.
- The delay is measured in cycles.
- The purchase, cycle, or usage start and end times are *not* modified when a bundle is purchased.

To align the purchase, cycle, and usage start and end times with the accounting cycle:

1. Go to *BRM_home***/sys/data/config**.
2. Create an XML file from the **/config/business_params** object:

   ```
   pin_bus_params -r BusParamsBilling bus_params_billing.xml
   ```
3. In the XML file, set the following entry to **enabled**:

   ```
   <CycleDelayAlign>enabled</CycleDelayAlign>
   ```

> **Note**
>
> If the entry is set to **disabled** or not set, the start and end times are not aligned.

4. Load the XML file into the BRM database:

```
pin_bus_params bus_params_billing.xml
```

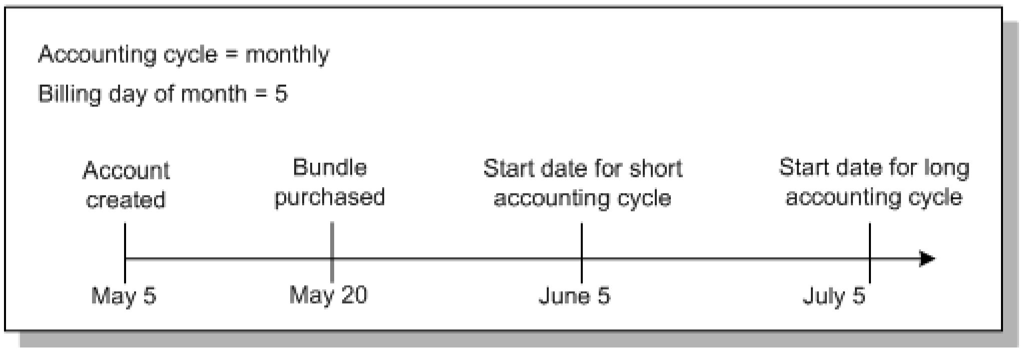
5. Stop and restart the CM.

The delayed purchase, cycle, or usage start time is set to the accounting cycle start date.

For example, if you create a customer account on May 5 and the accounting cycle is monthly, the billing DOM is set to the 5th of each month by default. If you configured the cycle start delay for 1 cycle, the customer purchases a bundle on May 20, and the accounting cycle is short, the charges begin on June 5. If the accounting cycle is long, the charges begin on July 5.

Figure 5-3 shows the cycle start time for the above example:

Figure 5-3 Aligning Account and Cycle Start and End Times


Accounting cycle = monthly
Billing day of month = 5
Account created
Bundle purchased
Start date for short accounting cycle
Start date for long accounting cycle
May 5
May 20
June 5
July 5

Defining When Billing-Time Discounts Are Applied

BRM performs accounting operations, such as applying cycle fees, rollover, and billing-time discounts, at the end of every accounting cycle. You can configure BRM to apply billing-time discounts at the end of the billing cycle instead of the accounting cycle. You might do this, for example, when the billing cycle spans multiple accounting cycles and a billing-time discount is based on the aggregated usage for the billing cycle.

> **Note**
>
> This configuration applies only to regular billing, not to bill-now and on-purchase billing.

To enable this feature, run the **pin_bus_params** utility to change the **BillTimeDiscountWhen** business parameter. For information about this utility, see *BRM Developer's Guide*.

To apply billing-time discounts at the end of the billing cycle:

1. Go to *BRM_home*/**sys/data/config**.
2. Create an XML file from the **/config/business_params** object:

   ```
   pin_bus_params -r BusParamsSubscription bus_params_subscription.xml
   ```
3. In the XML file, change **disabled** to **enabled**:

   ```
   <BillTimeDiscountWhen>enabled</BillTimeDiscountWhen>
   ```
4. Save the file as **bus_params_billing.xml**.
5. Load the XML file into the BRM database:

   ```
   pin_bus_params bus_params_billing.xml
   ```
6. Stop and restart the CM.

Note

By default, if the discount start/end dates fall exactly on accounting cycle dates, then the discount validity dates will not be altered based on its configured discount validity rules. However this behavior can be controlled using the business param **apply_validity_discount_rules**.

For example, you create an account and buy a cycle-forward product with a full discount on January 1. On February 1, BRM generates a bill for January and February, applying the discount to both cycles. If you cancel the discount on February 1, BRM will either refund the discount for February (if **apply_validity_discount_rule**s is set to 0) or extend the discount to the end of February (if **apply_validity_discount_rules** is set to 1).

Including Previous Balances in the Current Amount Due in Open Item Accounting

When you set the accounting type to open item accounting, the total amount due on the bill is reflected in the PIN_FLD_PENDING_RECV field in the **/billinfo** object. It is calculated by using the sum of the current balance and the current nonpaying child bill unit balances: the previous balance of open items is not included. As a result, the customer's bill will not include amounts from previous bills.

Note

When you set the default accounting type to balance forward accounting, the total amount due on the bill is reflected in the PIN_FLD_TOTAL_DUE field in the **/bill** object. It is calculated by using the sum of the previous balance, the current balance, and the current nonpaying child bill unit balances.

You can configure BRM to include the previous total amount due (PIN_FLD_PREVIOUS_TOTAL field) in the total amount due of the current bill unit during open item accounting. This will cause the current bill to reflect the total open charges on an account.

1. Open the CM configuration file (*BRM_home*/**sys/cm/pin.conf**) in a text editor.
2. Change the value of the **open_item_actg_include_prev_total** entry.

The values are:

- **0**: The previous total is not added to the pending amount due during open item accounting.
- **1**: The previous balance is added to the pending amount due during open item accounting.

3. Save the file.
4. Stop and restart the CM.

Specifying Which Billing Cycle to Assign to Deferred Purchase Fees

You can assign deferred purchase fees to the previous billing cycle or to the next billing cycle. By default, the purchase fee is assigned to the next billing cycle.

1. Open the CM configuration file (*BRM_home*/**sys/cm/pin.conf**) in a text editor.
2. Change the value of the **purchase_fees_backcharge** entry.

   The values are:

   - **0**: The purchase fees apply to the next cycle.
   - **1**: The purchase fees apply to the previous cycle.
3. Save the file.
4. Stop and restart the CM.

Calculating Cycle Fees for Backdating

By default, cycle fees are calculated by using the date that the current accounting cycle ends.

To handle cases where a charge offer's purchase date has been backdated, you can use the CM configuration file **calc_cycle_from_cycle_start_t** entry to calculate fees based on the charge offer's purchase date. This feature is useful when activating an inactive charge offer.

> **Note**
>
> If the cycle start time is not aligned with the billing DOM, the cycle start time is first aligned with the billing DOM before it is used to calculate the cycle charges for the charge offer. However, the cycle start time is aligned only after short and long billing cycle differences are considered.

To set the charge offer cycle start time:

1. Open the CM configuration file (*BRM_home*/**sys/cm/pin.conf**) in a text editor.
2. Edit the **calc_cycle_from_cycle_start_t** entry:

   ```
   - fm_bill calc_cycle_from_cycle_start_t 1
   ```

   - **0** retains the default BRM behavior to calculate cycle fees (based on the date specified in the PIN_FLD_ACTG_NEXT_T field).

- **1** sets the charge offer cycle start time to consider the date specified in the PIN_FLD_CYCLE_START_T field for calculating the cycle fees.

3. Save the file.

You do not need to restart the CM to enable this entry.

6 Configuring Bill Now

Learn how to use Bill Now to create a bill at any time in Oracle Communications Billing and Revenue Management (BRM).

Topics in this document:

About Bill Now

To create a bill at any time, use the Bill Now feature in Billing Care or Customer Center. Bill Now generates a bill that includes all pending items for a bill unit. Bill Now adds the previous total amount to the current total. For example, for an account that has a previous total of $20, and a current total of $10, when Bill Now is run, the account is billed for $30. Bill Now works with both open item and balance forward accounting types.

- When an account has multiple bill units, only one parent bill unit can be processed at a time. If you run Bill Now on a parent bill unit, a bill is created that contains a total of the items from both the parent and any nonpaying bill unit objects.
- If you run Bill Now on a nonpaying bill unit, a bill is created for the parent bill unit that includes only the nonpaying bill unit items.
- You can choose to bill all pending items or select specific items. When a bill is generated for specific items, it does not include the cycle arrears and cycle forward arrears fees.
- By default, Bill Now ignores the **CycleTaxInterval** business parameter, which specifies whether deferred taxes are calculated separately for the parent and each nonpaying child bill unit or are consolidated into a single item for the parent. When you run Bill Now on a parent bill unit that includes nonpaying child bill units, it rolls activities for each nonpaying child bill unit into the parent bill unit and calculates taxes for the parent only. The single tax item for the parent includes taxes from both the parent and the nonpaying child bill units. To configure Bill Now to generate a bill specifically for a nonpaying child bill unit and to calculate its taxes, see "Configuring Bill Now for Nonpaying Child Bill Units".

Providing Discounts to Closed Accounts When Using Bill Now

To apply discounts with Bill Now to closed accounts, you must ensure that BRM does not delete canceled discounts. For information about deleting canceled discounts, see "Deleting Canceled Discount Offers" in *BRM Managing Customers*.

To turn off automatic deletion of canceled discounts:

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

   ```
   pin_bus_params -r BusParamsSubscription bus_params_subscription.xml
   ```

   This command creates an XML file named **bus_params_subscription.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_subscription.xml.out**, set **KeepCancelledProductsOrDiscounts** to **enabled**:

   ```
   <KeepCancelledProductsOrDiscounts>enabled</KeepCancelledProductsOrDiscounts>
   ```

   This is the default value.

   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_subscription.xml.out** file to **bus_params_subscription.xml**.
6. Use the following command to load your changes into the **/config/business_params** object:

   ```
   pin_bus_params bus_params_subscription.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 Connection Manager (CM).

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

Prorating Cycle Arrears and Cycle Forward Arrears for Bill Now

By default, when you use Bill Now, cycle arrears charges and cycle forward arrears charges are not prorated.

You can specify to prorate cycle arrears charges and cycle forward arrears charges for Bill Now by running the **pin_bus_params** utility to change the **ApplyCycleFeeForBillNow** business parameter. For information about this utility, see *BRM Developer's Guide*.

> **Note**
>
> When you use Bill Now, you can enable proration for cycle arrears and cycle forward arrears charges; however, cycle forward fees are not prorated.

To prorate cycle forward arrears charges when you use Bill Now:

1. Go to *BRM_home*/**sys/data/config**.
2. Create an XML file from the **/config/business_params** object:

   ```
   pin_bus_params -r BusParamsBilling bus_params_billing.xml
   ```
3. In the XML file, change **disabled** to **enabled**:

   ```
   <ApplyCycleFeeForBillNow>enabled</ApplyCycleFeeForBillNow>
   ```
4. Save the file as **bus_params_billing.xml**.
5. Load the XML file into the BRM database:

   ```
   pin_bus_params bus_params_billing.xml
   ```
6. Stop and restart the CM.

Running Bill Now to Create Two Bills during the Delayed Billing Period

By default, generating two bills with Bill Now during the delayed billing period is disabled in BRM.

To enable this feature, run the **pin_bus_params** utility to change the **CreateTwoBillNowBillsInDelay** business parameter. For information about this utility, see *BRM Developer's Guide*.

Delayed billing must already be set up before enabling this feature. If you have not already set up delayed billing, see "Setting Up Delayed Billing".

To enable Bill Now during the delayed period:

1. Go to *BRM_home*/**sys/data/config**.
2. Create an XML file from the **/config/business_params** object:

   ```
   pin_bus_params -r BusParamsBilling bus_params_billing.xml
   ```
3. In the XML file, change **disabled** to **enabled**:

   ```
   <CreateTwoBillNowBillsInDelay>enabled</CreateTwoBillNowBillsInDelay>
   ```
4. Save the file as **bus_params_billing.xml**.
5. Load the XML file into the BRM database:

   ```
   pin_bus_paramsbus_params_billing.xml
   ```
6. Stop and restart the CM.

Customizing Bill Now

You can customize the following:

- The due date for a bill created with Bill Now is calculated as the billing cycle length minus one day after Bill Now is run. To change the Bill Now due date, customize the PCM_OP_BILL_POL_CALC_PYMT_DUE_T policy opcode.
- Bill Now generates a bill that includes all pending items. To change the default behavior, edit the search criteria in the PCM_OP_BILL_POL_GET_PENDING_ITEMS policy opcode.
- To apply discounts or folds when using Bill Now, use the PCM_OP_BILL_MAKE_BILL_NOW opcode.
- To generate a Bill Now type of bill for a specific service, use PCM_OP_BILL_MAKE_BILL_NOW, PCM_OP_BILL_CREATE_SPONSORED_ITEMS, and PCM_OP_BILL_POL_GET_PENDING_ITEMS.

See *BRM Opcode Guide*.

Configuring Bill Now for Nonpaying Child Bill Units

By default, Bill Now generates bills and calculates taxes at the parent level. This means that when you run Bill Now on a nonpaying child bill unit, it creates a bill for the parent bill unit that includes only the items from the nonpaying child's bill unit.

To configure Bill Now to generate a bill specifically for a nonpaying child bill unit and to calculate its taxes:

1. Configure Bill Now to generate bills at the child level by changing the **GenerateBillForChildBillNow** business parameter to **enabled**. For more information, see "Enabling Bill Generation for Nonpaying Child Accounts".
2. Configure BRM to calculate taxes for each individual nonpaying child bill unit by setting the **CycleTaxInterval** business parameter to **accounting**. For more information, see "Deferred Taxation Limitations for Hierarchical Bill Units" in *BRM Designing and Generating Invoices*.
3. Run Bill Now on the nonpaying child bill unit.

Enabling Bill Generation for Nonpaying Child Accounts

To generate bills at the child level using Bill Now, you must enable bill generation for nonpaying child accounts.

To enable generation of bills for nonpaying child accounts:

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

   ```
   pin_bus_params -r BusParamsBilling bus_params_billing.xml
   ```

   This command creates an XML file named **bus_params_billing.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_billing.xml.out**, set **GenerateBillForChildBillNow** to **enabled**:

   ```
   <GenerateBillForChildBillNow>enabled</GenerateBillForChildBillNow>
   ```

> ⚠ **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_billing.xml.out** file to **bus_params_billing.xml**.
6. Use the following command to load your changes into the **/config/business_params** object:

   ```
   pin_bus_params bus_params_billing.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*.

Configuring Tax Calculation for Nonpaying Child Accounts

By default, Bill Now calculates taxes for nonpaying child accounts at the parent account level using the parent's ship-to location. This means that when you run Bill Now on a parent account with nonpaying child accounts, tax is determined based on the parent's ship-to information for all related accounts.

To calculate taxes for each nonpaying child account separately based on the child's ship-to location:

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

   ```
   pin_bus_params -r BusParamsBilling bus_params_billing.xml
   ```

   This command creates an XML file named **bus_params_billing.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_billing.xml.out**, set **SplitCycleTaxForParentBillNow** to **enabled**:

   ```
   <SplitCycleTaxForParentBillNow>enabled</SplitCycleTaxForParentBillNow>
   ```

> **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_billing.xml.out** file to **bus_params_billing.xml**.
6. Use the following command to load your changes into the **/config/business_params** object:

   ```
   pin_bus_params bus_params_billing.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 Connection Manager (CM).

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

7
Setting Up Delayed Billing

Learn how to set up delayed billing in Oracle Communications Billing and Revenue Management (BRM).

Topics in this document:

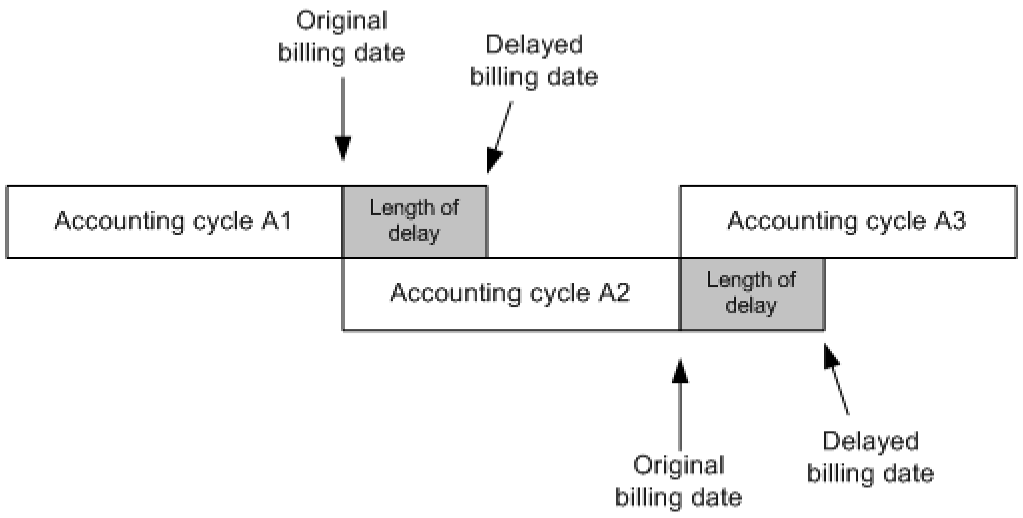
About Delayed Billing

You can set up BRM to delay billing for accounts after the end of a billing cycle. This is called *delayed billing*. Delayed billing essentially extends a billing cycle by the delay interval. You use delayed billing to bill for events that occur within a billing cycle but are not recorded during that cycle.

For example, if a batch of events does not arrive until after the end of the billing cycle, you delay billing until all events in the batch are recorded in the BRM database. When you use delayed billing, billing for all the accounts in your BRM system is delayed for the same amount of time; you cannot specify multiple billing delay periods.

Figure 7-1 shows that with delayed billing, the billing date occurs after the original billing date, during the next accounting cycle.

Figure 7-1 Delayed Billing Timeline



Note

The length of the delay interval must be shorter than one accounting cycle.

When your system is set up to use delayed billing, an account is created with two pending bills, one for the current cycle and one for the next cycle, which are combined in the balance seen in Billing Care or Customer Center. The combined pending bill includes separate items for the previous accounting cycle and for the next accounting cycle. When the bill for the current cycle is finalized at the end of the delay interval, the system makes the bill for the next cycle to be the current bill and creates a new bill for the next cycle.

The BRM system automatically triggers billing inside the delay interval when it detects that a *new* event has occurred for the *next* billing cycle. When billing is triggered during the delayed billing period, the bill for the previous cycle is partially processed (partial billing), but the bill is not finalized.

BRM performs partial billing to enable the new event to be rated and applied to the correct billing period. Partial billing ensures that new events impact bill items of the next billing cycle and old events impact bill items of the previous billing cycle. BRM maintains an internal list of bill items for the previous and next bill cycles. The bill for the previous cycle is finalized (final billing) after the delay interval.

During partial billing, BRM does the following:

- Applies deferred cycle forward fees to the next billing cycle.
- Applies cycle arrears fees to the previous billing cycle.
- Applies cycle forward fees.
- Applies cycle rollovers.

During final billing, BRM does the following:

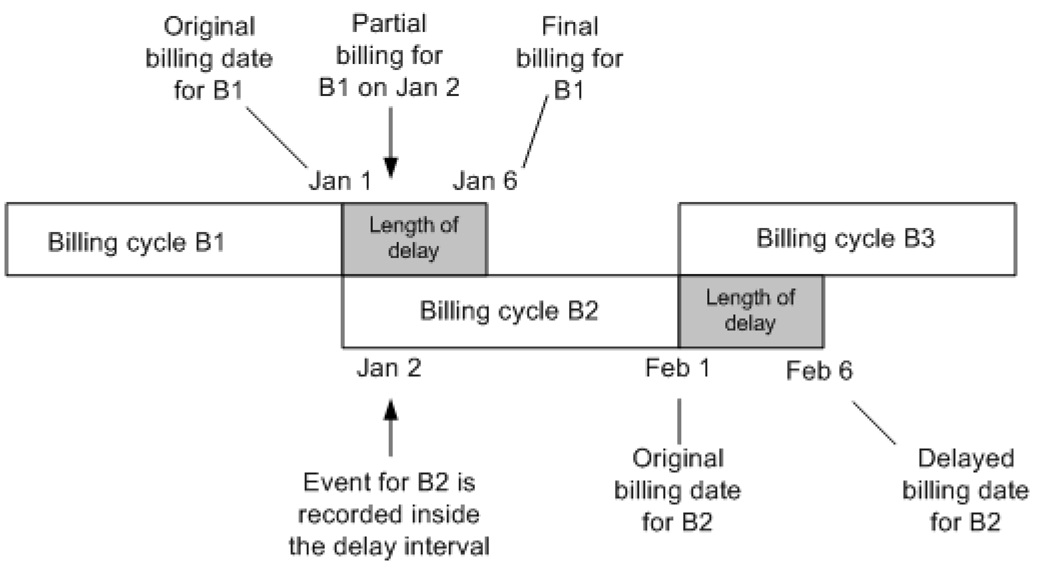
- Applies cycle discounts (billing time discounts).
- Applies cycle folds.

- Applies cycle taxes.
- Calculates a **/bill** object for the previous billing cycle.
- Initializes the *next* billing cycle.
- Creates a new empty **/bill** object for the next billing cycle with default and pre-created items.

The following example is illustrated in Figure 7-2:

- An account's billing date is January 1 and the billing delay interval is five days.
- On January 2, a new usage event for the account occurs for the next billing cycle.
- To ensure that the new usage event impacts items in the next billing period (B2), BRM performs partial billing.
- On January 6, final billing is run for the previous billing cycle (B1) by running the billing utility; the status of all the bill items for the previous bill is changed to *open* so that they stop accumulating charges.

Figure 7-2 Partial and Final Billing Timeline


Original billing date for B1
Partial billing for B1 on Jan 2
Final billing for B1
Jan 1
Jan 6
Billing cycle B1
Length of delay
Billing cycle B3
Billing cycle B2
Length of delay
Jan 2
Feb 1
Feb 6
Event for B2 is recorded inside the delay interval
Original billing date for B2
Delayed billing date for B2

When you use delayed billing, auto-triggered billing is *disabled* for all *but* the delay interval and only the last two days of each bill unit's accounting cycle.

When a bill-triggering event occurs during the delayed billing period, BRM auto-triggers partial billing and, if final billing has not occurred before the last two days of the next billing cycle, BRM auto-triggers final billing. This ensures previous billing is run before the next billing run occurs.

For example:

- An event for the next billing cycle is recorded after the billing delay interval on January 7.
- The BRM system detects that the delay period is over but final billing for the previous billing cycle has not occurred yet.
  - If auto-triggered billing is *disabled* (the default when delayed billing is configured), BRM does not run final billing on January 7. In this case, the delay interval is virtually

extended until final billing is performed by running the billing utility or auto-triggered during the last two days of the next accounting cycle.

– If auto-triggered billing is *enabled*, BRM automatically triggers final billing for the previous billing cycle on January 7.

> **Note**
>
> * You can change auto-triggered billing to be *always enabled* when delayed billing is used by setting the **AutoTriggeringLimit** business parameter to **0**.
> * You can also change the number of days auto-triggered billing is *enabled* at the *end* of each accounting cycle.

Delayed Billing and Rollovers

If you enable delayed billing, delayed events can borrow rollover from the current cycle even if events from the current cycle have consumed the rollover. Unless you set up rerating and rollover correction, current cycle events can remain rated as free even if their rollover has been consumed by delayed events.

Changing the Billing DOM When Delayed Billing is Enabled

If you change a customer's billing DOM during the delayed billing period, BRM defers the DOM change until after the delay interval ends. This ensures that the change to the billing DOM occurs in the future billing cycle. For example, assume that a customer's billing DOM is 1 and the delay interval is 5, making the billing date for the first three months of the year to be January 1, February 1, and March 1. If on January 3 the customer changes the billing DOM to 15, BRM defers making the change until January 6 (January 1 plus 5 days). This changes the billing date for the first three months to January 1, February 1, and March 15.

> **Note**
>
> To have the billing DOM change deferred until after the delay interval, auto-triggered billing must be disabled; otherwise, the billing DOM change occurs immediately. For example, changing the billing DOM to 15 on January 3 would change the billing date to February 15.

How BRM Assigns Delayed Events to Items

In BRM, every event object is associated with a bill item. When delayed events are rated by ECE, some of the events need to be assigned to the next accounting cycle. BRM pre-creates items in the following cases:

- When an account is created.
- When you run billing.
- When a CSR uses Bill Now.
- At the end of the accounting cycle when you use delayed billing.

To choose the correct bill item, ECE can do one of the following:

- Assign the event to the current open bill item.
- Assign the event to the next open bill item.

To decide which item to apply the bill to, ECE takes into account the following dates:

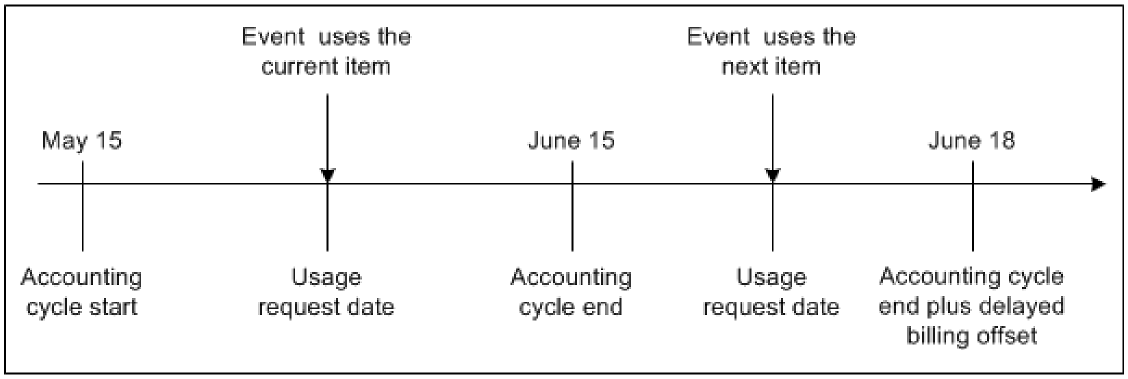
- The date when the call occurred.
- The current system date.
- The date when the current accounting cycle ends. This is called the *next accounting cycle* date.
- The number of days after the current accounting cycle ends when delayed billing runs. This number is called the *delayed billing offset*.

To assign the event to an item:

- If the event date falls before the next accounting date, the event is assigned to the current item.
- If the event date falls after the next accounting date, the event is assigned to the next item. This can happen because the event might occur after the close of the accounting cycle but before the delayed billing offset date.

Figure 7-3 shows how events are assigned:

Figure 7-3 Event Assignment

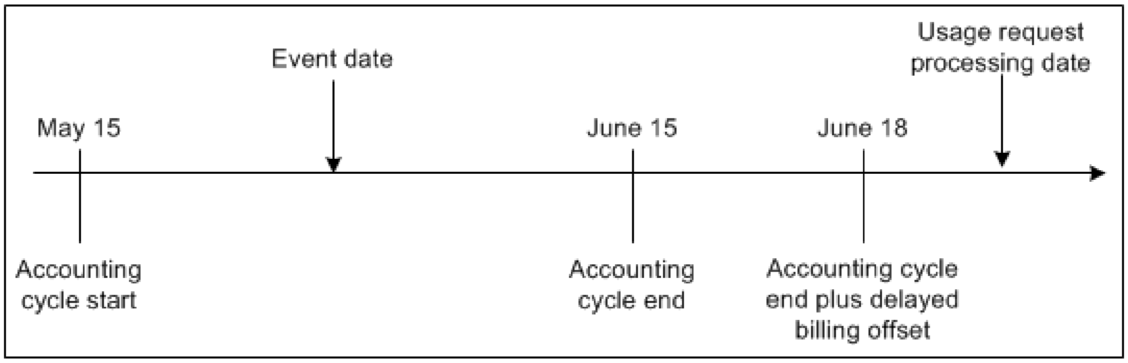


Note

The customer billing date is not relevant when choosing which item to use for the event. There might be multiple accounting cycles in one billing cycle. New items are created for each accounting cycle.

If ECE needs to assign an event to the current item, but billing for that item has already occurred, ECE includes the event and assigns it to the current item. For example, if the event is rated on May 20 and loaded after the account cycle ends, it is still included in the current item as shown in Figure 7-4.

Figure 7-4 Current Event Assignment after Billing


Usage request processing date
Event date
May 15
June 15
June 18
Accounting cycle start
Accounting cycle end
Accounting cycle end plus delayed billing offset

Configuring Delayed Billing

By default, delayed billing is enabled. You can configure delayed billing by setting the **ConfigBillingDelay** parameter using the **pin_bus_params** utility. For information about this utility, see "pin_bus_params" in *BRM Developer's Guide.*

To configure delayed billing:

1. Go to the *BRM_home***/sys/data/config** directory.
2. Create an XML file from the **/config/business_params** object:

   ```
   pin_bus_params -r BusParamsBilling bus_params_billing.xml
   ```
3. In the XML file, enter a value greater than **0**:

   ```
   <ConfigBillingDelay>D[:H]</ConfigBillingDelay>
   ```

   where *D* is the number of days and *H* is the number of hours. Leading zeros are allowed when specifying the delay interval.

   For example:

   - **0:12** sets billing delay interval to 12 hours.
   - **2** sets billing delay interval to 2 days.
   - **1:3** sets billing delay interval to 1 day and 3 hours.
   - **01:03** also sets billing delay interval to 1 day and 3 hours.
   - **0** sets the billing delay interval to zero.

   Note

   - To use delayed billing, set **ConfigBillingDelay** to a value ranging from 1 to 27. The length of the billing delay interval must be shorter than one accounting cycle.
   - If you do not want to use delayed billing, set **ConfigBillingDelay** to **0**.
   - To disable delayed billing, set this parameter to **-1**. This is supported for backward compatibility.

Caution

You can change the value of **ConfigBillingDelay** at any time. However, after you begin rating events in a production database, *do not* reset this parameter. Doing so might cause database errors.

4. Load the XML file into the BRM database:

   ```
   pin_bus_params bus_params_billing.xml
   ```

5. Stop and restart the CM.

Configuring Auto-Triggered Billing for Delayed Billing

When a systemwide billing delay is set in BRM, by default auto-triggered billing is disabled for all but the delay period and only the last two days of each bill unit's accounting cycle.

You can specify auto-triggered billing to be enabled for more than two days at the end of each accounting cycle, or you can specify that it is *always enabled* when delayed billing is used.

To change this feature, run the **pin_bus_params** utility to change the **AutoTriggeringLimit** business parameter. For information about this utility, see *BRM Developer's Guide*.

Note

This is a systemwide setting; it applies to the accounting cycle of *every* bill unit in your BRM system.

Configure the auto-triggered billing period for delayed billing as follows:

1. Go to *BRM_home*/**sys/data/config**.
2. Create an XML file from the **/config/business_params** object:

   ```
   pin_bus_params -r BusParamsBilling bus_params_billing.xml
   ```

3. In the XML file, change the number of days for which auto-triggered billing is enabled at the *end* of each accounting cycle. For example, if you change the value to **10**, auto-triggered billing is enabled for the last 10 days of every accounting cycle and in the billing delay interval.

   To *always enable* auto-triggered billing when delayed billing is used, change **2** to **0**. For example, if billing delay interval is five days and **AutoTriggeringLimit** is **0**, auto-triggered billing is enabled all the time.

   ```
   <AutoTriggeringLimit>0</AutoTriggeringLimit>
   ```

4. Save the file as **bus_params_billing.xml**.
5. Load the XML file into the BRM database:

   ```
   pin_bus_paramsbus_params_billing.xml
   ```

6. Stop and restart the CM.

Configuring an Accounting Cycle Delay Period

In offline charging, the following scenario can occur:

1. The event is rated by Offline Mediation Controller.
2. Before the event is loaded by Rated Event Loader (RE Loader), billing is run.
3. RE Loader must find the item that the event applies to and apply the balance impact.

Delayed billing assigns delayed events to items of the *billing cycle* in which they occurred. However, in the scenario described above, you might need to assign the delayed events to the *accounting cycle* in which they occurred. To assign delayed events to items of the accounting cycle in which they occurred, you configure an accounting cycle delay period. You can do this to enable accurate general ledger reporting.

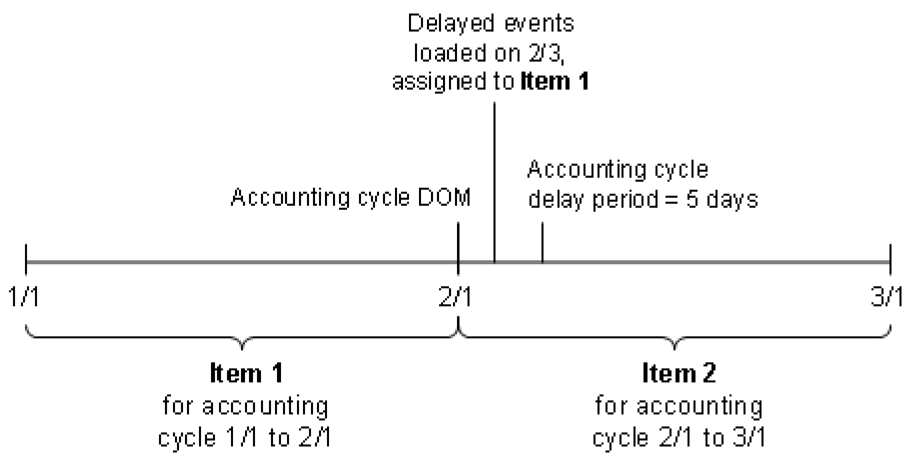
When a billing cycle spans multiple accounting cycles, the items for those accounting cycles are not closed until billing is run. If you run a general ledger (G/L) report at the end of an accounting cycle for which billing has not yet been run, the status of the revenue in the G/L can change if additional events are rated and loaded for that cycle before the accounts are billed.

If you require that G/L data not change after the G/L report is run, you can configure an accounting cycle delay period after which events are no longer assigned to items of that accounting cycle, even if those items are not closed. You then run G/L reports after the accounting cycle delay period has ended. This ensures that the revenue reported in the G/L is accurately represented and that the state of the revenue (earned, unearned, billed, and unbilled) does not change after the G/L report is run.

When you configure an accounting cycle delay period, BRM assigns delayed events to items based on when the accounting cycle delay period ends. BRM assigns events to items when RE Loader loads the events into the database:

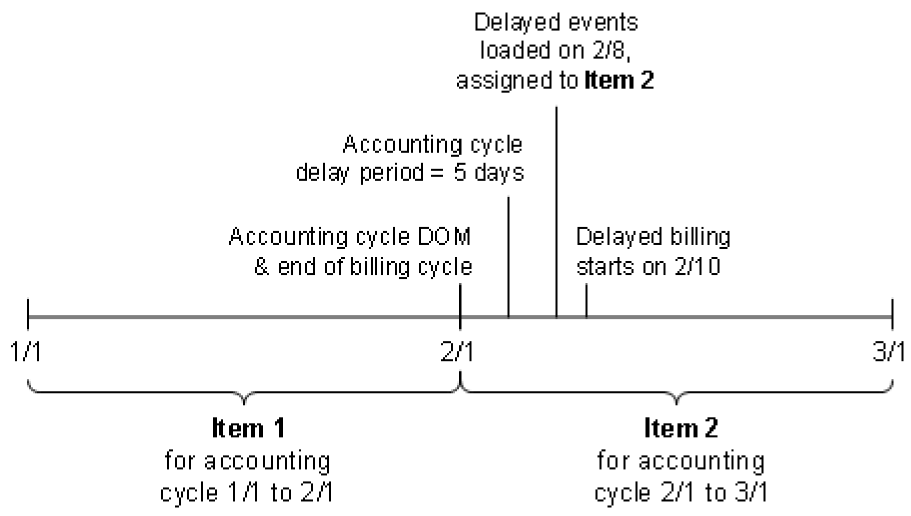
- When delayed events (events that occurred in the previous cycle) are loaded after the accounting day of month (DOM), but before the end of the accounting cycle delay period, those events are posted to the item for which the DOM has just passed as shown in Figure 7-5:

Figure 7-5 Delayed Events Arriving During Cycle Delay Period



- When the billing cycle has ended and delayed events are loaded after the end of the accounting cycle delay period, but before delayed billing is run, those events are posted to the item for the next (current) accounting cycle, even though the previous cycle has not been billed and its items are still pending. This is shown in Figure 7-6:

Figure 7-6 Delayed Events Arriving After Cycle Delay Period



- After the account is billed, items for the billed cycle are closed so delayed events are posted to the item for the following (the current) cycle.

> **Note**
>
> If the accounting cycle delay period is longer than the delayed billing period, the accounting cycle delay period is ignored after billing is run. After billing is run, if any remaining events that occurred in the previous cycle are rated and loaded in the current cycle, they are assigned to the current cycle's item.

You specify the accounting cycle delay period by running the **pin_bus_params** utility to change the **AcctCycleDelayPeriod** business parameter. For information on this utility, see *BRM Developer's Guide*.

To configure an accounting cycle delay period:

1. Go to *BRM_home***/sys/data/config**.
2. Create an XML file from the **/config/business_params** object:

   ```
   pin_bus_params -r BusParamsBilling bus_params_billing.xml
   ```

3. In the XML file, change **-1** to the number of days in the accounting cycle delay period. The number of days must be a positive value. (A value of **-1** indicates that there is no accounting cycle delay period.):

   ```
   <AcctCycleDelayPeriod>3</AcctCycleDelayPeriod>
   ```

> **Note**
>
> Do not set the accounting cycle delay period to be longer than the delayed billing period.

4. Load the XML file into the BRM database:

   ```
   pin_bus_params bus_params_billing.xml
   ```

You do not need to restart the CM to update this entry.

Specifying When to Apply Cycle Forward Fees and Cycle Rollovers

Cycle forward fees and cycle rollovers are normally applied at the beginning of the accounting cycle to charge for services provided during that cycle and to rollover unused balances for use in subsequent cycles. However, when your system is set up for delayed billing, cycle forward fees and cycle rollovers are applied during partial billing by default.

When you use delayed billing, the BRM system provides the flexibility to specify when to charge cycle forward fees and when to rollover balances. You can specify to charge cycle forward fees and rollover balances during either partial billing or final billing by setting the **delay_cycle_fees** entry in the CM configuration file (**pin.conf**).

Note

New events that occur inside the billing delay interval are rated and recorded for the next billing cycle. If cycle forward fees and rollover balances are not applied when new events occur in the delay interval, rating of the new events might produce incorrect results. Oracle recommends applying cycle forward fees and cycle rollovers during partial billing unless reasons exist not to do so.

To specify when to apply cycle forward fees and cycle rollovers:

1. Open the CM configuration file (*BRM_home*/**sys/cm/pin.conf**) in a text editor.
2. Add the **- fm_bill delay_cycle_fees** entry and set it to **0** or **1**.
   - **0** (the default) applies cycle forward fees and cycle rollover during partial billing.
   - **1** applies cycle forward fees and cycle rollover during final billing.

   Note

   You can change the setting for **delay_cycle_fees** either before partial billing or after final billing. Do not change this setting between partial billing and final billing.
3. Save the file.
4. Stop and restart the CM.

Enforcing Partial Billing in the Billing Delay Interval

Partial billing is run only when your BRM system is set up for delayed billing. The BRM system automatically triggers partial billing by default when it detects that a new event has occurred for the *next* billing cycle inside the billing delay interval.

When there are no new events in the delay interval and partial billing has not occurred, you can force the BRM system to run partial billing when the billing utility is run in the delay interval. Later, if a new event occurs in the delay interval, the new event is processed immediately, without waiting for the partial billing run to complete.

To force partial billing:

1. Open the billing utility configuration file (*BRM_home*/**apps/pin_billd/pin.conf**) in a text editor.
2. Set the **- pin_bill_accts enforce_billing** entry to **0** or **1**.

   0 does not enforce partial billing.

   1 enforces partial billing. This is the default.

   > **Note**
   >
   > The **enforce_billing** entry is used by the BRM system to enforce partial billing only if the **ConfigBillingDelay** business parameter is set to a number greater than zero. See "Configuring Delayed Billing" for more information.

3. Save the file.
4. Run the billing utility.

Setting Delayed Cycle Start Dates to the 29th, 30th, or 31st

By default, when you delay a customer's cycle fees by one month: for example, to provide a promotional month of free service: BRM sets the delayed cycle start date to any date from the 1st through the 28th of the month. Therefore, any delayed cycle fees due on the 29th, 30th, or 31st of the month are advanced to the first day of the *following* month. For example, if you delay cycle fees by one month for a bundle purchased on October 29, BRM sets the delayed cycle start date to December 1.

To configure BRM to enable delayed cycle start times on the 29th, 30th, or 31st of a month:

1. Open the CM configuration file (*BRM_home*/**sys/cm/pin.conf**) in a text editor.
2. Change the **fm_bill cycle_delay_use_special_days** entry:
   - To set the delayed cycle start date to the 1st of the following month for all bundles purchased on the 29th, 30th, or 31st, enter **0**. This is the default setting.
   - To enable BRM to assign delayed cycle start dates to the 29th, 30th, or 31st of the month, enter **1**.
3. Save the file.
4. Stop and restart the CM.

Billing Cycle Override for Delayed Billing

You can override the billing cycle for events that occur during the delayed billing interval. By default, events recorded during the delayed billing interval are billed in the previous billing cycle when the event time precedes the previous billing cycle end date. Otherwise, the event is billed in the current billing cycle. You can configure BRM to specify whether such events are billed in the previous or current billing cycle. BRM enables you to specify a configurable billing cycle interval. You can then choose which events recorded during this interval are to be billed in the previous or current billing cycle. Events that are not recorded during this interval are billed as usual, using the default delayed billing implementation.

To configure the billing cycle for events that occur during the delayed billing interval:

1. Go to the *BRM_home***/sys/data/config** directory.
2. Create an XML file from the **/config/business_params** object:

   ```
   pin_bus_params -r BusParamsRerate bus_params_billing.xml
   ```

   This command creates an XML file named **bus_params_billing.xml.out** in your working directory.
3. Open the **bus_params_billing.xml.out** file.
4. Set the **ConfigBillingCycle** parameter to the amount of time after a billing cycle ends that new events are included in the previous month's bill:

   ```
   <ConfigBillingCycle>days</ConfigBillingCycle>
   ```

   where *days* represent the number of days after a billing cycle ends. For example, enter 5 for 5 days.

   Note

   - The **ConfigBillingCycle** value must be greater than zero and less than or equal to the delayed billing interval value (set in the **ConfigBillingDelay** business parameter). Otherwise, BRM reports an error and terminates the CM. For information about setting the delayed billing interval, see "Configuring Delayed Billing".
   - Setting the configurable billing cycle to be the same as the delayed billing interval will affect system performance because each event occurring within the delayed billing interval is passed to the PCM_OP_ACT_POL_CONFIG_BILLING_CYCLE policy opcode for additional processing.

5. Save and rename the file as **bus_params_billing.xml**.
6. Load the XML file into the BRM database:

   ```
   pin_bus_params bus_params_billing.xml
   ```
7. Stop and restart the CM.
8. Modify the PCM_OP_ACT_POL_CONFIG_BILLING_CYCLE policy opcode:
   - To bill the event in the previous cycle, set the output flist field FLAGS to BILL_IN_PREVIOUS_CYCLE.
   - To bill the event in the current cycle, set the FLAGS field to BILL_IN_CURRENT_CYCLE.

   See "Customizing How to Bill Events That Occur between Billing Cycles" in *BRM Opcode Guide*.

8 Configuring 31-Day Billing

Learn how to enable Oracle Communications Billing and Revenue Management (BRM) to use all 31 days as the billing day of the month (DOM).

Topics in this document:

- About Using 31-Day Billing
- Enabling the 31-Day Billing Feature
- Disabling the 31-Day Billing Feature
- Setting the Forward and Back Billing Options

About Using 31-Day Billing

By default, you can set the billing day of the month (DOM) to any day between 1 and 28. If your customer signs up on the 29th, 30th, or 31st, the billing DOM is set to the 1st. This can result in a large number of customers being billed on the 1st of the month.

You can change this setting to support billing on all days of the month. For example, if you create a customer account on the 29th, the billing DOM is set to the 29th instead of the 1st.

If your customers' billing DOM is the 29th, 30th, or 31st, for the months that do not have these days, you can configure whether billing should be run on the last day for the same month (*set to back* option) or the first day of the next month (*set to forward* option). By default, the billing DOM is set *forward* to the 1st of the next month.

For example, when the DOM is set to 31 because April has 30 days:

- The set to forward option sets the billing date to May 1.
- The set to back option sets the billing date to April 30.

Enabling the 31-Day Billing Feature

To use 31-day billing, you must either modify the **init_objects.source** file before loading it into the database or set the **Billing31Day** business parameter using the **pin_bus_params** utility. After BRM is configured to allow 31-day billing, you can specify whether to use the back or forward option.

Switching to 31-Day Billing During BRM Installation

Before loading **init_objects.source**, change the value of the PIN_FLD_MAXIMUM field from **28** to **31** in the **/config/fld_validate** object that has the **Actg_cycle** value in the PIN_FLD_NAME field as follows:

```
# /config/fld_validate  - Actg_cycle validation
<PCM_OP $PIN_OPNAME=$PIN_CONF_INIT_OPNAME; $PIN_OPFLAGS=$PIN_CONF_INIT_OPFLAGS>
0 PIN_FLD_POID                 POID [0] $PIN_CONF_DB_NO /config/fld_validate 606 0
0 PIN_FLD_DESCR                 STR [0] "Field Validation"
0 PIN_FLD_HOSTNAME              STR [0] "-"
```

```
0 PIN_FLD_NAME                 STR [0] "Actg_cycle"
0 PIN_FLD_PROGRAM_NAME         STR [0] "-"
0 PIN_FLD_VALIDATION     SUBSTRUCT [0]
1 PIN_FLD_FIELD_TYPE           INT [0] 2
1 PIN_FLD_MAXIMUM              NUM [0] 31
1 PIN_FLD_MINIMUM              NUM [0] 31
</PCM_OP>
```

Note

When you upgrade to a new BRM release, ensure that you make this change in the new **init_objects.source** file. The installation program overwrites the **init_objects.source** file and the changes you have made will be lost.

Switching to 31-Day Billing After You Install BRM

To switch to 31-day billing after you have installed BRM:

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

   ```
   pin_bus_params -r BusParamsBilling bus_params_billing.xml
   ```

   This command creates an XML file named **bus_params_billing.xml.out** in your current directory. Use care when updating parameters in the file.
3. In **bus_params_billing.xml.out**, set **Billing31Day** to **enabled**:

   ```
   <Billing31Day>enabled</Billing31Day>
   ```

   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_billing.xml.out** file to **bus_params_billing.xml**.
6. Use the following command to load your changes into the **/config/business_params** object:

   ```
   pin_bus_params bus_params_billing.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 Connection Manager (CM).

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

Disabling the 31-Day Billing Feature

To disable the 31-day billing feature:

1. In the **init_objects.source** file, change the value of the PIN_FLD_MAXIMUM field from **31** to **28** in the **/config/fld_validate** object that has the **Actg_cycle** value in the PIN_FLD_NAME field as follows:

```
# /config/fld_validate  - Actg_cycle validation
<PCM_OP $PIN_OPNAME=$PIN_CONF_INIT_OPNAME; $PIN_OPFLAGS=$PIN_CONF_INIT_OPFLAGS>
0 PIN_FLD_POID                 POID [0] $PIN_CONF_DB_NO /config/fld_validate 606 0
0 PIN_FLD_DESCR                 STR [0] "Field Validation"
0 PIN_FLD_HOSTNAME              STR [0] "-"
0 PIN_FLD_NAME                  STR [0] "Actg_cycle"
0 PIN_FLD_PROGRAM_NAME          STR [0] "-"
0 PIN_FLD_VALIDATION      SUBSTRUCT [0]
1 PIN_FLD_FIELD_TYPE            INT [0] 2
1 PIN_FLD_MAXIMUM               NUM [0] 28
1 PIN_FLD_MINIMUM               NUM [0] 31
</PCM_OP>
```

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

```
pin_bus_params -r BusParamsBilling bus_params_billing.xml
```

   This command creates an XML file named **bus_params_billing.xml.out** in your current directory. Use care when updating parameters in the file.
4. In **bus_params_billing.xml.out**, set **Billing31Day** to **disabled**:

```
<Billing31Day>disabled</Billing31Day>
```

   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.

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

```
pin_bus_params bus_params_billing.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*.
8. 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*.

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

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

Setting the Forward and Back Billing Options

To set the forward and back billing options:

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

   ```
   pin_bus_params -r BusParamsBilling bus_params_billing.xml
   ```

   This command creates an XML file named **bus_params_billing.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_billing.xml.out**, set the following entry:

   ```
   <MoveDayForward>value</MoveDayForward>
   ```

   where *value* is one of the following:

   - **firstDay**: Sets the billing date to the first day of the month. This is the default.
   - **lastDay**: Sets the billing date to the last day of the month.

   Caution

   BRM uses the XML in this file to overwrite the existing instance of the **/config/business_params** object. Use care when updating parameters in the file.

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

   ```
   pin_bus_params bus_params_billing.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 Connection Manager (CM).

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

9

Configuring the Billing Cutoff Time

Learn how to configure Oracle Communications Billing and Revenue Management (BRM) to use a billing cutoff time other than midnight.

Topics in this document:

- About Configuring the BRM Cutoff Time
- Configuring the Billing Cutoff Time

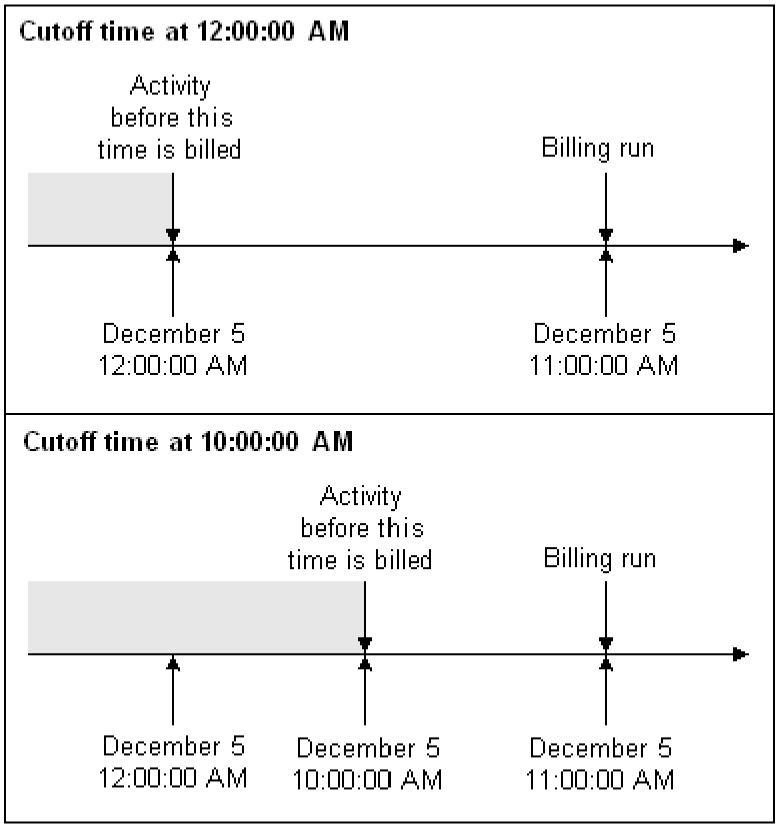
About Configuring the BRM Cutoff Time

By default, BRM defines the business day as starting at 12:00:00 a.m. and ending at 11:59:59 p.m. For example, if you run billing at any time on December 5, billing is performed for all activity that occurred until 11:59:59 p.m. on December 4 for the accounts to be billed.

You can change the cutoff time to start your billing activity at any time of the day. For example, if you set the cutoff time to 10 a.m., activity for events that occurred before 10 a.m. are billed.

Figure 9-1 shows how billing works for different cutoff times:

Figure 9-1 Billing Cutoff Time



Changing the cutoff time does not just change how billing works; it changes how all activities in BRM work, including accounting and billing cycles, usage rating, cycle fees, proration, general ledger posting, and searches. The cutoff time is used for all accounts.

How Billing and Invoicing Are Affected by Changing the Cutoff Time

- The start and end dates for accounting and billing cycles are based on the cutoff time. For example, if the cutoff time is 10:00 a.m., a customer who creates an account at 9:00 a.m. on December 5 has a billing date of December 4.
- The following utilities run by the **pin_bill_day** script use the cutoff time to calculate the billing periods:
  - **pin_deferred_act**
  - **pin_bill_accts**
  - **pin_collect**
  - **pin_refund**
  - **pin_inv_accts**
  - **pin_deposit**
  - **pin_cycle_fees**
- When searching for accounts, the **pin_inv_accts**, **pin_inv_send**, and **pin_inv_export** utilities use the cutoff time to calculate the start and end times for flagging accounts to be invoiced.

How Rating Is Affected by Changing the Cutoff Time

- When you define start and end times for any pricing component (for example, the start and end times for a discount), BRM uses the cutoff time. For example, if you specify that a discount is valid until December 5 and the cutoff time is 10:00 a.m., the discount is valid until 10:00 a.m. on December 5.
- You can set up special charges for events that occur on certain days. BRM uses the cutoff time to determine which day an event is assigned to.

How General Ledger (G/L) Is Affected by Changing the Cutoff Time

When searching for events for collecting G/L information and generating G/L reports, the **pin_ledger_report** utility uses the cutoff time to calculate the start and end times for the G/L report.

How Timestamp Fields Are Affected by Changing the Cutoff Time

Many BRM features use timestamps to determine how to perform activities. Timestamps are usually rounded to midnight. If you change the cutoff time, the timestamp is rounded to the cutoff time instead.

Note

The cutoff time is also considered while setting the timestamp values for product offering start and end dates.

These fields affect the accounting cycle dates:

- PIN_FLD_ACTG_LAST_T
- PIN_FLD_ACTG_NEXT_T
- PIN_FLD_ACTG_FUTURE_T

These fields affect rating and proration:

- PIN_FLD_PURCHASE_START_T
- PIN_FLD_PURCHASE_END_T
- PIN_FLD_USAGE_START_T
- PIN_FLD_USAGE_END_T
- PIN_FLD_CYCLE_START_T
- PIN_FLD_CYCLE_END_T

These fields affect billing cycle dates:

- PIN_FLD_LAST_BILL_T
- PIN_FLD_NEXT_BILL_T

Configuring the Billing Cutoff Time

Note

After you set the cutoff time, you *cannot* change it in a production system.

To configure the billing cutoff time:

1. Enable time-stamp rounding. See "Configuring Time-Stamp Rounding". You do not need to stop and restart the Connection Manager (CM) at the end of the procedure, although you may do so if you want.
2. Go to *BRM_home***/sys/data/config**.
3. Use the following command to create an editable XML file from the billing instance of the **/config/business_params** object:

   ```
   pin_bus_params -r BusParamsBilling bus_params_billing.xml
   ```

   This command creates an XML file named **bus_params_billing.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.
4. In **bus_params_billing.xml.out**, set **BillingCycleOffset** to the desired cutoff time:

   ```
   <BillingCycleOffset>value</BillingCycleOffset>
   ```

   For example, to set the cutoff time to 10:00 a.m., set the value as **10**. The default is **0** (midnight).

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.

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

   ```
   pin_bus_params bus_params_billing.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*.
8. 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*.
9. Stop and restart the CM.

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

10

Configuring Billing for Groups and Hierarchies

Learn how to configure the way Oracle Communications Billing and Revenue Management (BRM) bills sharing groups and account hierarchies.

Topics in this document:

Setting Up Billing for Charge and Discount Sharing Groups

By default, when billing is run, bill units are billed in this order:

1. All nonpaying child bill units in all accounts.
2. All remaining bill units in all accounts.

If you have discount or charge sharing groups in your BRM system, you must reconfigure your system to bill accounts in this order:

1. All nonpaying child bill units in all accounts.
2. All remaining discount group member bill units in all member accounts.
3. All remaining charge sharing group member bill units in all member accounts.
4. All remaining bill units in all accounts.

This ensures that billing is run for all member accounts before it is run for any discount or charge sharing group owner account.

Note

If you do not reconfigure your system, discount and charge sharing group owner accounts might be billed before some of their member accounts. When this occurs, the members' sponsored charges are not included in the owner's bill for the current billing cycle. Instead, they are added to the owner's bill for the next billing cycle.

To enable this feature, run the **pin_bus_params** utility to change the **BillingFlowSponsorship** business parameter. For information about this utility, see *BRM Developer's Guide*.

The following setups ensure that two-level discount and charge sharing relationships are correctly billed. For sharing relationships that exceed two levels, the billing sequence might be incorrect, resulting in incorrect billing.

For example, suppose account A owns a sharing group to which account B belongs, and account B owns a sharing group to which account C belongs. Owner account A and member account B are billed in the correct order: member before owner. Owner account B and member account C, however, might not be billed in the correct order. This happens because B is an owner of account C's group and a member of account A's group. As a member, it is billed at the same time all other member accounts are billed and thus might be billed before account C.

To set up billing for charge sharing groups:

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

   ```
   pin_bus_params -r BusParamsBilling bus_params_billing.xml
   ```

   This command creates an XML file named **bus_params_billing.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_billing.xml.out**, set the following entry:

   ```
   <BillingFlowSponsorship>value</BillingFlowSponsorship>
   ```

   where *value* is one of the following:

   - **sponsorsFirst**: Specifies to bill group owner accounts before member accounts.
   - **sponsoreesFirst**: Specifies to bill member accounts before owner accounts.
   - **undefined**: Specifies that the billing order of owner and member accounts does not matter. This is the default.

   Caution

   BRM uses the XML in this file to overwrite the existing instance of the **/config/business_params** object. Use care when updating parameters in the file.
4. Save and exit the file.
5. Rename the **bus_params_billing.xml.out** file to **bus_params_billing.xml**.
6. Use the following command to load your changes into the **/config/business_params** object:

   ```
   pin_bus_params bus_params_billing.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 Connection Manager (CM).

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

To set up billing for discount sharing groups:

1. Go to *BRM_home*/**sys/data/config**.

2. Use the following command to create an editable XML file from the **Billing** instance of the **/config/business_params** object:

   ```
   pin_bus_params -r BusParamsBilling bus_params_billing.xml
   ```

   This command creates an XML file named **bus_params_billing.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_billing.xml.out**, set the following entry:

   ```
   <BillingFlowDiscount>value</BillingFlowDiscount>
   ```

   where *value* is one of the following:

   - **discountParentsFirst**: Specifies to bill group owner accounts before member accounts.
   - **memberDiscountFirst**: Specifies to bill member accounts before owner accounts.
   - **undefined**: Specifies that the billing order of owner and member accounts does not matter. This is the default.

   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_billing.xml.out** file to **bus_params_billing.xml**.
6. Use the following command to load your changes into the **/config/business_params** object:

   ```
   pin_bus_params bus_params_billing.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 Connection Manager (CM).

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

Skipping Validation of Billing for Nonpaying Child Bill Units

If you use bill unit hierarchies, BRM validates that all nonpaying bill units have been billed successfully before billing the paying parent bill unit. When billing fails for a nonpaying child bill unit, the parent bill unit is not billed. In rare instances, however, when billing for the nonpaying child bill unit continuously fails and you want to proceed with billing the parent bill unit, you can skip validation of billing for the nonpaying child bill unit.

To enable this feature, run the **pin_bus_params** utility to change the **SkipCheckForSubordinatesBilled** business parameter. For information about this utility, see "pin_bus_params" in *BRM Developer's Guide*.

To skip validation of billing for nonpaying child bill units:

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

   ```
   pin_bus_params -r BusParamsBilling bus_params_billing.xml
   ```

   This command creates an XML file named **bus_params_billing.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_billing.xml.out**, set **SkipCheckForSubordinatesBilled** to **enabled**:

   ```
   <SkipCheckForSubordinatesBilled>enabled</SkipCheckForSubordinatesBilled>
   ```

   Caution

   BRM uses the XML in this file to overwrite the existing instance of the **/config/business_params** object. Use care when updating parameters in the file.
4. Save and exit the file.
5. Rename the **bus_params_billing.xml.out** file to **bus_params_billing.xml**.
6. Use the following command to load your changes into the **/config/business_params** object:

   ```
   pin_bus_params bus_params_billing.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 Connection Manager (CM).

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

About Changing the Billing Day of Month for Large Hierarchies

You can change the billing day of month (DOM) for parent accounts and child accounts in large hierarchies.

By default, BRM changes the billing DOM for the parent account and its child accounts as a single transaction. For accounts in a large hierarchy, the changes are first applied to the parent account followed by its child accounts. This locks the parent account, restricting any other operations during the process.

However, you can configure BRM to update the billing DOM for the parent account and its child accounts asynchronously. This is done through a multithreaded application that performs parallel updates for all accounts. It also removes the need to lock the parent account while updating the child accounts.

To change the billing DOM for large hierarchies synchronously, see "Editing a Bill Unit" in *Billing Care Online Help*.

To change the billing DOM for wholesale hierarchies asynchronously:

1. Configure the parent account to use the appropriate business profile. See "Loading the Business Profile for Changing the Billing Day of Month Asynchronously".
2. Change the billing DOM for the wholesale hierarchy. See "Changing the Billing Day of Month Asynchronously for All Accounts in Large Hierarchies".
3. Run the **pin_update_bdom** utility regularly. See "Processing Asynchronous Change in Billing Day of Month".

Loading the Business Profile for Changing the Billing Day of Month Asynchronously

Note

The change in the billing DOM takes effect from the next billing cycle.

To enable BRM for changing the billing DOM asynchronously, you need to load the **AsyncBDOMChange** business profile, or a custom profile that contains the **AsyncBDOMChange** key set to **yes**, into the BRM database. You can do this by loading the standard business profiles or editing your custom business profile to include the **AsyncBDOMChange** key set to **yes** and then loading your custom profile.

For the standard business profile:

1. Go to the directory that contains the **pin_business_profile.xml** file. By default, this file is in the *BRM_home***/sys/data/config** directory.
2. Load the **pin_business_profile.xml** file into the database by running the following command:

   ```
   load_pin_business_profile pin_business_profile.xml
   ```

For your custom business profile:

1. Open the file containing your business profiles in an XML editor or a text editor.
2. Inside your custom profile, add a new **NameValue** as follows:

   ```
   <NameValue key="AsyncBDOMChange" value="yes" />
   ```

3. Save and close the file.
4. Load the file into the database by running the following command:

   ```
   load_pin_business_profile yourFileName
   ```

   where *yourFileName* is the name of the file containing your custom business profile.

For general information about business profiles, see "Creating and Managing Business Profiles" in *BRM Managing Customers*.

Changing the Billing Day of Month Asynchronously for All Accounts in Large Hierarchies

There are several different ways to change the billing DOM asynchronously for all accounts in large hierarchies. To change the billing DOM for the parent account asynchronously, you must set its Business Profile to **AsyncBDOMChange** or to a custom profile that contains the **AsyncBDOMChange** key set to **yes**.

You can change the billing DOM for all the accounts in large hierarchies using:

- Billing Care. For more information, see "Changing the Billing Day of Month for Wholesale Billing Hierarchy" in *Billing Care Online Help*.
- A custom client application calling the PCM_OP_CUST_SET_BILLINFO opcode. See "Updating Bill Units" in *BRM Opcode Guide*.
- A custom client application calling the following Billing Care REST API endpoint:

  /bcws/webresources/v1.0/billunits/account/{id}

  For more information, see *REST API Reference for Billing Care*.

Processing Asynchronous Change in Billing Day of Month

You use the **pin_update_bdom** utility to process deferred billing DOM jobs. You can run the utility regularly by creating a **cron** job or by using Business Operations Center.

> **Note**
>
> The utility performs jobs for open and inactive accounts only.

The utility does the following:

1. Searches the **/job/bdom** object for jobs associated with the specified parent account or bill unit.

   If the utility is run without parameters, it finds **/job/bdom** jobs with a status of NEW or PARTIALLY_COMPLETE.
2. Finds all child bill units whose account receivables (A/R) field in the **/billinfo** object matches the **/billinfo** object provided for the wholesale parent.
3. Updates the job’s status to IN_PROGRESS.
4. For each bill unit, it generates a thread that calls the PCM_OP_CUST_SET_BILLINFO opcode to update the bill unit’s billing DOM.
5. Determines whether the job processed successfully.
   - If all bill units were updated successfully, it sets the job’s status to SUCCESS.
   - If any bill unit could not be updated, it sets the job’s status to ERROR.
   - If the **pin_update_bdom** utility is run with the **-count** option, it sets the job status to PARTIALLY_COMPLETE once the job is completed.

For more information on how to run the utility, see "pin_update_bdom".

If the utility runs into errors or crashes, rerun the utility with the **-retry** option to continue to process bill units in all jobs with errors. For example:

```
pin_update_bdom -retry
```

To do this for a specific job, also include the **-billinfo** option. For example:

```
pin_update_bdom -retry -billinfo POID
```

The status of the job changes based on its current stage, which is one of the following:

- NEW: Indicates that there is no change in billing DOM for any bill units so far.
- PARTIALLY_INCOMPLETE: Indicates that some of the bill units have been updated.
- IN_PROGRESS: Indicates that the change in billing DOM is in progress.
- ERROR: Indicates that the utility has run into an error.
- SUCCESS: Indicates that all the bill units have been updated successfully.

11 Configuring Wholesale Billing

Learn how to configure Oracle Communications Billing and Revenue Management (BRM) to perform wholesale billing for your large wholesale business accounts.

Topics in this document:

About Wholesale Billing

Wholesale business accounts with large account hierarchies can have a large number of services, each representing a subscription account. This can affect billing and invoicing performance if your customers' accounts have a large number of billing items to process.

You can set up wholesale billing for handling large wholesale business accounts. In wholesale billing, you set up a bill unit hierarchy for account receivable (A/R) operations. The wholesale business account is the parent account with the paying parent bill unit and the services (subscriptions) in this account are child accounts with nonpaying child bill units.

BRM consolidates the charges, discounts, A/R items, bill items, journals, and taxes across the services under the wholesale business account and performs the A/R operations, billing, and invoicing at the wholesale business account level.

Note

The currency balances for parent account and child accounts are synchronized with Oracle Communications Elastic Charging Engine (ECE) during wholesale billing at the end of a billing cycle. This information is published through the **/event/notification/billing/updateBalances** event notification. See *BRM Event Notification Reference* for more information.

Setting Up Billing for Wholesale Account Hierarchies

You set up wholesale billing for the whole system by:

1. Enabling Wholesale Billing for All Accounts
2. Creating Wholesale Accounts and Bill Unit Hierarchies

To set up wholesale billing only for specific accounts, see "Creating Wholesale Accounts and Bill Unit Hierarchies".

You can also convert existing account hierarchies to use wholesale billing. See "Converting Existing Bill Unit Hierarchies to Wholesale Billing" for more information.

Enabling Wholesale Billing for All Accounts

When wholesale billing is enabled using the business parameter, BRM lets you create only wholesale accounts and bill unit hierarchies.

To enable this feature, run the **pin_bus_params** utility to change the **WholesaleBillingSystem** business parameter. See "pin_bus_params" in *BRM Developer's Guide* for information about the utility's syntax and parameters.

To enable wholesale billing for all of your accounts:

1. Go to *BRM_home*/**sys/data/config**.
2. Create an XML file from the **/config/business_params** object:

   ```
   pin_bus_params -r BusParamsBilling bus_params_billing.xml
   ```

3. In the XML file, set the following entry to **enabled**:

   ```
   <WholesaleBillingSystem>enabled</WholesaleBillingSystem>
   ```

4. Save the file as **bus_params_billing.xml**.
5. Load the file into the BRM database:

   ```
   pin_bus_params bus_params_billing.xml
   ```

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

If you enable this parameter in a system with existing bill unit hierarchies, run the **pin_cust_convert_wholesale_hierarchy** utility to convert the existing hierarchies to use wholesale billing. See "pin_cust_convert_wholesale_hierarchy" for more information about this utility.

Creating Wholesale Accounts and Bill Unit Hierarchies

You can create accounts and bill unit hierarchies for wholesale billing by using Billing Care or by using custom applications that call BRM opcodes. You can also convert existing bill unit hierarchies to wholesale billing. For more information about accounts and bill unit hierarchies, see "Managing Account and Bill Unit Hierarchies" in *BRM Managing Customers*.

A wholesale parent account contains one default wholesale bill unit, to which charges from child bill units are rolled up. It can also contain additional bill units for handling other charges the parent makes.

You create accounts and bill unit hierarchies and set up the wholesale parent account by:

1. Loading the Wholesale Business Profile
2. Creating a Wholesale Billing Account
3. (Optional) Precreating Item Types for the Wholesale Billing Parent

You can add any existing bill unit to a wholesale bill unit hierarchy or set up a new wholesale bill unit hierarchy by using the existing bill units in BRM. However, you must ensure the following:

- There are no pending items or payments in the bill unit that you are adding to the hierarchy.
- The parent bill unit is the paying bill unit and it is set as the wholesale parent for billing.
- The wholesale parent for the wholesale bill unit hierarchy is set before creating the hierarchy.

This ensures that the charges and other billing-related items of the nonpaying child bill units in the hierarchy are rolled up to the paying parent bill unit during billing.

Loading the Wholesale Business Profile

To use wholesale billing, you must load a business profile that has wholesale billing enabled. You can do this by loading the standard business profiles or editing your custom business profile to include the **WholesaleBilling** key set to **yes** and then loading your custom profile.

For the standard business profile:

1. Go to the directory that contains the **pin_business_profile.xml** file. By default, this file is in the *BRM_home***/sys/data/config** directory.
2. Load the **pin_business_profile.xml** file into the database by running the following command:

   ```
   load_pin_business_profile pin_business_profile.xml
   ```

For your custom business profile:

1. Open the file containing your business profiles in an XML editor or a text editor.
2. Inside your custom profile, add a new **NameValue** as follows:

   ```
   <NameValue key="WholesaleBilling" value="yes" />
   ```

3. Save and close the file.
4. Load the file into the database by running the following command:

   ```
   load_pin_business_profile yourFileName
   ```

   where *yourFileName* is the name of the file containing your custom business profile.

For general information about business profiles, see "Creating and Managing Business Profiles" in *BRM Managing Customers*.

Creating a Wholesale Billing Account

To set up a wholesale billing account, you:

1. Open the account that will contain the wholesale parent bill unit.
2. Change the parent's default bill unit into a wholesale bill unit by assigning it to the wholesale business profile (in the **/config/business_profile** object).
3. Create an account and bill unit for each child under the wholesale billing parent. In the bill unit, set the payment method to the parent's wholesale billing unit.
4. Create a billing hierarchy by associating each child's bill unit with the wholesale parent's bill unit.

You can create wholesale billing hierarchies using:

- Billing Care. See "Wholesale Billing Hierarchies" in *Billing Care Online Help*.
- A custom client application calling the following customer management opcodes:
  - To change the parent's default unit into a wholesale billing parent bill unit, use one of the following opcodes:
    The PCM_OP_CUST_UPDATE CUSTOMER

    PCM_OP_CUST_CHANGE_BUSINESS_PROFILE
  - To create a wholesale child bill unit, use the PCM_OP_CUST_UPDATE_CUSTOMER opcode.
  - To change a child's payment method to the parent's wholesale bill unit and add its bill unit to the parent's billing hierarchy, use the PCM_OP_CUST_SET_BILL_INFO opcode.

  For more information, see "Customer Management Opcode Workflows" in *BRM Opcode Guide*.
- A custom client application calling the following Billing Care REST API endpoints:
  - To change the parent's default unit into a wholesale billing parent bill unit, use the Update a Bill Unit for an Account endpoint.
  - To create a wholesale child bill unit, use the Create a Bill Unit endpoint.
  - To change a child's payment method to the wholesale parent's bill unit, use the Update a Payment Method endpoint.
  - To add a child's bill unit to the parent's billing hierarchy, use the Add a Bill Unit to a Hierarchy endpoint.

  For more information, see *REST API Reference for Billing Care*.

Precreating Item Types for the Wholesale Billing Parent

During final billing, all charges (such as recurring, purchase, and usage charges) applied to the nonpaying child bill units (*wholesale child accounts*) are aggregated based on the item-tag-to-item-type mapping (*item configuration*) and are rolled up to the corresponding bill items of the paying parent bill unit (*wholesale parent account*).

If the bill item for any item type does not exist for the paying parent bill unit, the bill item is created during billing and the charges are rolled up to that item. However, Oracle recommends to pre-create the bill items for the different item types by setting the **precreate** element to **true** in the *BRM_home***/sys/data/pricing/example/config_item_types.xml** file. For more information, see "Mapping Item Tags to Item Types".

The total and due amounts of the paying parent bill unit are updated to reflect the roll-up and the due amount of each nonpaying child bill unit is set to **0**. Payments are applied only to the paying parent bill unit.

If you want the charges for different services to be rolled up to different bill items, you can assign different item types for different services. For example, for rolling up cycle forward fees for IP and GSM services, you can configure and assign the following items: **/item/ip/cycle_forward** for the IP service and **/item/gsm/cycle_forward** for the GSM service.

You can also assign a different item type (or a noncumulative bill item) to track charges specific to the paying parent bill unit. For information about assigning items, see "Assigning Item Tags Based on Event and Service Combinations".

About Rolling Journals Up to the Wholesale Parent

For nonpaying child bill units, the **/tmp_journals_to_process** objects are created instead of the **/journal** objects at the time of rating. The **/tmp_journals_to_process** objects are created only if the **CycleTaxInterval** business parameter is set to **billing**. For more information, see the discussion about tax calculation for account groups in *BRM Calculating Taxes*.

The **/tmp_journals_to_process** objects contain revenue and tax data. For wholesale billing, BRM uses these objects primarily to track and consolidate taxes for billing-time taxation. To roll up journals for wholesale billing, you use the **pin_update_journals** utility.

Note

To roll up journals for wholesale billing, you must ensure that the general ledger reporting is enabled. For more information, see the discussion about general ledger reporting in *BRM Collecting General Ledger Data*.

The **pin_update_journals** utility processes all **/tmp_journals_to_process** objects of the nonpaying child bill units and rolls them up to the corresponding **/journal** object of the paying parent bill unit.

You can run the **pin_update_journals** utility on a daily basis to ensure that the paying parent bill unit is kept up to date. However, you must run this utility once before billing the paying parent bill unit. For more information, see "pin_update_journals".

If deferred taxation is configured to:

- Consolidate taxes into a single item (if **CycleTaxInterval** is set to **billing**), the **pin_update_journals** utility enables you to roll the taxes up into a single item for both the paying parent and nonpaying child bill units. The total tax is calculated at the paying parent level for the entire hierarchy using the aggregated total due as the basis.
- Calculate taxes separately for the parent and each nonpaying child bill unit (if **CycleTaxInterval** is set to **accounting**), the **/journal** objects are created for the nonpaying child bill units instead of **/tmp_journals_to_process** objects and the taxes are not rolled up to the paying parent bill unit.

About Rolling A/R Actions Up to the Wholesale Parent

BRM manages the balances for A/R actions by using these A/R items: adjustment, dispute, settlement, payment, refund, payment reversal, write-off, and write-off reversal. See "About A/R Management" in *BRM Concepts*.

To roll up A/R actions for wholesale billing, you use the **pin_roll_up_ar_items** utility. This utility processes all temporary A/R items (**/tmp_ar_item_to_roll_up** object) for nonpaying child bill units, and rolls the balance impact up to the corresponding A/R items of the paying parent bill unit. For example, it rolls the adjustments allocated to the nonpaying child bill unit's **/item/cycle_forward** item up to the adjustment item associated with the **/item/adjustment** item of the paying parent bill unit.

You can run the **pin_roll_up_ar_items** utility on a daily basis to ensure that the A/R items of the paying parent bill unit are kept up to date. However, you must run this utility once before billing the paying parent bill unit. See "pin_roll_up_ar_items" in *BRM Managing Accounts Receivable*.

In addition, you can use the **pin_roll_up_ar_items** utility to roll up the adjustment items that are created as a result of rerating. During rerating, the temporary A/R items (**/tmp_ar_item_to_roll_up** object) are created for the nonpaying child bill units if these conditions are met:

- The event has already been billed.
- The event occurred prior to general ledger posting.
- The event is unbilled but the automatic allocation of rerating adjustments is disabled.

If the event is unbilled and the automatic allocation of rerating adjustments is enabled, the rerating adjustment is allocated to the bill item of the nonpaying child bill unit.

Note

Rerating adjustments rolled up to the paying parent bill unit are allocated to the corresponding A/R item only if it exists in the paying parent bill unit. If the A/R item does not exist, the rerating adjustments remain unallocated at the parent level.

Rolling Up Balances from Members to Owners for Wholesale Sharing Groups

You can roll up the cumulative charges from members of a sharing group to the owner. Typically, BRM locks the owner’s bill unit during billing and rolls up the currency and noncurrency balances from all sharing group members to the owner’s bill unit. This locks the entire sharing group during the billing process. This process takes a long time for wholesale sharing groups and reduces the overall roll-up performance.

However, you can improve performance and avoid locking the owner’s bill unit while rolling up balances by doing the following:

- Enabling the sharing roll-up functionality. You enable this functionality when you create or modify a wholesale sharing group.

- Running the **pin_rollup_sharing_balances** utility. This multithreaded utility launches a separate thread to roll up each member's noncurrency balance to the owner.

> **Note**
>
> - This functionality is only applicable for wholesale hierarchies and subscription events.
> - If a wholesale child is a member of one sharing group, it cannot be an owner of another sharing group.

You should enable the functionality and run the **pin_rollup_sharing_balance** utility before billing if you want the rollup to happen before the actual billing process. It allows you to roll up the noncurrency balances as frequently as required, providing more flexibility. If it is not run ahead of time, the rollup will occur during the billing process.

When you create or modify a wholesale sharing group and enable the sharing roll-up functionality, BRM associates the following with each group member:

- An **/item/sponsor** object, which temporarily stores the portion of the member's currency balance that the owner is responsible for. This creates a temporary table of currency balances for the owner, preventing BRM from needing to lock the owner's bill unit during the billing process.
- A **/tmp_sharing_rollup_imp** object, which temporarily stores the portion of a member's noncurrency balance that the owner is responsible for.

When you run the **pin_rollup_sharing_balances** utility, it looks for each **/tmp_sharing_rollup_imp** object in the sharing group and launches a thread that calls the PCM_OP_SUBSCRIPTION_SHARING_ROLLUP_BALANCE opcode. The opcode rolls up the noncurrency balance to the parent's bill unit and generates an **/event/notification/billing/updateRollupBalances** notification event for synchronizing the balance update with ECE. See "Rolling Up Non-Currency Balances from Members to the Owner of a Wholesale Sharing Group" in *BRM Opcode Guide*.

During billing, each member's currency balance is rolled up from its **/item/sponsor** object to the parent's **/item/sponsor** object.

Optimizing Balance Roll-up for Wholesale Sharing Group

To improve the performance for rolling up balances from group members to the sharing group owner:

1. When you create a wholesale sharing group, enable the sharing roll-up functionality in the wholesale parent's bill unit. See "Enabling the Sharing Roll-Up Functionality in a Wholesale Sharing Group".
2. Create a file that contains the list of owner account POIDs and bill units for which rollup is required. See "Specifying the Wholesale Sharing Groups to Roll Up".
3. Run the **pin_rollup_sharing_balances** utility. See "Running the pin_rollup_sharing_balances Utility".

Enabling the Sharing Roll-Up Functionality in a Wholesale Sharing Group

You can enable the sharing roll-up functionality when you create or modify a wholesale sharing group using one of the following:

- A custom client application calling the sharing group opcodes. See "Managing Sharing Groups" in *BRM Opcode Guide*.
- A custom client application calling the Create a Sharing Group endpoint in the Billing Care REST API. See *REST API Reference for Billing Care*.
- Billing Care. See "Creating a New Charge Sharing Group" in *Billing Care Online Help*.

Specifying the Wholesale Sharing Groups to Roll Up

You specify which wholesale sharing groups the **pin_rollup_sharing_balances** utility rolls up by creating an XML file. To do so:

1. Open the *BRM_home***/apps/pin_billd/PinRollupSharingBalInput.xml** file.
2. Under <**RollupSharingBalConfiguration**>, include each sharing group owner's account POID and bill unit POID you want to have rolled up.

   For example:

```
    <BusinessConfiguration
    xmlns=http://www.portal.com/schemas/BusinessConfig
    xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xsi:schemaLocation=http://www.portal.com/schemas/BusinessConfig
BusinessConfiguration.xsd>

    <RollupSharingBalConfiguration>
    <!-- List of Owner accounts -->
    <Billinfo>
    <BillinfoId>311038</BillinfoId>
    <Account>1210</Account>
    </Billinfo>
    <Billinfo>
    <BillinfoId>311039</BillinfoId>
    <Account>1423</Account>
    </Billinfo>
    </RollupSharingBalConfiguration>
    </BusinessConfiguration>
```

   Table 11-1 describes the XML elements under <**RollupSharingBalConfiguration**>.

   Table 11-1 XML Element Description

Element	Description
Billinfo	The sharing group owner's account and bill units for which you want to roll up the balance.
BillinfoId	The bill unit ID.
Account	The account ID of the sharing group owner.

3. Save and close the file.

Running the pin_rollup_sharing_balances Utility

To optimize performance for rolling up noncurrency balances from members of a sharing group to their owner:

1. Define the wholesale sharing groups to roll up in an XML file. See "Specifying the Wholesale Sharing Groups to Roll Up".
2. Run **pin_rollup_sharing_balances** utility before you run billing:

   ```
   pin_rollup_sharing_balances -file PinRollupSharingBalInput.xml
   ```

   See "pin_rollup_sharing_balances" for details about the utility's syntax and parameters.

The utility rolls up each member's noncurrency balance from its **/tmp_sharing_rollup_imp** object to the sharing group owner. It also clears the temporary balance from each **/tmp_sharing_rollup_imp** object.

Converting Existing Bill Unit Hierarchies to Wholesale Billing

You can convert existing account hierarchies to use wholesale billing. The account hierarchy must meet the qualifications for a wholesale hierarchy, so you cannot convert a hierarchy where the parent account has multiple parent bill units.

The method you use to convert the account hierarchy depends on the method you used to enable wholesale billing.

If you enabled wholesale billing only for specific accounts, use the **PCM_OP_CUST_CONVERT_WHOLESALE_HIERARCHY** opcode to convert the account hierarchy. See "Converting an Account Hierarchy to Wholesale Billing" in *Opcode Guide* for more information about this opcode.

If you enabled wholesale billing for the whole system, as described in "Enabling Wholesale Billing for All Accounts", use the **pin_cust_convert_wholesale_hierarchy** utility. See "pin_cust_convert_wholesale_hierarchy" for more information about this utility.

Create the input file for the utility using the following format:

```
<BusinessConfiguration
    xmlns="http://www.portal.com/schemas/BusinessConfig"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.portal.com/schemas/BusinessConfig
BusinessConfiguration.xsd">

    <ConvertWholesaleConfiguration>
        <!-- List of Billinfos to be converted -->
        <Billinfo>
            <BillinfoId>311038</BillinfoId>
            <Account>1210</Account>
        </Billinfo>
        <Billinfo>
            <BillinfoId>304210</BillinfoId>
            <Account>1423</Account>
        </Billinfo>

        <!-- List of Business profiles to be converted
        <BusinessTypeArray>
            <BusinessProfile>122816</BusinessProfile>
            <BusinessProfileName>WholesaleBilling</BusinessProfileName>
        </BusinessTypeArray>
        <BusinessTypeArray>
            <BusinessProfileName>WholesaleBilling</BusinessProfileName>
        </BusinessTypeArray>
```

```
            <BusinessTypeArray>
                <BusinessProfile>311122</BusinessProfile>
            </BusinessTypeArray>
            -->
        </ConvertWholesaleConfiguration>
    </BusinessConfiguration>
```

Running Wholesale Billing

> **Note**
>
> The following features are not supported for wholesale billing:
>
> - Bill Now
> - On-purchase (on-demand) billing
> - Skipped billing
>
> If you are using bill suppression for billing wholesale accounts, you must run wholesale billing at the end of each accounting cycle.

To run wholesale billing:

1. Roll up A/R actions from nonpaying child bill units to the paying parent bill unit by running the **pin_roll_up_ar_items** utility:

   ```
   pin_roll_up_ar_items
   ```

   See "pin_roll_up_ar_items" in *BRM Managing Accounts Receivable* for information about the utility's syntax and parameters.
2. Roll up temporary journals from nonpaying child bill units to the paying parent bill unit by running the **pin_update_journals** utility:

   ```
   pin_update_journals
   ```

   See "pin_update_journals" for information about the utility's syntax and parameters.
3. Generate regular bills by running the **pin_bill_accts** utility:

   ```
   pin_bill_accts -active
   ```

   See "pin_bill_accts" for information about the utility's syntax and parameters.

   > **Note**
   >
   > After the nonpaying child bill units are billed, you cannot perform A/R activities (such as adjustments, disputes, and settlements) on the billed events until the paying parent bill unit is billed.

4. Generate regular invoices for the wholesale parent account by running the **pin_inv_accts** utility:

   ```
   pin_inv_accts -reg_bills
   ```

   See "pin_inv_accts" in *BRM Designing and Generating Invoices* for more information about the utility's syntax and parameters.

Configuring Billing Delay for Wholesale Hierarchies

Delayed billing is supported for wholesale hierarchies. You must specify the billing delay even if it is not used. In this case, you can set the billing delay interval to **0**. See "Configuring Delayed Billing".

Setting Up Billing-Time Discounts for Wholesale Hierarchies

For wholesale bill unit hierarchies, you set up a billing-time discount as follows:

- Configure billing-time discount only for the paying parent bill unit in the hierarchy.
- Configure BRM to apply the billing-time discount at the end of the billing cycle instead of the accounting cycle.
- Configure non-billing-time discounts (usage discounts) for the nonpaying child bill units in the hierarchy. The usage discount increments the counter. For example, if the billing-time discount for the paying parent bill unit is based on total monthly charges, you can create a discount for a nonpaying child bill unit that increments the counter when charges are applied.

For rolling the discounts up to the paying parent bill unit at the time of billing, you customize the PCM_OP_SUBSCRIPTION_POL_PRE_CYCLE_DISCOUNT policy opcode to return the list of balance element/resource IDs of the counters (in the PIN_FLD_BALANCES output flist field) for which the balances to be rolled up to the paying parent bill unit in the hierarchy.

For more information about billing-time discounts, see "Creating Discount Offers" in *BRM Creating Product Offerings*.

Suppressing Bills for Wholesale Hierarchies

You can use bill suppression to postpone finalizing bills for wholesale accounts. When bill suppression is enabled, the charges applied to the nonpaying child bill units are rolled up to the paying parent bill unit at the end of the accounting cycle. Therefore, you must run the **pin_roll_up_ar_items**, **pin_update_journals**, and **pin_bill_accts** utilities in the same order at the end of each accounting cycle. See "Running Wholesale Billing".

Trial Billing for Wholesale Hierarchies

When you perform trial billing for wholesale bill unit hierarchies, you must run billing for nonpaying child bill units (*wholesale child accounts*) before running billing for the paying parent bill unit (*wholesale parent account*). To do so, run the **pin_trial_bill_accts** utility with the **-pay_type** parameter.

Trial billing generates temporary bills for validation, ensuring accurate charge calculation during the final billing process. If discrepancies arise in the trial bills or charges, customers can adjust the charges and rerun the trial billing to validate them.

In wholesale bill unit hierarchies, run trial billing for all nonpaying subordinate accounts and paying parent accounts. When customers make corrections or adjustments to a few bills, you can rerun trial billing for only the affected accounts, minimizing processing time and ensuring efficient performance across the system.

To improve performance, you can trigger trial billing for:

- All child accounts under a specific parent billing account and the parent billing account
- Specific child accounts alone under a parent billing account
- Specific parent billing account alone skipping the check on whether trial billing is performed for all child accounts under the parent billing account

For more information, see "About Trial Billing for Bill Unit Hierarchies".

Support for A/R Activities

For wholesale bill unit hierarchies, the support for A/R activities varies from level to level:

- **Account level.** Adjustments, disputes, and settlements can be performed only at the parent bill unit level after billing.
- **Bill level.** Adjustments, disputes, and settlements can be performed only after billing at the parent bill unit level.
- **Event level.** Adjustments, disputes, and settlements can be performed at the child bill unit level before and after billing.
- **Item level.** Adjustments and disputes can be performed at the child bill unit level only before billing. After billing, adjustments and disputes are allowed only at the parent bill unit level. However, settlements can be performed before and after billing at the child bill unit level.

Write-offs can be performed only at the parent bill unit level after billing.

Specifying Search Criteria for Retrieving Items, Events, and Bills

To retrieve a list of items, events, or bills, BRM uses the following A/R and payment opcodes:

- PCM_OP_AR_GET_ACTION_ITEMS
- PCM_OP_AR_GET_ACCT_ACTION_ITEMS
- PCM_OP_AR_GET_BAL_SUMMARY
- PCM_OP_AR_GET_ACCT_BAL_SUMMARY
- PCM_OP_AR_GET_BILL_ITEMS
- PCM_OP_AR_GET_BILLS
- PCM_OP_AR_GET_DISPUTE_DETAILS
- PCM_OP_AR_GET_DISPUTES
- PCM_OP_AR_GET_ACCT_BILLS
- PCM_OP_PYMT_ITEM_SEARCH
- PCM_OP_PYMT_MBI_ITEM_SEARCH
- PCM_OP_PYMT_SELECT_ITEMS

Based on the search criteria provided as input, these opcodes search through all bill units in a hierarchy. This can impact wholesale billing performance if you have large wholesale bill unit hierarchies. To improve performance, restrict your search to specific bill units.

Similarly, BRM uses the PCM_OP_AR_GET_ITEM_DETAIL and PCM_OP_AR_GET_ITEMS opcodes to retrieve details about a bill unit's A/R items or bill items. These opcodes cannot retrieve all details about rolled-up items in a wholesale hierarchy. For example, for the A/R items of the paying parent bill unit, these opcodes cannot retrieve the corresponding transfer events for the rolled-up disputes and settlements. Therefore, modify your search to retrieve only the details that are available for wholesale hierarchies.

For information about the search criteria for these opcodes, see "Retrieving A/R Information" in *BRM Opcode Guide*.

Moving Bill Units into or out of Wholesale Hierarchies

You can move nonpaying child bill units into or out of a wholesale bill unit hierarchy. You can also move them between wholesale bill unit hierarchies. Before moving a bill unit, ensure that there are no pending items or payments in that bill unit.

When you move a nonpaying child bill unit to another hierarchy, all items in that bill unit are associated with the corresponding items of the new paying parent bill unit and the new parent is billed for them. If a bill item does not exist in the new parent, it is created and the charges are rolled up to that item.

Specifying How to Calculate Deferred Taxes for Wholesale Billing

You specify how BRM calculates deferred taxes for wholesale billing by setting the **CycleTaxInterval** business parameter to one of the following:

- **Accounting**: The tax for a child account is applied to its bill. BRM calculates taxes for parent accounts and for all child accounts. This is the default.
- **Billing**: The tax is forwarded from the child account to the parent account. BRM calculates taxes for the parent account only, but the single tax item on the parent account includes taxes from both the parent and child accounts.

You set the **CycleTaxInterval** business parameter by using the **pin_bus_params** utility. For information about the utility's syntax and parameters, see "pin_bus_params" in *BRM Developer's Guide*.

To specify how to calculate deferred taxes for wholesale billing:

1. Go to the *BRM_home***/sys/data/config** directory, where *BRM_home* is the directory where you installed BRM components.
2. Create an XML file from the **/config/business_params** object:

   ```
   pin_bus_params -r BusParamsBilling bus_params_billing.xml
   ```

   This command creates the XML file named **bus_params_billing.xml.out** in your working directory. To place this file in a different directory, specify the path as part of the file name.
3. Open the **bus_params_billing.xml.out** file.
4. Set the **<CycleTaxInterval>** element:

   ```
   <CycleTaxInterval>value</CycleTaxInterval>
   ```

   where *value* is either **accounting** (default) or **billing**.

5. Save this file as **bus_params_billing.xml**.
6. Load the XML file into the BRM database:

   ```
   pin_bus_params bus_params_billing.xml
   ```
7. Stop and restart Connection Manager (CM).

12 Load Balancing Billing Runs

Learn how to apply load balancing to billing operations in Oracle Communications Billing and Revenue Management (BRM) by running billing based on the customer's accounting day of month (DOM).

Topics in this document:

> **Note**
>
> Implementing this feature requires using opcodes to create a custom application.

About Managing Billing Cycles

Accounting cycles and billing cycles begin on a bill unit's billing day of month (DOM). By default, the DOM on which an account is created automatically becomes the billing DOM for all the account's bill units. This can result in an uneven distribution of a system's billing operations across each month.

To load-balance billing operations, you can use billing segments to assign accounts to billing DOMs. A *billing segment* is a user-defined category, such as wholesale, retail, and senior citizen. To implement bill cycle management, you create billing segments in your BRM system and then assign bill units to them. The billing segment determines which DOMs are available to the bill units.

A billing segment can be associated with any number of DOMs. For example, billing segment A might be associated with DOMs 1 through 31 and billing segment B might be associated with DOMs 1, 15, and 31. For each DOM with which it is associated, a billing segment contains the following information:

- A status (open, closed, or restricted) that determines whether the DOM can be assigned to the bill units that belong to the segment.
- The maximum number of accounts that can be associated with the DOM–billing segment pair.
- The maximum number of services that can be associated with the DOM–billing segment pair.
- The following data, which you must use third-party data warehousing software to gather:
  - The number of accounts currently associated with the DOM–billing segment pair.
  - The number of services currently associated with the DOM–billing segment pair.
  - The total amount of time that it took to process the bills associated with the DOM–billing segment pair during the *previous* billing run.

Note

To accommodate the frequent updates that such data may require, BRM automatically refreshes cached billing segment data once a day (see "Updating Billing Segments").

BRM uses billing segment information with new bill cycle management functions in the PCM_OP_CUST_POL_PREP_BILLINFO policy opcode to determine the accounting DOMs to which a bill unit can be assigned. For more information, see "Implementing Bill Cycle Management" and *BRM Opcode Guide*.

The status of the accounting DOMs in each billing segment determines whether and how a particular DOM can be assigned to the bill units associated with the billing segment as shown in Table 12-1.

Table 12-1 DOM Status Assignments

DOM Status	Manually Assignable? (1)	Automatically Assignable? (2)
open	Yes	Yes
restricted	Yes	No
closed	No	No

Note

- See "Manually Assigning a Billing DOM".
- See "Automatically Assigning a Billing DOM".

You specify the status of accounting DOMs in the **pin_billing_segment.xml** file. For more information, see "Setting Up Billing Segments".

Note

- Accounting DOMs not explicitly associated with a billing segment are considered *closed* for the segment. For example, if billing segment C is associated only with DOMs 1 and 2, DOMs 3 through 31 are closed for that segment and cannot be assigned to bill units that belong to the segment.
- If a billing segment is not associated with any accounting DOMs, all DOMs are *open* for the segment. In such cases, the default process, not the bill cycle management process, is used to assign an accounting DOM to the bill units that belong to the segment.

Implementing Bill Cycle Management

To implement bill cycle management:

1. Set up billing segments in your system. See "Setting Up Billing Segments".
2. Perform the following tasks programmatically or through a custom user interface:

- Associate bill units with billing segments at account creation and account maintenance time. See "Associating Bill Units with Billing Segments ".
- (Optional) Select a billing DOM for bill units. See "Assigning Accounting Days of Month to Bill Units in Billing Segments".

3. (Optional) Customize the PCM_OP_CUST_POL_PREP_BILLINFO opcode to select the DOM most likely to have the lightest billing load of all available DOMs. See the chapter about billing in *BRM Opcode Guide*.

Setting Up Billing Segments

To set up billing segments in your system, edit the billing segment configuration file *BRM_home***/sys/data/config/pin_billing_segment.xml**, and load its contents into the **/config/billing_segment** object in the BRM database.

> **Note**
>
> The utility that loads billing segments into the database overwrites existing billing segments. When updating billing segments, you cannot load new segments only. You must load the complete set of billing segments each time you run the utility.

1. Open the **pin_billing_segment.xml** file in an XML editor or a text editor.
2. Enter the appropriate information into the file. See "Editing the Billing Segment Configuration File".
3. Save the file.
4. Use the following command to run the "load_pin_billing_segment" utility from the directory in which the **pin_billing_segment.xml** file is located:

   ```
   load_pin_billing_segment pin_billing_segment.xml
   ```

   If you do not run the utility from the directory in which **pin_billing_segment.xml** is located, include the complete path to the file. For example:

   ```
   load_pin_billing_segment BRM_home/sys/data/config/pin_billing_segment.xml
   ```

   For more information, see "load_pin_billing_segment".
5. Activate the feature that automatically refreshes billing segment data in the Connection Manager (CM) cache. See "Automatically Refreshing Billing Segment Data".
6. Stop and restart the CM.

   > **Note**
   >
   > - If the data does not need to be added to the CM cache until the next time the cache is automatically refreshed, you do not have to do this.
   > - If an error occurs at CM startup, the space allocated to billing segment data in the CM cache may not be sufficient to accommodate the size of your billing segment data. See "Increasing the Size of the CM Cache for Billing Segment Data".

7. To verify that the billing segment information was loaded, display the **/config/billing_segment** object by using one of the following features:
   - Object Browser
   - **robj** command with the **testnap** utility

   For general instructions on using **testnap**, see "Using the testnap Utility to Test BRM" in *BRM Developer's Guide*. For information about reading an object and writing its contents to a file, see "Reading an Object and Writing Its Contents to a File" in *BRM Developer's Guide*.

Editing the Billing Segment Configuration File

You configure all the billing segments in your BRM system in the *BRM_home***/sys/data/config/pin_billing_segment.xml** file.

To edit this configuration file, open it in an XML editor or a text editor and then perform these tasks:

- To add billing segments to the file, see "Defining Billing Segments".
- To associate the billing segments with billing DOMs. See "Associating Billing Segments With Accounting Days of Month".

Defining Billing Segments

In the billing segment configuration file, billing segments are defined as **BillingSegment** child elements of the **BillingSegments** parent element.

A **BillingSegment** child element consists of a billing segment ID and a description (*string*):

```
<BillingSegmentConfiguration>
   <BillingSegments>
      <BillingSegment ID="int ">string</BillingSegment>
   </BillingSegments>
</BillingSegmentConfiguration>
```

To create a billing segment, add a **BillingSegment** child element to the **BillingSegments** parent element. In the child element, specify values for the items listed in Table 12-2.

Table 12-2 XML Elements in Billing Segment

XML Element or Attribute	Description	Possible Values
ID	A number that identifies the billing segment in the BRM database. When a bill unit is linked to a billing segment, this number is put in the PIN_FLD_BILLING_SEGMENT field of the **/billinfo** object. An array of all the billing segment IDs is stored in the **/config/billing_segment** object.	To use bill cycle management to assign billing DOMs to bill units in the segment, specify any integer greater than or equal to **101**. **Note:** ID 0 triggers BRM to use the non-bill-cycle-management assignment process.

Table 12-2 (Cont.) XML Elements in Billing Segment

XML Element or Attribute	Description	Possible Values
string	A character string that describes the type of accounts in the billing segment (for example, **wholesale** or **retail**).	Minimum length is 1 character. Maximum length is 1023 characters. **Note:** This string is mapped to the PIN_FLD_DESCR field in the **/config/billing_segment** object, which can be used to populate a list of billing segments in a user interface (UI). When creating the string, take any UI length restrictions into account.

Associating Billing Segments With Accounting Days of Month

In the billing segment configuration file, the **DomAssignments** parent element contains **DomAssignment** child elements, each of which associates a billing segment with a DOM:

```
<BillingSegmentConfiguration>
   <DomAssignments>
      <DomAssignment billingSegmentRef="int" status=" status " dom=" ---gDay "
       maxAccounts="int" maxServices="int">
         <NumAccounts>int</NumAccounts>
         <NumServices>int</NumServices>
         <TotalProcessTime>duration</TotalProcessTime>
      </DomAssignment>
   </DomAssignments>
</BillingSegmentConfiguration>
```

To create a billing segment–DOM association, add a **DomAssignment** child element to the **DomAssignments** parent element. In the child element, specify values for the items listed in Table 12-3.

Note

- You must add one **DomAssignment** child element for every billing segment–DOM association. For example, to associate billing segment 101 with DOMs 1 through 31, you must add 31 **DomAssignment** child elements. The **billingSegmentRef** value of each child element will be the *same* (101), but each child element will have a *different* **dom** value.
- If a billing segment is not explicitly associated with any accounting DOMs, all DOMs are *open* for the segment.
- An array of all the billing segment and DOM pairs in a BRM system (PIN_FLD_MAP) is stored in the **/config/billing_segment** object.

Table 12-3 DOM Assignments

XML Element or Attribute	Description	Possible Values
billingSegmentRef	The billing segment ID value in the **ID** attribute of a **BillingSegment** child element in the **BillingSegments** parent element.	See the **billingSegmentId** entry in the table in "Defining Billing Segments".
status	The status of a DOM for the billing segment.	One of the following: • **open** • **restricted** • **closed** **Important:** These values are case sensitive.
dom	A billing DOM.	Any two-digit value from **01** through **31**.
(Optional) **maxAccounts**	Maximum number of accounts that can be associated with the DOM–billing segment pair.	Any non-negative integer.
(Optional) **maxServices**	Maximum number of services that can be associated with the DOM–billing segment pair.	Any non-negative integer.
(Optional) **NumAccounts**	Number of accounts currently associated with the DOM–billing segment pair.	Any non-negative integer. **Note:** This information is generated by third-party data warehousing software.
(Optional) **NumServices**	Number of services currently associated with the DOM–billing segment pair.	Any non-negative integer. **Note:** This information is generated by third-party data warehousing software.
(Optional) **TotalProcessTime**	Total amount of time (in seconds) that it took to process the bills associated with the DOM–billing segment pair during the previous billing run.	Any duration type value. For example, **P1Y3M2DT1H20M30S** (1 year, 3 months, 2 days, 1 hour, 20 minutes, and 30 seconds). The "load_pin_billing_segment" utility converts this value into seconds. **Note:** This information is generated by third-party data warehousing software.

Sample Billing Segment Configuration File

The following is a sample **pin_billing_segment.xml** file:

```
<?xml version="1.0" encoding="UTF-8"?>
<BusinessConfiguration xmlns="http://www.portal.com/schemas/BusinessConfig"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.portal.com/schemas/BusinessConfig
business_configuration.xsd">
   <BillingSegmentConfiguration>
      <BillingSegments>
         <BillingSegment ID="101">First Billing Segment</BillingSegment>
         <BillingSegment ID="102">Second Billing Segment</BillingSegment>
         <BillingSegment ID="103">Third Billing Segment</BillingSegment>
      </BillingSegments>
      <DomAssignments>
         <DomAssignment billingSegmentRef="101" status=" restricted " dom="
---31  "
          maxAccounts="7400" maxServices="70033">
            <NumAccounts>4</NumAccounts>
            <NumServices>5</NumServices>
            <TotalProcessTime>PT20S</TotalProcessTime>
         </DomAssignment>
         <DomAssignment billingSegmentRef="102" status="open" dom="---07"
maxAccounts="7400"
          maxServices="733">
            <NumAccounts>76</NumAccounts>
            <NumServices>5</NumServices>
            <TotalProcessTime>P1D</TotalProcessTime>
         </DomAssignment>
      </DomAssignments>
   </BillingSegmentConfiguration>
</BusinessConfiguration>
```

Validating Your Billing Segment Configuration File Edits

After editing the contents of the XML file, you use the "load_pin_billing_segment" utility to load the contents of the file into the **/config/billing_segment** object in the database. See "Setting Up Billing Segments".

Before loading the contents of the file, the utility validates the contents against the file's schema definition. If the contents do not conform to the schema definition, the load operation fails. By default, the schema definition is *BRM_home*/**xsd/pin_billing_segment.xsd**.

The XML file is not directly linked to its schema definition file. Instead, it is linked to the *BRM_home*/**sys/data/config/business_configuration.xsd** reference file.

Updating Billing Segments

To update billing segment data, re-edit the billing segment configuration file, and then run the "load_pin_billing_segment" utility to load the updated contents of the file into the **/config/billing_segment** object in the BRM database. See "Setting Up Billing Segments".

Note

The utility that loads billing segments into the database overwrites existing billing segments. When updating billing segments, you cannot load new segments only. You must load the complete set of billing segments each time you run the utility.

Adding Updated Billing Segment Data to the CM Cache

To add newly loaded billing segment data to the CM cache, do one of the following:

- Manually refresh the cache by stopping and restarting the CM after running the load utility.
- Automatically refresh the cache. See "Automatically Refreshing Billing Segment Data".

Note

If a re-edited billing segment configuration file is significantly larger than the previous version of the file, you might have to increase the space allocated to the data in your CM cache to prevent an error from occurring at CM startup. See "Increasing the Size of the CM Cache for Billing Segment Data".

Automatically Refreshing Billing Segment Data

To refresh billing segment data automatically:

1. Open the CM configuration file (*BRM_home*/**sys/cm/pin.conf**).
2. Uncomment the **fm_cust billing_segment_config_refresh_delay** entry by deleting the number sign (#) at the beginning of the entry.
3. (Optional) Change the refresh frequency.

   By default, this entry is set to **86400** (24 hours, in seconds). This refreshes the cache once a day. To change the frequency, replace this value with the appropriate number of seconds.

   For example, to refresh the data only once a week, change the value to **604800** (60 seconds x 60 minutes x 24 hours x 7 days).
4. Stop and restart the CM.

Note

To turn off the refresh feature, see "Preventing Automatic Updates of Billing Segment Data".

Increasing the Size of the CM Cache for Billing Segment Data

If your billing segment configuration file contains a lot of data, you might need to increase the space allocated to that data in the CM cache to prevent an error from occurring at CM startup:

1. Open the CM configuration file (*BRM_home*/**sys/cm/pin.conf**).
2. Increase cache_size in the following entry:

   ```
   - cm_cache fm_cust_billing_segment number_of_entries, cache_size, hash_size
   ```

   The default is 51200 bytes.

   For example, change this:

   ```
   - cm_cache fm_cust_billing_segment 1, 51200, 1
   ```

   To this:

   ```
   - cm_cache fm_cust_billing_segment 1, 102400, 1
   ```
3. Increase cache_size in the following entry:

   ```
   - cm_cache fm_cust_dom_map number_of_entries, cache_size, hash_size
   ```

   The default is 102400 bytes.

   For example, change this:

```
- cm_cache fm_cust_dom_map 1, 102400, 1
```

To this:

```
- cm_cache fm_cust_dom_map 1, 204800, 1
```

4. Stop and restart the CM.

Preventing Automatic Updates of Billing Segment Data

To prevent your system from automatically refreshing billing segment data in the CM cache:

1. Open the CM configuration file (*BRM_home***/sys/cm/pin.conf)**.
2. Comment out the **fm_cust billing_segment_config_refresh_delay** entry by inserting a number sign (#) at the beginning of the entry.
3. Stop and restart the CM.

Note

By default, this feature is turned *off*. To turn it on, see "Automatically Refreshing Billing Segment Data".

Associating Bill Units with Billing Segments

All the billing segments in a BRM system are defined in the **/config/billing_segment** object. Within that object, the PIN_FLD_BILLING_SEGMENTS array contains the ID and description (PIN_FLD_DESCR) of each billing segment.

To link a bill unit to a billing segment, put the ID of the billing segment into the PIN_FLD_BILLING_SEGMENT field of the **/billinfo** object.

Note

Every bill unit in an account and every nonpaying bill unit associated with the account must belong to the same billing segment. If you try to associate the bill units with different billing segments, BRM returns an error.

To change a bill unit's billing segment, you need to run the PCM_OP_CUST_UPDATE_CUSTOMER opcode. See *BRM Opcode Guide*.

Changing a Bill Unit's Billing Segment

You can change a bill unit's billing segment in one of two ways:

- **Change the billing segment but not the billing DOM**

  To change a bill unit's billing segment *but not* its billing DOM, call the PCM_OP_CUST_UPDATE_CUSTOMER opcode with the new billing segment ID to put in the PIN_FLD_BILLING_SEGMENT field of the **/billinfo** object.

> **Note**
>
> The status of the DOM in the new billing segment must be **open**. If it is not, an error is returned.

- **Change the billing segment and the billing DOM**

  To change both a bill unit's billing segment *and* its billing DOM, call PCM_OP_CUST_UPDATE_CUSTOMER with the following **/billinfo** field values:

  – A new billing segment ID to put in the PIN_FLD_BILLING_SEGMENT field.

  – A new billing DOM to put in the PIN_FLD_ACTG_FUTURE_DOM field. The status of the DOM must be **open** in the specified billing segment.

  To change only a bill unit's billing DOM, see "Changing a Bill Unit's Billing DOM".

Assigning Accounting Days of Month to Bill Units in Billing Segments

When a bill unit is associated with a billing segment, one of the available billing DOMs in the segment must be assigned to the bill unit. A DOM's availability depends in part on the status of the DOM.

DOMs can be assigned in either of the following ways:

- Manually Assigning a Billing DOM
- Automatically Assigning a Billing DOM

To change a bill unit's billing DOM, see "Changing a Bill Unit's Billing DOM".

Manually Assigning a Billing DOM

To assign billing DOMs to new or existing bill units manually, create an application that enables customer service representatives to select one of the open or restricted DOMs in the billing segment with which a bill unit is associated.

> **Note**
>
> Essentially, the CSR is selecting the bill unit's *billing DOM*.

The PIN_FLD_MAP array in the **/config/billing_segment** object contains all the billing segment–DOM pairs configured in your system.

For each pair, the status of the DOM is stored in the PIN_FLD_STATUS field.

To validate the CSR's selection, the application should call PCM_OP_CUST_UPDATE_CUSTOMER. That opcode calls the PCM_OP_CUST_SET_BILLINFO opcode, which calls the PCM_OP_CUST_POL_PREP_BILLINFO and the PCM_OP_CUST_POL_VALID_BILLINFO policy opcodes.

To link a bill unit to a billing DOM, put the DOM value (1-31) into the PIN_FLD_ACTG_CYCLE_DOM field of the **/billinfo** object.

Automatically Assigning a Billing DOM

If a billing DOM is not *manually* selected for a bill unit after the unit is assigned to a billing segment, the PCM_OP_CUST_POL_PREP_BILLINFO policy opcode *automatically* assigns a DOM to the bill unit.

The policy opcode assigns DOMs whose status is **open**; it cannot assign DOMs whose status is **restricted**.

Changing a Bill Unit's Billing DOM

To change a bill unit's billing DOM, call PCM_OP_CUST_UPDATE_CUSTOMER with the new DOM to put the PIN_FLD_ACTG_CYCLE_DOM field of the **/billinfo** object.

Note

The status of the new DOM must be **open** in the bill unit's billing segment. If it is not, an error is returned.

To change a bill unit's billing segment, see "Changing a Bill Unit's Billing Segment".

13 About Proration

Learn how proration works in Oracle Communications Billing and Revenue Management (BRM).

Topics in this document:

Calculating Prorated Cycle Fees

BRM calculates the prorated fee for a specified period by multiplying the cycle fee defined in the applicable charge offer by a scale based on the amount of time the charge offer is valid during the cycle. The scale for a whole cycle (unit interval, or usually an accounting cycle) is equal to 1.

BRM calculates the scale for a partial cycle by dividing the number of days in the partial cycle by the number of days in the unit interval within which it falls. The number of days in a unit interval will vary depending on its start and end dates.

If the period for which the prorated fee is being calculated is greater than one unit interval, the scale will be greater than 1. Likewise, if the period is less than one unit interval, the scale will be less than 1. For example, if the cycle fee is $100, and the period being prorated is half a unit interval, the scale for that interval will be 0.5. So the prorated cycle fee is 0.5 * $100 = $50.

BRM does the following to calculate the prorated fee for each charge offer that is valid during the cycle:

1. Calculates the unit intervals. See "Calculating the Unit Interval".
2. Calculates the scales for the part of the period to be prorated that falls into each unit interval.
3. Sums up the scales for all the unit intervals to get the scale for the entire period to be prorated.
4. Calculates the prorated amount by multiplying the scale for the entire period by the cycle fee amount.

Calculating the Unit Interval

BRM takes into account what the **use_number_of_days_in_month** entry in the Connection Manager (CM) configuration (**pin.conf)** file is set to while calculating the unit interval. Also, the use of special days (29th, 30th, and 31st of a month) for billing is taken into account while calculating the unit interval.

Note

If the **TimestampRounding** business parameter is disabled, the unit interval is calculated in seconds because the time stamp will not be rounded to midnight and the proration will begin from the time that is indicated by the time stamp. If **TimestampRounding** is enabled, the unit interval will be calculated in days because the time stamp will be rounded to midnight.

Calculating Unit Interval When use_number_of_days_in_month Is Not Set or 0

If **use_number_of_days_in_month** is not set or is set to **0**, the unit interval is calculated based on the billing time. Assuming the billing time to be March 22, starting from the billing time, BRM calculates the last unit interval by moving to the left (on the time axis) one month at a time until the beginning of the period to be prorated is covered. For example, while calculating the prorated fee for Mar 1–Mar 15, it takes the unit interval as 28 (the number of days between February 22 and March 22) because both March 1 and March 15 fall between February 22 and March 22. See "Example 1: If Use_number_of_days_in_month Is Not Set or Set to 0".

Calculating Unit Interval When use_number_of_days_in_month Is Set to 1

If **use_number_of_days_in_month** is set to **1** *and* the period to be prorated falls within the same calendar month, the unit interval is the number of days in the whole calendar month in which the period to be prorated falls. For example, while calculating the prorated fee for Mar 1–Mar 15 (both dates fall in March), BRM calculates the unit interval as 31, because March has 31 days. See "Example 2: If Use_number_of_days_in_month Is Set to 1".

Note

If the period for which BRM is prorating the fees is less than a month but spans across multiple months (for example, Aug 19–Sep 15) or if the cycle fee is for multimonth, **use_number_of_days_in_month** is ignored.

Calculating Unit Interval When Billing Day of Month Is 29, 30, or 31

When the billing day of month (DOM) is set to 29, 30, or 31, the unit interval calculation is based on the option you set for this feature (the forward or back option set in the **/config/business_params** object.) If the option is set to forward and the month does not have the billing DOM, billing will run on the first day of the next month. If the option is set to back and the month does not have the billing DOM, billing will run on the last day of the previous month. This causes the start date of the unit interval to shift based on what option is set.

Examples of Proration

Following are some examples of proration. In each of these examples, the unit interval is calculated differently. .

Example 1: If Use_number_of_days_in_month Is Not Set or Set to 0

In this example, illustrated in Figure 13-1, the prorated cycle fee is calculated for the interval Feb 15–Apr 13 with the billing time as April 22. To calculate the prorated fee for this period, BRM does the following:

Calculates the Unit Intervals for the Period

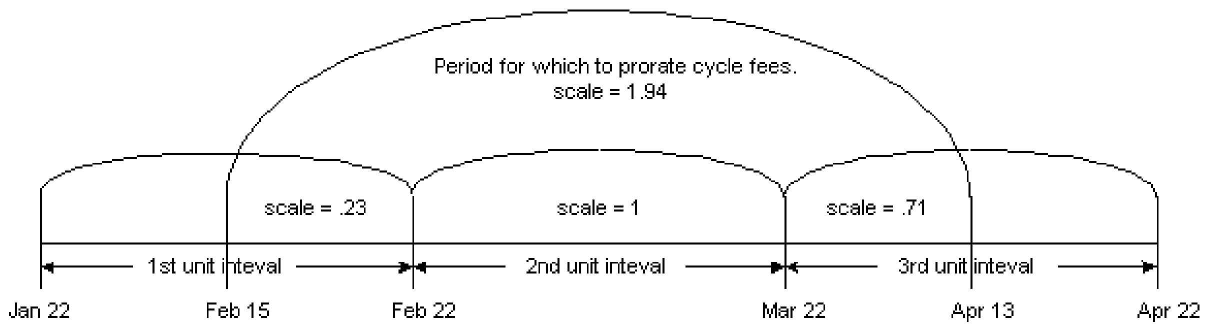
Starting from the billing time (April 22), BRM calculates the last unit interval by moving to the left one month at a time (on the time axis). Therefore, the last unit interval will be Mar 22–Apr 22. Similarly, it will continue to calculate unit intervals until the start time of the first unit interval (January 22) will be equal to or before the start time of the given period to be prorated (February 15). As a result, it gets the following unit intervals:

Mar 22–Apr 22 (last unit interval)

Feb 22–Mar 22 (second unit interval)

Jan 22–Feb 22 (first unit interval)

Figure 13-1 Example 1



Calculates the Scale

To calculate the scale for the period Feb 15–Apr 13, BRM does the following:

1. Calculates the scale for the period Feb 15–Feb 22 with the unit interval as Jan 22–Feb 22.

   Scale = 7/31 = 0.23

   where 7 is the number of days between February 15 and February 22 and 31 is the number of days between January 22 and February 22.
2. Calculates the scale for the period Feb 22–Mar 22 with the unit interval as Feb 22–Mar 22. This scale is 28/28 = 1 because the period to be prorated is the unit interval.
3. Calculates the scale for the period Mar 22–Apr 13 with the unit interval as Mar 22–Apr 22.

   Scale = 22/31 = 0.71

   where 22 is the number of days between March 22 and April 13 and 31 is the number of days between March 22 and April 22.
4. Calculates the scale for the whole period Feb 15–Apr 13 as the sum of the scales from steps 1, 2, and 3: 0.23 + 1.0 + 0.71 = 1.94.

Calculates the Prorated Amount

If the cycle fee amount is $100, BRM calculates the prorated amount for the period Feb 15–Apr 13 as follows:

1.94 * $100 = $194

Example 2: If Use_number_of_days_in_month Is Set to 1

In this example, illustrated in Figure 13-2, the prorated cycle fee is calculated for the interval Feb 15–Apr 13 with the billing time as April 22. Because **use_number_of_days_in_month** is set to **1**, BRM calculates the unit interval based on calendar month for the part of the proration time that falls in the same calendar month. To calculate the prorated fee for this period, it does the following:

Calculates the Unit Intervals for the Period

Starting from the billing time (April 22), BRM calculates the last unit interval by moving to the left one month at a time (on the time axis). Therefore, the unit intervals will be as follows:

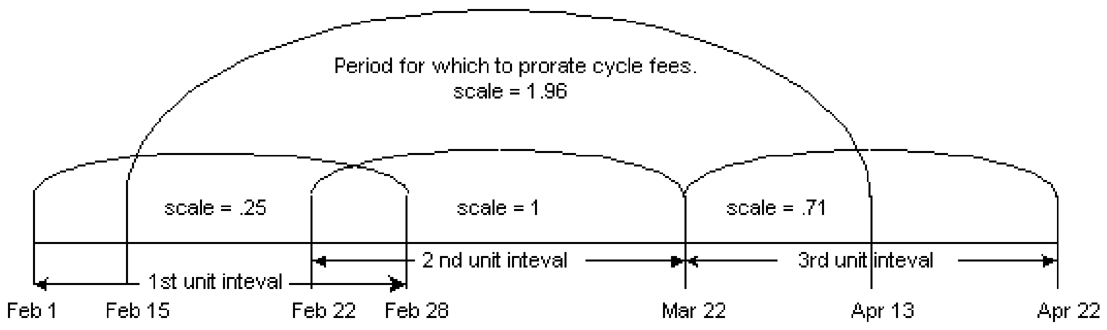
Mar 22–Apr 22 (last unit interval)

Feb 22–Mar 22 (second unit interval)

Feb 1–Feb 28 (first unit interval)

> **Note**
>
> This unit interval is the number of days in February because the entire period Feb 15–Feb 22 falls in February.

Figure 13-2 Example 2



Calculates the Scale

To calculate the scale for the period Feb 15–Apr 13, BRM does the following:

1. Calculates the scale for the period Feb 15–Feb 22 with the unit interval as Feb 1–Feb 28.

Note

If the part of the period to be prorated falls in the same calendar month, the unit interval is the number of days in the month. In this example, because the entire period Feb 15–Feb 22 falls in February, the unit interval is 28, the number of days in February.

Scale = 7/28 = 0.25

where 7 is the number of days between February 15 and February 22 and 28 is the number of days in February.

2. Calculates the scale for the period Feb 22–Mar 22 with the unit interval as Feb 22–Mar 22. This scale is 28/28 = 1 because the period to be prorated is the unit interval.
3. Calculates the scale for the period Mar 22–Apr 13 with the unit interval as Mar 22–Apr 22.

   Scale = 22/31 = 0.71

   where 22 is the number of days between March 22 and April 13 and 31 is the number of days between March 22 and April 22.
4. Calculates the scale for the whole period Feb 15–Apr 13 as the sum of the scales from steps 1, 2, and 3: 0.25 + 1.0 + 0.71 = 1.96.

Calculates the Prorated Amount

If the cycle fee amount is $100, BRM calculates the prorated amount for the period Feb 15–Apr 13 as follows:

1.96 * $100 = $196

Examples Using the 29th, 30th, and 31st for Billing Day of Month

The examples below show the use of special days (29th, 30th, 31st) as the billing DOM because of which the unit interval calculation will be based on the option you set for this feature (the forward or back option set in **/config/business_params** object). If the option is set to forward and the month does not have the billing DOM, billing is run on the first day of the next month. If the option is set to back and the month does not have the billing DOM, billing is run on the last day of the previous month. This causes the unit interval start date to shift based on the option set.

Example 3a: Use Forward Option with use_number_of_days_in_month Set to 0

This example, illustrated in Figure 13-3, assumes the billing option to be set to forward and **use_number_of_days_in_month** to be set to **0**. To calculate the prorated cycle fee for the period Feb 15–Apr 13 with the billing day of month as April 30, BRM does the following:

Calculates the Unit Intervals for the Period

Starting from the billing time (April 30), BRM calculates the last unit interval by moving to the left (on the time axis) one month at a time. Therefore, the last unit interval will be Mar 30–Apr 30. Similarly, it continues to calculate unit intervals until the start time of the first unit interval (January 30) will be equal to or before the start time of the given period to be prorated (February 15). As a result, BRM gets the following unit intervals:

Mar 30–Apr 30 (last unit interval)

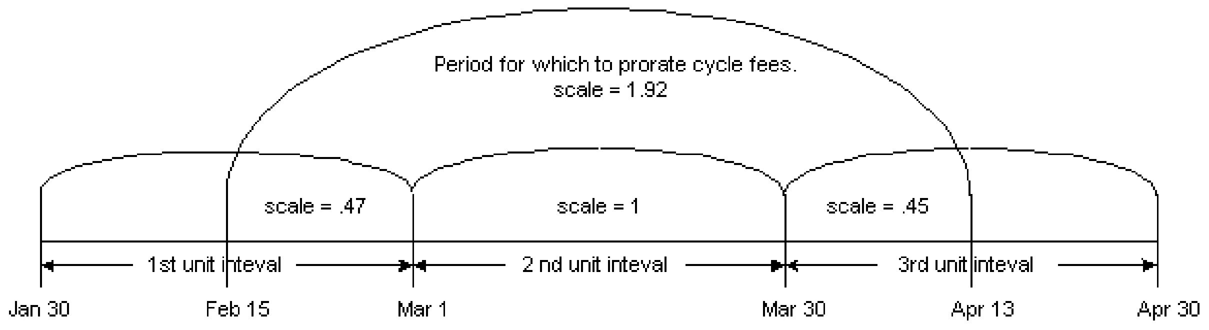
Mar 1–Mar 30 (second unit interval)

> **Note**
>
> Because February does not have 30 days and the billing option is set to forward, BRM starts the unit interval on March 1 (the first day of the next month).1.

Jan 30–Mar 1 (first unit interval)

> **Note**
>
> Because January does have 30 days, it calculates this unit interval as beginning on January 30.

Figure 13-3 Example 3a



Calculates the Scale

To calculate the scale for the period Feb 15–Apr 13, BRM does the following:

1. Calculates the scale for the period Feb 15–Mar 1 with the unit interval as Jan 30–Mar 1 as follows:

   Scale = 14/30 = 0.47

   where 14 is the number of days from February 15 and March 1 and 30 is the number of days from January 30 and March 1.
2. Calculates the scale for the period Mar 1–Mar 30 with the unit interval as Mar 1–Mar 30.

   The scale for this period is 29/29 = 1 because number of days to be prorated is equal to number of days in the unit interval.
3. Calculates the scale for the period Mar 30–Apr 13 with the unit interval as Mar 30–Apr 30 as follows:

   Scale = 14/31 = .45

   where 14 is the number of days from March 30 and April 13 and 31 is the number of days from March 30 and April 30.

4. Adds the scales from the steps above to get the scale for the whole period:

   Scale for the period Feb 15–Apr 13 = .47 + 1.0 + .45 = 1.92

Calculates the Prorated Cycle Fee Amount

If the cycle fee is $100, the prorated cycle fee for the period Feb 15–Apr 13 will be:

1.92 * $100 = $192

Example 3b: Use Forward Option with use_number_of_days_in_month Set to 1

This example, illustrated in Figure 13-4, assumes the billing option to be set to forward and **use_number_of_days_in_month** to be set to **1**. To calculate the prorated cycle fee for the period Feb 15–Apr 13 with the billing DOM as April 30, BRM does the following:

Calculates the Unit Intervals for the Period

Starting from the billing time (April 30), BRM calculates the last unit interval by moving to the left one month at a time (on the time axis). Therefore, the last unit interval will be Mar 30–Apr 30. Similarly, it continues to calculate unit intervals until the start time of the first unit interval (January 30) is equal to or before the start time of the given period to be prorated (February 15). As a result, BRM gets the following unit intervals:

Mar 30–Apr 30 (last unit interval)

Mar 1–Mar 31 (second unit interval)

Note

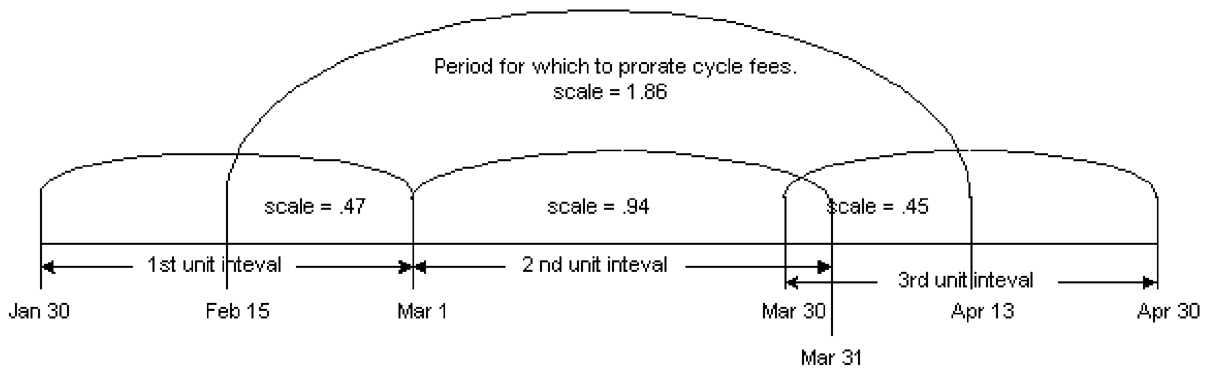
Because February does not have 30 days and the billing option is set to forward, BRM starts the unit interval on March 1 (the first day of the next month).. Also, **use_number_of_days_in_month** is set to **1**, it calculates this unit interval to be 31, the number of days in March.

Jan 30–Mar 1 (first unit interval)

Note

Because January does have 30 days, this unit interval will begin on Jan 30.

Figure 13-4 Example 3b



Calculates the Scale

To calculate the scale for the period Feb 15–Apr 13, BRM does the following:

1. Calculates the scale for the period Feb 15–Mar 1 with the unit interval as Jan 30–Mar 1 as follows:

   Scale = 14/30 = 0.47

   where 14 is the number of days from February 15 and March 1 and 30 is the number of days between January 30 and March 1.

2. Calculates the scale for the period Mar 1–Mar 30 with the unit interval as Mar 1–Mar 31.

   > **Note**
   >
   > Because March 1 and March 30 fall in the same calendar month, the unit interval here will be the number of days in March, because **use_number_of_days_in_month** is set to **1**.

   Scale = 29/31 = .94

   where 29 is the number of days between March 1 and March 30 and 31 is the number of days in March.

3. Calculates the scale for the period Mar 30–Apr 13 with the unit interval as Mar 30–Apr 30 as follows:

   Scale = 14/31 = .45

   where 14 is the number of days from March 30 and April 13 and 31 is the number of days between March 30 and April 30.

4. Adds the scales in the steps above to get the scale for the whole period:

   Scale for the period Feb 15–Apr 13 = .47 + .94 + .45 = 1.86

Calculates the Prorated Cycle Fee Amount

If the cycle fee is $100, the prorated cycle fee for the period Feb 15–Apr 13 will be:

1.86 * $100 = $186

Example 3c: Use Back Option with use_number_of_days_in_month Set to 1

This example, illustrated in Figure 13-5, assumes the billing option to be set to back and **use_number_of_days_in_month** to be set to **1**. To calculate the prorated cycle fee for the period Feb 15–Apr 13 with the billing DOM as April 30, BRM does the following:

Calculates the Unit Intervals for the Period

Starting from the billing time (April 30), BRM calculates the last unit interval by moving to the left one month at a time (on the time axis). Therefore, the last unit interval will be Mar 30–Apr 30. Similarly, it continues to calculate unit intervals until the start time of the first unit interval is equal to or before the start time of the given period (February 15) to be prorated. As a result, it gets the following unit intervals:

Mar 30–Apr 30 (last unit interval)

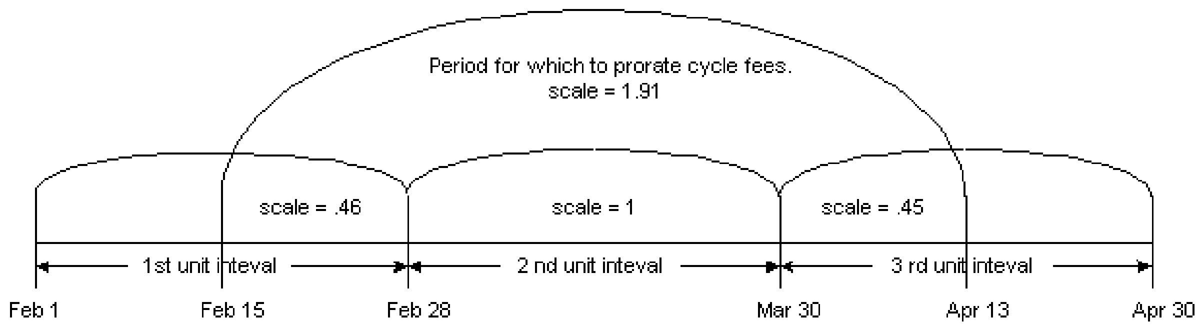
Feb 28–Mar 30 (second unit interval)

> **Note**
>
> Because February does not have 30 days and the billing option is set to back, BRM calculates this unit interval as beginning on the last day of the previous month, which is February 28.

Feb 1–Feb 28 (first unit interval)

> **Note**
>
> In this example, the first unit interval would have started on January 30, because January does have 30 days, but BRM ignores this and takes the number of days in February as the unit interval because **use_number_of_days_in_month** is set to **1**.

Figure 13-5 Example 3c



Calculates the Scale

To calculate the scale for the period Feb 15–Apr 13, BRM does the following:

1. Calculates the scale for the period Feb 15–Feb 28 with the unit interval as Feb 1–Feb 28 as follows:

   Scale = 13/28 = 0.46

   where 13 is the number of days from February 15 and February 28 and 28 is the number of days in February.

   > **Note**
   >
   > Because February 15 and February 28 fall in the same calendar month, the unit interval here will be the number of days in February because **use_number_of_days_in_month** is set to **1**.

2. Calculates the scale for the period Feb 28–Mar 30 with the unit interval as Feb 28–Mar 30.

   The scale for this period is 30/30 = 1 because the number of days to be prorated is equal to number of days in the unit interval.

3. Calculates the scale for the period Mar 30–Apr 13 with the unit interval as Mar 30–Apr 30 as follows:

   Scale = 14/31 = .45

   where 14 is the number of days from March 30 and April 13 and 31 is the number of days between March 30 and April 30.

4. Adds the above three scales to get the scale for the whole period:

   Scale for the period Feb 15–Apr 13 = .46 + 1.0 + .45 = 1.91

Calculates the Prorated Cycle Fee Amount

If the cycle fee is $100, the prorated cycle fee for the period Feb 15–Apr 13 will be:

1.91 * $100 = $191

Example 3d: Use Back Option with use_number_of_days_in_month Set to 0

This example, illustrated in Figure 13-6, assumes the billing option to be set to back and **use_number_of_days_in_month** to be set to **0**. To calculate the prorated fee for the period Feb 15–Apr 13 with the billing day of month as April 30, BRM does the following:

Calculates the Unit Intervals for the Period

Starting from the billing time (April 30), it calculates the last unit interval by moving to the left one month at a time (on the time axis). Therefore, the last unit interval will be Mar 30–Apr 30. Similarly, it continues to calculate unit intervals until the start time of the first unit interval is equal to or before the start time of the given period to be prorated (February 15). As a result, BRM gets the following unit intervals:

Mar 30–Apr 30 (last unit interval)

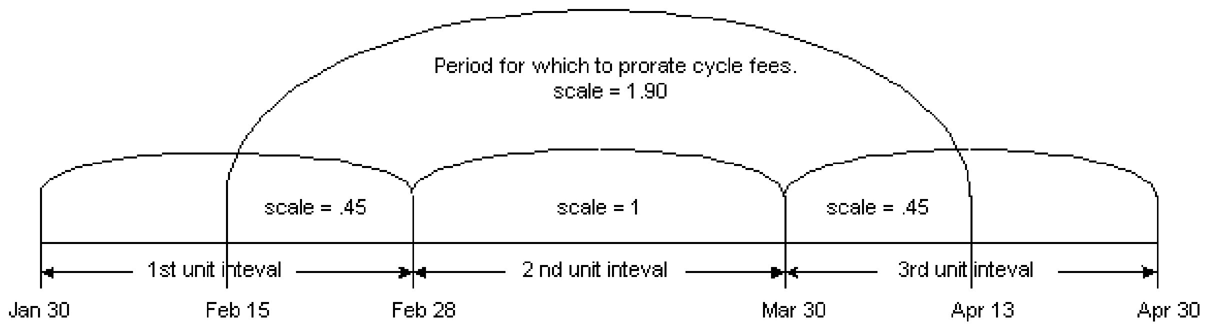
Feb 28–Mar 30 (second unit interval)

> **Note**
>
> Because February does not have 30 days and the billing option is set to back, BRM calculates this unit interval as beginning on the last day of the previous month, which is February 28.

Jan 30–Feb 28 (first unit interval)

> **Note**
>
> Because January does have 30 days, BRM calculates this unit interval as beginning on January 30.

Figure 13-6   Example 3d



Calculates the Scale

To calculate the scale for the period Feb 15–Apr 13, BRM does the following:

1. Calculates the scale for the period Feb 15–Feb 28 with the unit interval Jan 30–Feb 28:

   Scale = 13/29 = 0.45

   where 13 is the number of days from February 15 and February 28 and 29 is the number of days from January 30 and February 28.
2. Calculates the scale for the period Feb 28–Mar 30 with the unit interval as Feb 28–Mar 30.

   The scale for this period is 30/30 = 1 because the number of days to be prorated is equal to number of days in the unit interval.
3. Calculates the scale for the period Mar 30–Apr 13 with the unit interval as Mar 30–Apr 30.

   Scale = 14/31 = .45

   where 14 is the number of days from March 30 and April 13 and 31 is the number of days from March 30 and April 30.
4. Adds the scales for the above steps to get the scale for the whole period:

   Scale for the period Feb 15–Apr 13 = .45 + 1.0 + .45 = 1.90

Calculates the Prorated Cycle Fee Amount

If the cycle fee is $100, the prorated cycle fee for the period Feb 15–Apr 13 will be:

1.90 * $100 = $190

Proration for Special Cases

BRM includes proration settings that address customers with cycle arrears fees who purchase, inactivate, and reactivate charge offers within the first cycle.

You choose the rating behavior when such events occur by setting the **cycle_arrear_proration** parameter in the CM configuration (**pin.conf**) file:

```
- fm_rate cycle_arrear_proration = 0|1
```

Note

You must restart the CM after you change this parameter.

When you set up proration in a charge, you choose a proration setting for when the charge offer is purchased and for when it is canceled:

- To use the purchase-time proration setting to rate periods in which a customer both purchases and inactivates a charge offer, set the **cycle_arrear_proration** parameter to **0**.
- **To use the cancellation proration setting** to rate periods in which a customer cancels a charge offer**, set the cycle_arrear_proration** parameter to **1**.

Special Cases

Some special proration cases may result in unexpected billing results:

- A customer purchases, inactivates, and reactivates a charge offer with a cycle arrears fee all in the same accounting cycle. The charge offer is set to prorate on purchase and to charge full on cancellation, and the **cycle_arrear_proration** entry in the CM **pin.conf** is set to **1**.

  In this case, instead of charging a prorated amount based on the total time the charge offer is active during the first cycle, the full amount for the cycle is charged.
- A customer purchases, inactivates, and reactivates a charge offer with a cycle arrears fee all in the same accounting cycle. The charge offer is set to no charge on purchase and to prorate on cancellation, and the **cycle_arrear_proration** entry in the CM **pin.conf** is set to **1**.

  In this case, instead of being charged nothing for the first accounting cycle, the customer is charged a prorated amount based on the time the charge offer is active.

  For example, if the customer purchases a charge offer, inactivates it after 5 days, reactivates it later for another 5 days before inactivating it, and then reactivates it later for a total of 8 days before the cycle ends, you charge the customer for 5 + 5 + 8 or 18 days of use.
- A customer purchases, inactivates, and reactivates a charge offer with a cycle arrears fee all in the same accounting cycle. The charge offer is set to charge full on purchase and to

prorate on cancellation, and the **cycle_arrear_proration** entry in the CM **pin.conf** is set to **1**.

In this case, instead of being charged for the whole accounting cycle, the customer is charged a prorated amount based on the total time during the cycle that the charge offer is active.

- With a cycle arrears fee, the customer inactivates and reactivates a charge offer in a cycle other than the cycle in which the charge offer is purchased. The charge offer has proration settings of charge full, no charge, or prorate on purchase and charge full or no charge on cancellation.

  In this case, the customer is charged a prorated amount equal to the time the charge offer is active during the accounting cycle from the last reactivation to the end of the cycle. There is no charge for all other active time periods during the accounting cycle.

- With cycle forward billing, the customer purchases, inactivates, and reactivates a charge offer in the same period. The charge offer has proration settings of charge full on purchase and prorate on cancellation.

  In this case, instead of being charged for the whole accounting cycle, the customer is charged for the whole cycle less the period of time that the charge offer is inactivated.

Note

- You can set your policy settings so that customers are not charged for short usage periods. This establishes the possibility of long cycles over 30 days for which a customer is not billed.
- For the proration settings **Charge for the full cycle** on purchase and **Don't charge this cycle** on cancellation, a cycle is always “1" or less, even if a customer purchases and cancels within a long cycle.
- The following proration settings determine what is refunded when a charge offer for cycle forward billing is inactivated or canceled:

  Charge for the full cycle: Nothing is refunded.

  Don't charge this cycle: The total cycle fee is refunded.

  Charge based on usage: The unused portion of the monthly usage fee is refunded.

Addressing Special Cases

There are several techniques you can use to address any special cases that may occur when customers inactivate and reactivate an account in the same accounting cycle:

- Inform your customers of the discrepancies.
- Do not allow your customer service representatives (CSRs) to inactivate and reactivate accounts frequently, especially during the first accounting cycle in which a charge offer is purchased.
- Because most special cases occur when you use cycle arrears fees, consider using cycle forward fees instead.

About 30-Day-Based Proration

To work in parallel with older legacy billing systems, you can use 30-day-based proration.

In older legacy billing systems, it is common to use 30 days for calculating proration, irrespective of the actual number of days in the month or the billing cycle. By default, BRM calculates proration based on the number of days in the billing cycle. You use 30-day-based proration when you have BRM with an older billing system requiring 30-day-based proration.

Note

Making 30-day-based proration work with a normal calendar year can cause unexpected behavior. See "Special Cases".

Note

30-day-based proration cannot be used with multimonth billing cycles.

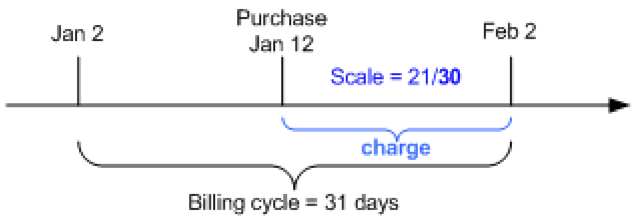
Examples of 30-Day-Based Proration

Based on your proration setting, BRM calculates the prorated cycle amount differently, which may yield different results.

In the examples below, illustrated in Figure 13-7, Figure 13-8, Figure 13-9, and Figure 13-10, February has 28 days and billing occurs on the second day of every month. The monthly cycle forward fee is $30 and the **TimestampRounding** business parameter is enabled.

Example 6: Prorated Purchase Fee with 31-day Billing Cycle

Figure 13-7 Example 6



In this example, BRM calculates the prorated cycle fee as follows:

Using 30-Day proration

1. Calculates the scale using 30 days as the base:

   scale = 21/**30 =** .70

   where 21 is the number of days from midnight January 12 to midnight February 2.

2. Calculates the prorated cycle fee:

   cycle fee amount * scale = $30.00 * .70 = $21.00

Using 31 Days in the Billing Cycle

1. Calculates the scale using 31 days as the base:

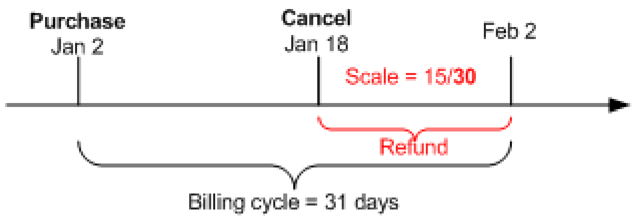
   scale = 21/**31 =** .68

   where 21 is the number of days from midnight January 12 to midnight February 2.
2. Calculates the prorated cycle fee:

   cycle fee amount * scale = $30.00 * .68 = $20.32

Example 7: Prorated Cancel Fee with 31-day Billing Cycle

Figure 13-8 Example 7



In this example, BRM calculates the cycle fee refund as follows:

Using 30-Day Proration

1. Calculates the scale using 30 days as the base:

   scale = 15/30 = .50

   where 15 is the number of days from midnight January 18 to midnight February 2.
2. Calculates the refund amount:

   cycle fee amount * scale = $30.00 * .50 = $15.00

Using 31 Days in the Billing Cycle

1. Calculates the scale using 31 days as the base:

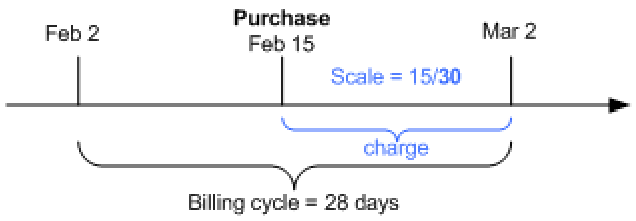
   scale = 15/31 = .48

   where 15 is the number of days from midnight January 18 to midnight February 2.
2. Calculates the refund amount:

   cycle fee amount * scale = $30.00 * .48 = $14.52

Example 8: Prorated Purchase Fee with 28-Day Billing Cycle

Figure 13-9 Example 8



In this example, BRM calculates the prorated cycle fee as follows:

Using 30-Day Proration

1. Calculates the scale using 30 days as the base:

   scale = 15/30 = .50

   where 15 is the number of days from midnight February 15 to midnight March 2.
2. Calculates the prorated cycle fee:

   cycle fee amount * scale = $30.00 * .50 = $15.00

Using 28 Days in the Billing Cycle

1. Calculates the scale using 28 days as the base:

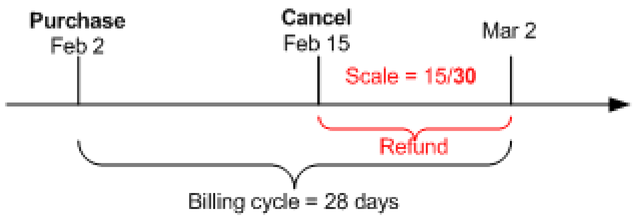
   scale = 15/28 = .54

   where 15 is the number of days from midnight February 15 to midnight March 2.
2. Calculates the prorated cycle fee:

   cycle fee amount * scale = $30.00 * .54 = $16.07

Example 9: Prorated Cancel Fee with 28-Day Billing Cycle

Figure 13-10 Example 9



In this example, BRM calculates the cycle fee refund as follows:

Using 30-Day Proration

1. Calculates the scale using 30 days as the base:

scale = 15/30 = .50

where 15 is the number of days from midnight January 18 to midnight February 2.

2. Calculates the refund amount:

   cycle fee amount * scale = $30.00 * .50 = $15.00

Using 28 Days in the Billing Cycle

1. Calculates the scale using 28 days as the base:

   scale = 15/28 = .54

   where 15 is the number of days from midnight January 18 to midnight February 2.

2. Calculates the refund amount:

   cycle fee amount * scale = $30.00 * .54 = $16.07

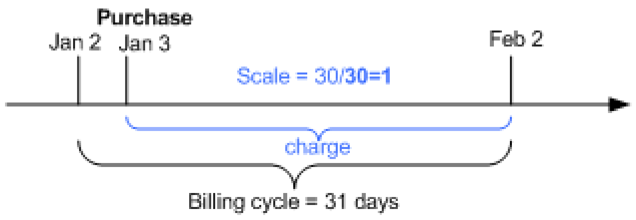
Special Cases

When using 30-day proration, there can be unexpected results, as shown in the following examples:

Example 10: Full Purchase Fee Charged When Service Is Provided for 1 Day Less

In this example, illustrated in Figure 13-11, the billing cycle is 31 days. The customer purchases the charge offer on January 3, so the scale is **1**. The customer pays the full $30 cycle fee even though the service is available for one day less.

Figure 13-11 Example 10



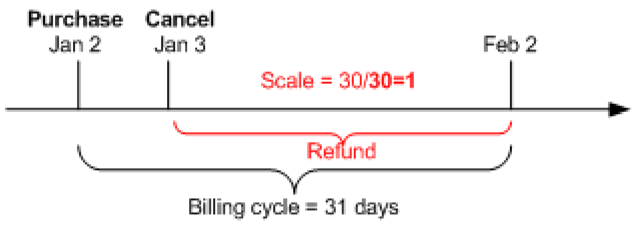
The prorated cycle fee:

cycle fee * scale = $30.00 * 1 = $30.

Example 11: Full Cancel Fee Refunded When Service Has Been Used for 1 Day

In this example, illustrated in Figure 13-12, the billing cycle is 31 days and the monthly service fee is $30. The customer purchases the charge offer on January 1 and cancels the charge offer on January 2. The scale is **1**. The customer gets a full $30 refund even though the customer owned the charge offer for one day.

Figure 13-12 Example 11



The refund amount is:

cycle fee * scale = $30.00 * 1 = $30.

Note

To avoid such situations, use the number of days in the billing cycle or number of days in the month proration settings, not the 30-day proration setting.

Enabling 30-Day-Based Proration

To enable 30-day-based proration:

1. Open the CM configuration file (*BRM_home*/**sys/cm/pin.conf**). *BRM_home* is the directory where you installed BRM components.
2. Edit the following entry:

   ```
   -fm_bill enable_30_day_proration 0
   ```

   where:

   - **0** bases proration on the number of days in the billing cycle or number of days in the month. This is the default.
   - **1** bases proration on a 30-day month.
3. Save and close the file.

Using Two Events to Prorate Charges for Charge Offers Whose Validity Ends in First Cycle

When prorating a cycle fee for a charge offer whose cycle validity period ends during the same billing cycle in which it is purchased, BRM can generate one or two events:

- **One event:** By default, BRM generates a single event to charge for the used portion of the cycle.
- **Two events:** BRM generates an event to charge for the entire cycle and then generates an event to refund the charge for the unused portion of the cycle (charge offer cycle validity end date through billing cycle end date).

For example, by default, if the charge offer is purchased on January 1 and its cycle validity period ends on January 16, only one event is generated for a prorated cycle fee for the number of days from midnight January 1 to midnight January 16. If you specify to create two events, the first event is generated for a cycle fee for the number of days from midnight January 1 to midnight February 1, and a second event is generated for a refund for the number of days from midnight January 16 to midnight February 1.

To enable this feature, run the **pin_bus_params** utility to change the **CreateTwoEventsInFirstCycle** business parameter. For information about this utility, see *BRM Developer's Guide*.

To use two events to prorate a cycle fee for a charge offer whose cycle validity period ends in the first cycle:

1. Go to *BRM_home***/sys/data/config**.
2. Create an XML file from the **/config/business_params** object:

   ```
   pin_bus_params -r -c "Subscription" bus_params_subscription.xml
   ```
3. In the XML file, change **disabled** to **enabled**:

   ```
   <CreateTwoEventsInFirstCycle>enabled</CreateTwoEventsInFirstCycle>
   ```
4. Save the file as **bus_params_subscription.xml**.
5. Load the XML file into the BRM database:

   ```
   pin_bus_paramsbus_params_subscription.xml
   ```
6. Stop and restart the CM.

Prorating Cycle Fees after a Discount Purchase or Cancellation

When an off-cycle discount is purchased or canceled and the discount is configured to support any of the charge offer's cycle forward event types, the cycle fees are discounted for the duration of the cycle for which the discount is valid.

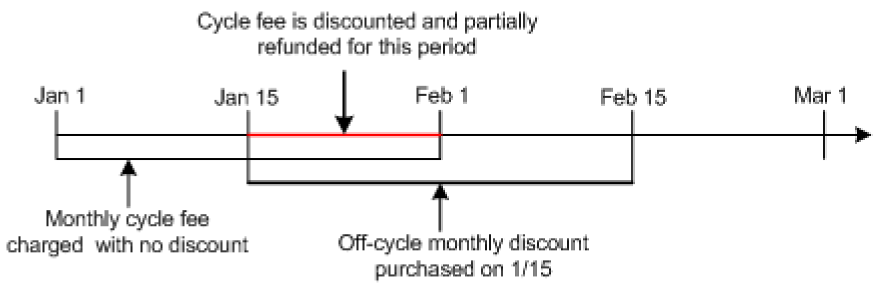
Examples of Cycle Fee Proration

These examples show how cycle fee proration works.

Example 12: Cycle Fee Is Refunded after a Discount Purchase

In this example, illustrated in Figure 13-13, a $30 monthly cycle fee is charged on January 1. An off-cycle discount is purchased on January 15 that discounts 10% of the monthly fee. The 10% discount is applied to the prorated monthly charge of $15 from Jan 15–Feb 1. A refund amount of $1.50 is credited to the account or service balance.

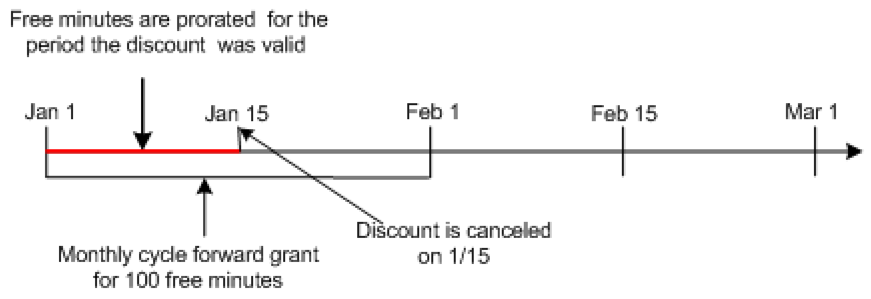
Figure 13-13 Example 12


Cycle fee is discounted and partially refunded for this period
Jan 1
Jan 15
Feb 1
Feb 15
Mar 1
Monthly cycle fee charged with no discount
Off-cycle monthly discount purchased on 1/15

Example 13: Minutes Are Prorated after a Discount Cancellation

In this example, illustrated in Figure 13-14, a cycle forward monthly event grants 100 minutes. The discount is canceled on January 15. The account made 200 minutes of calls between January 1 and January 15 at $0.10 per minute. The 100 minutes are prorated between January 1 and January 15. The account is granted 50 minutes and is charged $0.10 per minute for the remaining 150 minutes.

Figure 13-14 Example 13


Free minutes are prorated for the period the discount was valid
Jan 1
Jan 15
Feb 1
Feb 15
Mar 1
Discount is canceled on 1/15
Monthly cycle forward grant for 100 free minutes

Example 14: Canceled Discount Proration Is Not Taken into Account When Charge Offer Is Canceled

If you cancel a discount for a cycle fee and then you cancel the charge offer that owns that cycle fee, the prorated cycle fee is refunded without considering any discount that was applied. Therefore, the refunded amount might be more than the charged amount. However, this can be corrected by rerating the cycle event.

In this example, an account owns a charge offer with a cycle fee of $60, cancellation proration settings set to **Do not charge for the cycle**, and a discount of 10% on the cycle fee. At the beginning of the cycle, $54 is charged ($60 cycle fee minus $6 discount). In the middle of the cycle, if the discount is canceled, $3 is charged back. Later, if the charge offer is canceled and $60 is refunded (because the cancellation proration setting is **Do not charge for the cycle**). Therefore, the net charge for the cycle is a credit of $3. If you rerate the events for the cycle, the expected charge of $60 will be applicable.

Prorating Cycle Fees When a Discount's Cycle Start or End Date Is Changed

When a discount's cycle start or end dates are changed, cycle fees may be discounted and refunded or charged depending on the new cycle start and end dates.

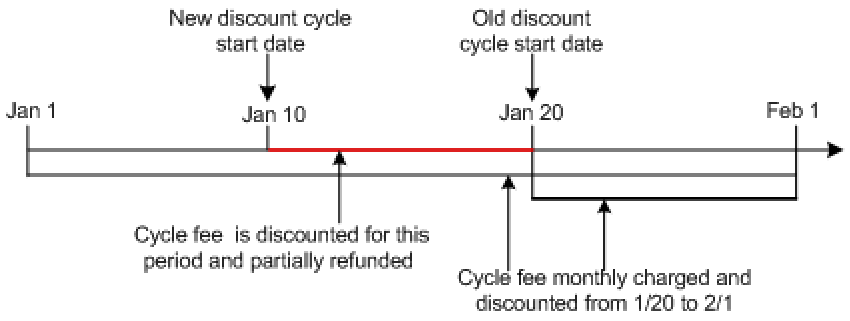
Examples of Cycle Fee Proration

These examples show how cycle fee proration works.

Example 15: Cycle Fee Is Refunded When a Discount's Cycle Start Date Is Changed

In this example, illustrated in Figure 13-15, a monthly cycle fee is charged on January 1. The monthly fee is discounted from Jan 20–Feb 1, and the discounted amount is credited to the account balance. When the discount cycle start date is changed to January 10, the monthly fee is discounted from Jan 10–Jan 20, and the discounted amount is credited to the account balance.

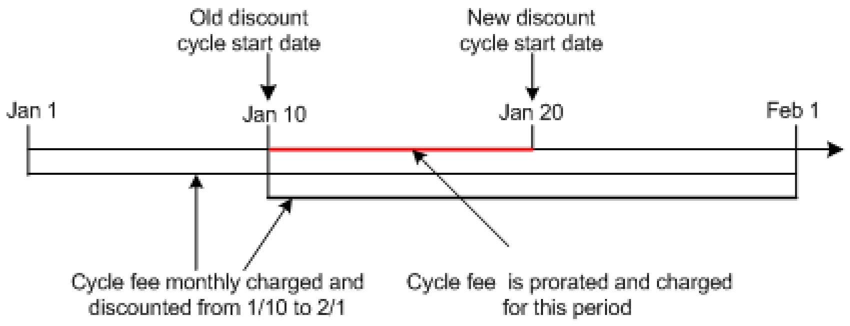
Figure 13-15 Example 15


New discount cycle start date
Old discount cycle start date
Jan 1
Jan 10
Jan 20
Feb 1
Cycle fee is discounted for this period and partially refunded
Cycle fee monthly charged and discounted from 1/20 to 2/1

Example 16: Cycle Fee Is Charged When a Discount's Cycle Start Date Is Changed

In this example, illustrated in Figure 13-16, a monthly cycle fee is charged on January 1. The monthly fee is discounted from Jan 10–Feb 1, and the discounted amount is credited to the account balance. When the discount cycle start date is changed to January 20, the discounted cycle fee amount is charged from Jan 10–Jan 20 because the discount is no longer valid during this period.

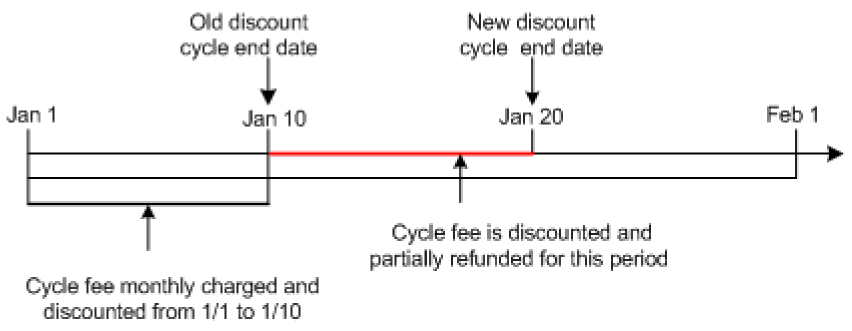
Figure 13-16 Example 16


Old discount cycle start date
New discount cycle start date
Jan 1
Jan 10
Jan 20
Feb 1
Cycle fee monthly charged and discounted from 1/10 to 2/1
Cycle fee is prorated and charged for this period

Example 17: Cycle Fee Is Refunded When Discount's Cycle End Date Is Changed

In this example, illustrated in Figure 13-17, a monthly cycle fee is charged on January 1. The monthly fee is discounted from Jan 1–Jan 10. When the discount cycle end date is changed to January 20, the cycle fee is discounted from Jan 10–Jan 20, and the discounted amount is credited to the account balance.

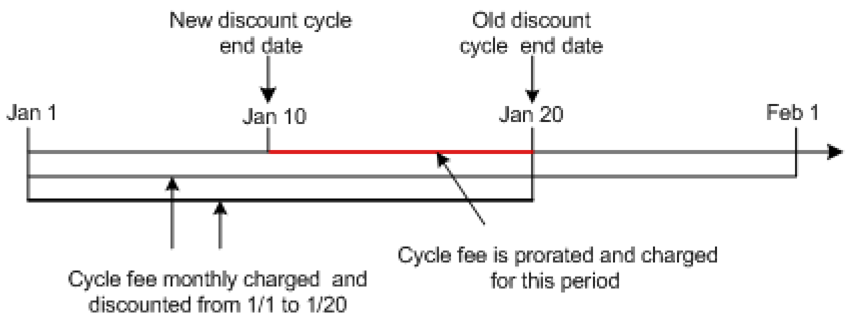
Figure 13-17 Example 17


Old discount cycle end date
New discount cycle end date
Jan 1
Jan 10
Jan 20
Feb 1
Cycle fee is discounted and partially refunded for this period
Cycle fee monthly charged and discounted from 1/1 to 1/10

Example 18: Cycle Fee Is Charged When Discount's Cycle End Date Is Changed

In this example, illustrated in Figure 13-18, a monthly cycle fee is charged on January 1. The monthly fee is discounted from Jan 1–Jan 20. When the discount cycle end date is changed to January 10, the discounted cycle fee amount is charged from Jan 10–Jan 20.

Figure 13-18 Example 18


New discount cycle end date
Old discount cycle end date
Jan 1
Jan 10
Jan 20
Feb 1
Cycle fee is prorated and charged for this period
Cycle fee monthly charged and discounted from 1/1 to 1/20

Prorating Cycle Fees When Accounting Cycle and Cycle Fee Start Date Align

By default, BRM applies a product's proration rules to all purchases and cancellations, regardless of when they occur in a customer's accounting cycle.

However, you can configure BRM to not apply the product’s proration rules when customers purchase or cancel the product on their accounting cycle date. For example, if a customer’s accounting cycle starts on May 15 and ends on June 14, and the customer purchases a product on May 15, BRM can charge the full cycle fee to the May 15 accounting cycle regardless of the product level prorate settings. Likewise, if the customer cancels the product on June 14, no refund is given for the May 15 accounting cycle. If the customer already has the service and cancels it on May 15, the full refund will be given for the May 15 accounting cycle.

You can configure this by enabling the **ApplyProrationRules** business parameter in the **subscription** instance of the **/config/business_params** object.

To enable BRM to apply proration rules:

1. Go to *BRM_home***/sys/data/config.**

2. Use the following command to create an editable XML file from the **subscription** instance of the **/config/business_params** object:

   ```
   pin_bus_params -r BusParamsSubscription bus_params_subscription.xml
   ```

   This command creates an XML file named **bus_params_subscription.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_subscription.xml.out**, set **ApplyProrationRules** to **enabled**:

   ```
   <ApplyProrationRules>enabled</ApplyProrationRules>
   ```

   > ⚠ **Caution**
   >
   > BRM uses the XML in this file to overwrite the existing instance of the **/config/business_params** object. Use care when updating parameters in the file.

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

   ```
   pin_bus_params bus_params_subscription.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 Connection Manager (CM).

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

14 Managing Large Billing Runs

Learn how to manage large billing runs in Oracle Communications Billing and Revenue Management (BRM).

Topics in this document:

- Billing Only Specified Accounts and Bill Units
- Splitting a Billing Run into Multiple Runs
- Managing Bill Due Dates

Implementing these features includes editing the billing run configuration file (**pin_bill_run_control.xml**).

When splitting a billing run or using customized due dates, you must run the **pin_bill_day** script manually. See "Running the pin_bill_day Script for Bill Run Management".

Billing Only Specified Accounts and Bill Units

To bill a single account or a limited set of accounts with the **pin_bill_day** script, you specify the accounts and their bill units in a modified version of the billing run configuration file (**pin_bill_run_control.xml**). In this case, BRM does not perform a database search but retrieves the account and bill unit information directly.

> **Note**
>
> If you specify a paying bill unit that is part of a bill unit hierarchy, ensure that all nonpaying bill units are billed first.

1. Open the *BRM_home***/apps/pin_billd/pin_bill_run_control.xml** file.
2. Edit the file to specify the accounts and their bill units to bill.

   Add an **Account** and **Billinfo** child element to the **BillingList** parent element for each account and bill unit to include. In the child element, specify the POID of the account and bill unit.

   > **Note**
   >
   > To bill specific accounts and bill units, you must include both the account POID and bill unit POID. If only one is specified, the account or bill unit is not billed.

   A **BillingList** parent element looks like this:

   ```
   <BillingList>
       <Account>Account_POID</Account>
       <Billinfo>Bill_unit_POID</Billinfo>
   </BillingList>
   ```

For example, the following **BillingList** parent element generates bills only for the account with POID 55612 and its bill unit with POID 34589:

```
<BillingList>
    <Account>55612</Account>
    <Billinfo>34589</Billinfo>
</BillingList>
```

3. (Optional) Delete or comment out any billing DOM lists or billing segment lists (specified by **DOMList** and **BillSegmentList** parent elements). If you include accounts and bill units along with DOMs or billing segments, only the account's bill units are processed.
4. Save the file under a different name and close it. For example, when billing a single account, include the account POID in the file name (such as **pin_bill_run_account_8445.xml**). If billing a group of accounts, include the account range or reason for billing in the file name.

   Note

   When you run **pin_bill_day**, the default *BRM_home***/apps/pin_billd/business_configuration.xsd** file and filename must be in the same directory.

5. *Manually* run the **pin_bill_day** script using this command:

   ```
   pin_bill_day -file filename
   ```

   where *filename* is the name and location of the modified version of the billing run configuration file. If the file is in a different directory from which you run the **pin_bill_day** script, you must also include the entire path for the file.

   Note

   - The **-file** parameter when used with **pin_bill_day**, affects only the **pin_bill_accts** utility; it does not apply to other billing utilities run by the **pin_bill_day** script. For example, **pin_cycle_fees** which performs a database search to find all accounts with cycle forward fees that are due, does not use the accounts passed in with the **-file** parameter.
   - Ensure that the accounts specified in the billing run configuration file reside on the same database schema where **pin_bill_day** is run. If the file contains accounts from different database schemas, **pin_bill_day** reports an error. See "Setting Up Billing in a Multischema Environment".
   - When you run **pin_bill_day** with the **-file** parameter, do not run it as a **cron** job. If you do, depending on the restrictions in *filename*, some bill units might never be billed.

Splitting a Billing Run into Multiple Runs

When you use the **pin_bill_day** script to run billing, the billing run includes *all* bill units whose current accounting cycle end date is any day before midnight (12:00:00 a.m.) of the day on which the billing run takes place.

To reduce the load and duration of billing runs triggered by the script, you can split large, lengthy billing runs into smaller billing runs based on billing days of month (DOMs) and billing segments. The smaller billing runs can overlap or occur at different times.

Splitting a billing run into multiple runs includes two steps:

1. Configure auto-triggered billing.
2. Configure the split billing run.

Configuring Auto-Triggered Billing for Bill Run Management

To split large billing runs into smaller billing runs, you first disable auto-triggered billing on the days that you process the smaller billing runs. If auto-triggered billing is enabled on those days, it reduces your ability to control the way your billing load is distributed.

For example, the billing cycle for two million customers ends on August 1. To reduce the number of bills finalized on August 1, you split the billing run into four smaller runs, each of which includes about 500,000 bill units. You process one smaller billing run per day from August 1 through August 4. If auto-triggered billing is enabled on those days, bill-triggering events might cause BRM to finalize some bills in the smaller billing runs before you process the runs. For example, bill-triggering events might cause BRM to finalize 1,250,000 bills on August 1 instead of only 500,000.

To configure auto-triggered billing for bill run management:

1. Set a system-wide billing delay. See "Setting Up Delayed Billing".

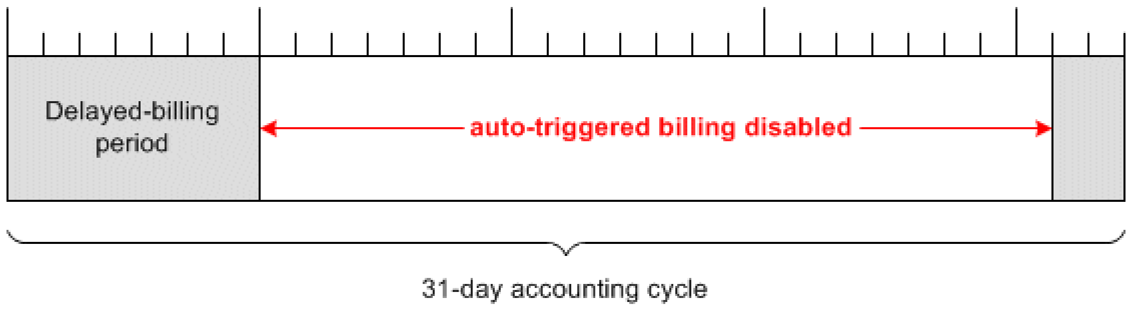
   By default, auto-triggered billing is always enabled. To disable it, you must set a billing delay in your BRM system.

   > **Note**
   >
   > If you use delayed billing, skip this step. A billing delay is already set in your system. For more information, see "Setting Up Delayed Billing".

   By default, after you set a billing delay, auto-triggered billing is enabled for only the delay period and the last two days of each bill unit's accounting cycle.

   For example, if your delayed billing cycle is 7 days long, auto-triggered billing is disabled during the following days in a 31-day accounting cycle as shown in Figure 14-1.

Figure 14-1 Auto-Triggered Billing Disabled in Accounting Cycle



> **Note**
>
> If a bill-triggering event occurs during the delayed-billing period, the bill is only partially processed: rollovers and cycle fees are applied, but the bill is not finalized. If a bill-triggering event occurs during the last two days of the cycle, the bill is finalized.

2. (Optional) Change the number of days auto-triggered billing is enabled at the end of each accounting cycle. See "Configuring Auto-Triggered Billing".

Configuring a Split Billing Run

To split a billing run into smaller billing runs, you configure the smaller billing runs in multiple versions of the billing run configuration file (**pin_bill_run_control.xml**). Each of the smaller billing runs is limited to bill units associated with one or both of the following:

- **Specified accounting days of month (DOMs)**. Bill units associated with any other billing DOM are excluded from the billing run.
- **Specified billing segments**. Bill units associated with any other billing segment are excluded from the billing run.

  To restrict the smaller billing runs to specified billing segments, you must first set up billing segments in your system and then associate bill units with them. See "Load Balancing Billing Runs".

> **Note**
>
> When you split a large billing run into smaller billing runs:
>
> – Configure the smaller billing runs so that in total they include all due bill units.
>
> – To ensure that no bill unit remains unbilled, periodically run the **pin_bill_day** script without the **-file** parameter.
>
> – Do not include the *same* billing segment in multiple small billing runs. If you do, your accounts receivable (A/R) data may become inaccurate.

To split a billing run into smaller billing runs:

1. Open the **pin_bill_run_control.xml** file in an XML editor or a text editor.

   By default, the file is in the *BRM_home***/apps/pin_billd** directory.

   > **Note**
   >
   > You also edit this file to specify billing run-time due date adjustments. See "Specifying Due Date Adjustments in a Billing Run".

2. Edit the file to restrict a billing run to bill units associated with one of the following:
   - For specific billing DOMs, see "Including Only Specified Billing DOMs in Billing Runs".
   - For specific billing segments, see "Including Only Specified Billing Segments in Billing Runs".

Note

Only one smaller billing run at a time can be configured in a billing run configuration file.

3. Save the file.

Note

To create multiple versions of this file, save the file under a different name for each of the smaller billing runs. For example, if a version of the file limits a smaller billing run to billing segment 1001, save the file as **pin_bill_run_control_BS1001.xml**. The **pin_bill_day** script can take any XML file name as a parameter if the file's contents conform to the appropriate schema definition. See "Validating Your Billing Run Configuration File Edits".

4. Repeat steps 1 through 3 as often as necessary to configure a set of smaller billing runs that includes all due bill units in a daily billing run.
5. *Manually* run the **pin_bill_day** script with each version of the XML file by using this command:

```
pin_bill_day -file filename
```

where *filename* is the name and location of a version of the billing run configuration file. If you run the command in a different directory from where *filename* is located, you must also include the entire path for the file.

In addition, *filename* must be in the same directory as the default *BRM_home*/**apps/pin_billd/business_configuration.xsd** file.

Note

- The **-file** parameter when used with **pin_bill_day**, affects only the **pin_bill_accts** utility; it does not apply to other billing utilities run by the **pin_bill_day** script. For example, **pin_cycle_fees** which performs a database search to find all accounts with cycle forward fees that are due, does not use the accounts passed in with the **-file** parameter.
- Ensure that the accounts specified in the billing run configuration file reside on the same database schema where **pin_bill_day** is run. If the file contains accounts from different database schemas, **pin_bill_day** reports an error. See "Setting Up Billing in a Multischema Environment".
- When you run **pin_bill_day** with a configuration file, do not run it as a **cron** job. If you do, depending on the restrictions in configuration file, some bill units might never be billed.

About Sponsored Charges in Split Billing Runs

If the following conditions occur when you split a billing run, sponsored cycle forward charges might appear in the sharing group owner's bill one cycle late:

- Sharing group owner account and member account have the same billing DOM.
- Sharing group owner account and member account belong to different billing segments.

Including Only Specified Billing DOMs in Billing Runs

To include bill units associated only with specified billing DOMs in a billing run, add a **DOM** child element to the **DOMList** parent element in the billing run configuration file for each day whose bill units you want to include.

A **DOMList** parent element looks like this:

```
<DOMList>
    <DOM>---gDay1</DOM>
    <DOM>---gDay2</DOM>
    <DOM>---gDay3</DOM>
</DOMList>
```

where gDay is any two-digit value from **01** through **31**.

For example, to include only bill units whose billing DOM is 1 or 15, add child elements:

```
<DOMList>
    <DOM>---01</DOM>
    <DOM>---15</DOM>
</DOMList>
```

If the **DOMList** parent element is omitted, bill units associated with any billing DOM can be included in the billing run.

Including Only Specified Billing Segments in Billing Runs

To include bill units associated only with specified billing segments in a billing run, add a **BillSegment** child element to the **BillSegmentList** parent element in the billing run configuration file for each billing segment whose bill units you want to include.

A **BillSegmentList** parent element looks like this:

```
<BillSegmentList>
    <BillSegment>ID</BillSegment>
    <BillSegment>ID</BillSegment>
    <BillSegment>ID</BillSegment>
</BillSegmentList>
```

where ID is the ID of any billing segment defined in the **/config/billing_segment** object in your BRM system.

For example, to include only bill units associated with billing segments 101 or 102, add child elements:

```
<BillSegmentList>
    <BillSegment>101</BillSegment>
    <BillSegment>102</BillSegment>
</BillSegmentList>
```

For information about billing segments, see "Load Balancing Billing Runs".

If the **BillSegmentList** parent element is omitted, bill units associated with any billing segment can be included in the billing run.

Sample Billing Run Configuration File

The following is a sample **pin_bill_run_control.xml** file. Billing run loads are restricted to billing DOMs and billing segments specified in the **bold** elements.

```
<DOMList>
    <DOM>---03</DOM>
    <DOM>---07</DOM>
    <DOM>---15</DOM>
</DOMList>
<BillSegmentList>
    <BillSegment>101</BillSegment>
    <BillSegment>102</BillSegment>
</BillSegmentList>
<DueDateAdjustment Length=5 >
    <PaymentTerm id = 1001 />
    <PaymentTerm id = 1002 />
</DueDateAdjustment>
<DueDateAdjustment Length = 7/>
```

For information about the **DueDateAdjustment** and **PaymentTerm** elements, see "Specifying Due Date Adjustments in a Billing Run".

Validating Your Billing Run Configuration File Edits

After editing the XML file, you use the file name as a parameter when you run the **pin_bill_day** script. The script passes the file name to the **pin_bill_accts** utility, which validates the contents of the XML file against its schema definition. If the contents do not conform to the schema definition, the utility returns an error. The schema definition is *BRM_home***/xsd/pin_bill_run_control.xsd**.

The XML file is not directly linked to its schema definition file. Instead, it is linked to the *BRM_home***/sys/data/config/business_configuration.xsd** reference file.

Managing Bill Due Dates

This section explains how to manage bill due dates by performing the following tasks:

- Managing Payment Terms. Payment terms enable you to use different methods to calculate due dates for bills that have the same payment method.
- Managing Billing Calendars. Billing calendars enable BRM to implement payment terms based on *business* days.
- Specifying Due Date Adjustments in a Billing Run. Billing run-time adjustments enable you to accommodate operational delays in your billing process.

By default, the bill due date for any payment method is 30 days after the date the bill is finalized. You can change default due dates only at the payment-type level.

The BRM delayed billing feature enables billing for all the bill units in your system to be run a specified number of days after the end of their billing cycle. If you use delayed billing, be careful to avoid configuring bill due dates that occur before bills are finalized. For example, if your system has a 14-day billing delay and you configure a bill due date that is fewer than 14 days after the end of a bill unit's billing cycle, the bill due date will occur before the bill is finalized.

Managing Payment Terms

A *payment term* specifies how to calculate the due date of a bill. You use payment terms to set due dates a specified number of days after the billing cycle end date or on a specified day of the month. For example, payment terms can be linked to functions that set bill due dates as follows:

- 21 days after the billing cycle end date
- 15 business days after the billing cycle end date
- The second Tuesday of the month

You can define as many payment terms as you need in your BRM system. Each payment term can be associated with one or more **/payinfo** objects, but a **/payinfo** object can be associated with only one payment term at a time. The **/payinfo** object's payment term applies to all bills associated with the object.

Payment terms enable you to use different methods to calculate due dates for bills that have the same payment method. For example, by default, all bills paid by direct debit are due on the day the bill is finalized. Customer A pays a monthly bill by direct debit and has paychecks automatically deposited every two weeks. However, Customer A wants the account debited on the third Tuesday of every month regardless of when the bill is finalized. To do so without affecting other direct debit customers, you can create a third-Tuesday-of-the-month payment term and associate it with the **/payinfo** object to which customer A's bill is linked.

To manage payment terms, you perform these tasks:

- Editing the Payment Terms Configuration File
- Loading Payment Terms
- Assigning Payment Terms to Bill Units

Editing the Payment Terms Configuration File

You configure all the payment terms in your BRM system in the *BRM_home***/sys/data/config/pin_payment_term.xml** file.

In the file, the **PaymentTerms** parent element must contain a **PaymentTerm** child element for each payment term in your system. A **PaymentTerm** child element looks like this:

```
<PaymentTerm ID="int">description</PaymentTerm>
```

where *int* is a payment term ID. For more information, see Table 14-1.

To create a payment term, add a **PaymentTerm** child element to the **PaymentTerms** parent element. In the child element, specify values for the items listed in Table 14-1.

Table 14-1 Payment Term Elements

XML Element or Attribute	Description	Possible Values
ID	The ID of the payment term.	Any nonnegative integer. **Note** • Payment term ID 0 is the default payment term. • Payment term IDs 1 through 1000 are reserved for BRM use. **Important:** In the PCM_OP_BILL_POL_CALC_PYMT_DUE_T policy opcode, the ID number must be associated with a function that calculates bill due dates. See the chapter about billing in *BRM Opcode Guide*.
description	An explanation of the payment term. For example, **3rd Tuesday of the month**.	Any string. **Note:** This string is mapped to the PIN_FLD_DESCR field in the **/config/payment_term** object, which can be used to populate a list of payment terms in a user interface (UI). When creating the string, take any UI length restrictions into account.

Sample Payment Terms Configuration File

The following is a sample **pin_payment_term.xml** file:

```
<?xml version="1.0" encoding="UTF-8"?>

<BusinessConfiguration xmlns="http://www.portal.com/schemas/BusinessConfig"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.portal.com/schemas/BusinessConfig
business_configuration.xsd">

<!-- Sample file. Modify according to guidelines -->

<PaymentTermConfiguration>
    <PaymentTerms>
        <PaymentTerm ID="1001">17 days</PaymentTerm>
        <PaymentTerm ID="1002">14 business days</PaymentTerm>
        <PaymentTerm ID="1003">3rd Tuesday of the month</PaymentTerm>
    </PaymentTerms>
</PaymentTermConfiguration>

</BusinessConfiguration>
```

Loading Payment Terms

To set up payment terms in your system, edit the payment terms configuration file (**pin_payment_term.xml**), and then load its contents into the **/config/payment_term** object in the BRM database.

1. Open the **pin_payment_term.xml** file in an XML editor or a text editor.

   The file is in the *BRM_home***/sys/data/config** directory.

2. Enter the appropriate information into the file. See "Editing the Payment Terms Configuration File ".
3. Save the file.
4. Use the following command to run the "load_pin_payment_term" utility from the directory in which the **pin_payment_term.xml** file is located:

   ```
   load_pin_payment_term pin_payment_term.xml
   ```

   If you do not run the utility from the directory in which **pin_payment_term.xml** is located, include the complete path to the file; for example:

   ```
   load_pin_payment_term BRM_home/sys/data/config/pin_payment_term.xml
   ```
5. Stop and restart the Connection Manager (CM).
6. For each payment term in your system, customize the PCM_OP_BILL_POL_CALC_PYMT_DUE_T policy opcode to specify the function and parameters to use to calculate the due dates of bills associated with the payment term. See *BRM Opcode Guide*.

Updating the pin.conf File to Use Payment Terms

To use configured payment terms, update the **pin.conf** file.

1. Open the *BRM_home*/**sys**/**cm**/**pin.conf** file in a text editor.
2. Add the following line:

   ```
   - cm_cache fm_cust_pol_paymentterm_cache 256, 40960, 64
   ```
3. Save and close the file.
4. Stop and restart the CM.

Assigning Payment Terms to Bill Units

To assign a payment term to a bill unit, you associate the payment term with the **/payinfo** object to which the **/billinfo** object is linked.

To do this at account creation time, call the PCM_OP_CUST_COMMIT_CUSTOMER opcode to put the ID of the payment term into the PIN_FLD_PAYMENT_TERM field of the appropriate **/payinfo** object. (PCM_OP_CUST_COMMIT_CUSTOMER calls the PCM_OP_CUST_SET_PAYINFO opcode to perform this task.)

Note

- All the payment terms in a BRM system are stored in the **/config/payment_term** object. In the object, the PIN_FLD_PAYMENT_TERMS array contains the ID and description (PIN_FLD_DESCR) of each payment term.
- If you do not assign a payment term to a **/payinfo** object at account creation time, BRM automatically assigns the default payment term ID 0 to the **/payinfo** object. This occurs even if there are no payment terms in your system.

To do this at account maintenance time, call the PCM_OP_CUST_UPDATE_CUSTOMER opcode to perform the task described in the preceding paragraphs.

Note

When assigning payment terms based on days of month (DOMs), ensure the payment term is compatible with the bill unit's billing DOM.

For example, account A's billing DOM is 20 and its payment term is “3rd Tuesday of the month." When billing is run on August 20, 2004, the due date of account A's August bill is set to September 21 because the third Tuesday of August (August 17) has passed. When billing is run on September 20, the due date of account A's September bill is also set to September 21. Thus, account A has two bills due on the same day, which may be undesirable.

See *BRM Opcode Guide*.

Managing Billing Calendars

By default, a *billing calendar* contains a list of days on which bills cannot be due.

Billing calendars enable BRM to implement payment terms based on *business* days. To calculate due dates for bills associated with such payment terms, BRM must determine which days of the year are considered business days and which are not. To do so, it uses billing calendars to exclude days such as weekends, holidays, and other user-specified nonbilling days from the calculation.

For example, if a billing cycle ends on December 10, 2016, and its payment term adds 15 business days to that date, a billing calendar can be used to prevent the bill's payment being due on New Year's Eve.

You can add multiple billing calendars to your system to accommodate different countries and customers. For example, a US billing calendar would include Thanksgiving and Independence Day (July 4) as nonbilling days when they occur on weekdays.

When calculating due dates for bills associated with payment terms based on *business* days, the PCM_OP_BILL_POL_CALC_PYMT_DUE_T policy opcode uses billing calendars to exclude days such as weekends, holidays, and any other user-specified nonbilling day from the calculation.

To manage billing calendars, you perform these tasks:

- Setting Up Billing Calendars
- Editing the Billing Calendar Configuration File
- Updating Billing Calendars
- Associating Billing Calendars with Payment Terms

Setting Up Billing Calendars

To set up billing calendars in your system, edit the billing calendar configuration file (**pin_calendar.xml**) and then load its contents into **/config/calendar** objects in the BRM database (each calendar is loaded into a separate object).

To set up billing calendars:

1. Open the **pin_calendar.xml** file.

   By default, the file is in the *BRM_home***/sys/data/config** directory.

2. Enter the appropriate information into the file. See "Editing the Billing Calendar Configuration File ".

   > **Note**
   >
   > - Bill run management includes a default billing calendar. Its case-sensitive name is **default**. By default, the PCM_OP_BILL_POL_CALC_PYMT_DUE_T policy opcode uses this calendar to calculate due dates for payment terms based on *business* days.
   > - When you edit the configuration file, if you unintentionally change or delete the calendar, due date calculations based on business days may be incorrect.

3. Save the file.
4. Use the following command to run the "load_pin_calendar" utility from the directory in which the **pin_calendar.xml** file is located:

   ```
   load_pin_calendar pin_calendar.xml
   ```

   > **Note**
   >
   > - When you run the utility, the **pin_calendar.xml** and **business_configuration.xsd** files must be in the same directory. By default, both files are in *BRM_home***/sys/data/config**.
   > - This utility needs a configuration (**pin.conf**) file in the directory from which you run the utility. For information, see "Connecting BRM Utilities" in *BRM System Administrator's Guide*.

   If you do not run the utility from the directory in which **pin_calendar.xml** is located, include the complete path to the file, for example:

   ```
   load_pin_calendar BRM_home/sys/data/config/pin_calendar.xml
   ```

5. Stop and restart the CM.

Editing the Billing Calendar Configuration File

You configure all the billing calendars in your BRM system in the *BRM_home***/sys/data/config/pin_calendar.xml** file.

> **Note**
>
> This configuration file contains one predefined billing calendar. Its case-sensitive name is **default**. By default, the PCM_OP_BILL_POL_CALC_PYMT_DUE_T policy opcode uses this calendar to calculate due dates for payment terms based on *business* days.
>
> When you edit the configuration file, if you unintentionally change or delete the calendar, due date calculations based on business days may be incorrect.

To edit this configuration file, open it in an XML editor or a text editor.

In the file, the **CalendarConfiguration** parent element must contain a **Calendar** child element for each billing calendar in your system. A **Calendar** child element looks like this:

```
<Calendar name="calendar_name">
    <Date>
        <!-- day_description -->
        <Day>---dd</Day>
        <Month>--mm--</Month>
        <Year>yyyy</Year>
    </Date>
</Calendar>
```

For an example, see "Sample Billing Calendar Configuration File".

To add a calendar to the file, see "Adding Calendars".

To add a day to a calendar, see "Adding Days to Calendars".

Adding Calendars

To create a billing calendar, add a **Calendar** child element to the **CalendarConfiguration** parent element. In the child element, specify values for the items listed in Table 14-2.

Table 14-2 Calendar Elements

XML Element or Attribute	Description	Possible Values
name	The name of the billing calendar, such as Gregorian, European_Holidays, and Korean. The name of the default billing calendar is **default**.	Any string. Maximum length is 255 characters. **Note** • The name is case sensitive. • This string is mapped to the PIN_FLD_NAME field in the **/config/calendar** object, which can be used to populate a list of billing calendars in a user interface (UI). When creating the string, take any UI length restrictions into account.
Date	A day on which you *do not* want bill payments to be due. A **Calendar** child element can have multiple **Date** elements. For example, in a US billing calendar, you might include a **Date** element for every US holiday (New Year's Day, President's Day, Memorial Day, and so on). **Note:** The default due date function that uses billing calendars (**fm_bill_pol_add_n_bus_days**) automatically skips weekends, so you do not need to create a *Date* element for Sundays and Saturdays in calendars used only by that function. See the chapter about billing in *BRM Opcode Guide*.	See "Adding Days to Calendars".

Adding Days to Calendars

In a **Calendar** child element, you must add a **Date** element for each day on which you do not want bill payments to be due. A **Date** element looks like this:

```
<Date>
    <!-- description_of_day -->
    <Day>---dd</Day>
    <Month>--mm--</Month>
    <Year>yyyy</Year>
</Date>
```

To add a day to a calendar, add a **Date** element to the **Calendar** element. In the **Date** element, specify values for the items listed in Table 14-3.

Table 14-3 Date Element Entries

XML Element or Attribute	Description	Possible Values
Day	A day of the month.	**1** through **31**.
Month	A month of the year.	**1** through **12**.
Year	The year in which the date is a nonbilling day.	One of the following: • To associate the date with *one year only*, specify the appropriate year in *yyyy* format (for example, **2005**). • To associate the date with *all years* (a*recurring* nonbilling day), use **0000**. • For example, to make December 25 (Christmas) a recurring nonbilling day, use these values: **Day 25**, **Month 12**, **Year 0000**.

Sample Billing Calendar Configuration File

The following is a sample **pin_calendar.xml** file:

```
<?xml version="1.0"?>
<BusinessConfiguration xmlns="http://www.portal.com/schemas/BusinessConfig"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.portal.com/schemas/BusinessConfig
business_configuration.xsd">
<!-- Sample file. Modify according to guidelines -->

    <CalendarConfiguration>
        <Calendar name="default">
            <!-- holiday specific to indicated year -->
            <Date>
                <!-- Thanksgiving -->
                <Day>---25</Day>
                <Month>--11--</Month>
                <Year>2004</Year>
            </Date>
        </Calendar>
        <Calendar name="Western Australia">
            <!-- recurring holiday on same date. use 0 for year value -->
```

```
                <Date>
                    <!-- Christmas -->
                    <Day>---25</Day>
                    <Month>--12--</Month>
                    <Year>0000</Year>
                </Date>
                <!-- holiday date specific to indicated year -->
                <Date>
                    <!-- Anzac Day -->
                    <Day>---26</Day>
                    <Month>--04--</Month>
                    <Year>2004</Year>
                </Date>
                <Date>
                    <!-- Australia Day -->
                    <Day>---26</Day>
                    <Month>--01--</Month>
                    <Year>2005</Year>
                </Date>
            </Calendar>
        </CalendarConfiguration>
    </BusinessConfiguration>
```

Validating Your Billing Calendar Configuration File Edits

After editing the contents of the XML file, you use the "load_pin_calendar" utility to load the contents of the file into **/config/calendar** objects in the BRM database. See "Setting Up Billing Calendars".

Before loading the contents of the file, the utility validates the contents against the file's schema definition. If the contents do not conform to the schema definition, the load operation fails. The schema definition is in this file:

*BRM_home***/xsd/pin_calendar.xsd**

The XML file is not directly linked to its schema definition file. Instead, it is linked to this XSD *reference* file:

*BRM_home***/sys/data/config/business_configuration.xsd**

Updating Billing Calendars

To update billing calendars, re-edit the billing calendar configuration file, and then run the "load_pin_calendar" utility to load the updated contents of the file into **/config/calendar** objects in the BRM database. See "Setting Up Billing Calendars" for the complete procedure.

Note

This utility overwrites existing billing calendars. When updating billing calendars, you cannot load new calendars only. You must load the complete set of billing calendars each time you run the utility.

To add the newly loaded data to the CM cache, stop and restart the CM.

Associating Billing Calendars with Payment Terms

When using payment terms based on business days to calculate bill due dates, the PCM_OP_BILL_POL_CALC_PYMT_DUE_T policy opcode uses billing calendars to omit nonbilling days from the calculation.

To associate a billing calendar with such payment terms, see the chapter about billing in *BRM Opcode Guide*.

Specifying Due Date Adjustments in a Billing Run

You use *billing run-time due date adjustments* to add days to the due dates of bills in a billing run. You can add the same number of days to all the bills in the billing run, or you can add different numbers of days to bills whose accounts are associated with different payment terms.

Billing run-time adjustments enable you to accommodate operational delays in your billing process. For example, bills associated with payment term A are due on the third Thursday of the month. On May 3, a problem in your system makes you unable to run billing. On May 10, you fix the problem and run billing for accounts whose billing DOM is 3. As a result, DOM 3 bills are generated a week late. To make up for this delay, you add a due date adjustment of 7 days for payment term A to the billing run. This gives DOM 3 customers associated with payment term A the usual time between receipt of their bill and its due date.

> **Note**
>
> To specify adjustments based on the payment term with which a bill is associated, you must first set up payment terms in your system and associate accounts with them. See "Loading Payment Terms".

To specify billing run-time due date adjustments:

1. Open the billing run configuration file (**pin_bill_run_control.xml**) in an XML editor or a text editor.

   By default, the file is in the *BRM_home***/apps/pin_billd** directory.

   > **Note**
   >
   > You also use this file to split your daily billing run into smaller billing runs. See "Configuring a Split Billing Run".

2. Specify the appropriate due date adjustments in the file. See "Editing the Billing Run Configuration File to Adjust Bill Due Dates ".
3. Save the file.

Note

If you want, you can save a copy of the file under a different name. The **pin_bill_day** script can take any XML file name as a parameter if the file's contents conform to the appropriate schema definition. See "Validating Your Billing Run Configuration File Edits".

4. *Manually* run the **pin_bill_day** script, using this syntax:

   ```
   pin_bill_day -file filename
   ```

   where *filename* is the name and location of the billing run configuration file. If you run the command in a different directory from where the file is located, you must also include the entire path for the file.

   In addition, *filename* must be in the same directory as the default *BRM_home***/apps/pin_billd/business_configuration.xsd** file.

Note

- The **-file** parameter when used with **pin_bill_day**, affects only the **pin_bill_accts** utility.
- Ensure that the accounts specified in the billing run configuration file reside on the same database schema where **pin_bill_day** is run. If the file contains accounts from different database schemas, **pin_bill_day** reports an error. See "Setting Up Billing in a Multischema Environment".
- When you run **pin_bill_day** with a configuration file, do not run it as a **cron** job. If you do, depending on the restrictions in configuration file, some bill units might never be billed.

Editing the Billing Run Configuration File to Adjust Bill Due Dates

By editing the *BRM_home***/apps/pin_billd/pin_bill_run_control.xml** file and then using it as a parameter for the **pin_bill_day** script, you can add days to the due dates of bills in a billing run. The due date adjustment can apply to all bills in the billing run or to bills associated only with specified payment terms.

To edit the file, open it in an XML editor or a text editor, and then perform one or both of these tasks:

- Associating Due Date Adjustments with Payment Terms
- Specifying a Default Due Date Adjustment

By default, the added days include weekends and nonbusiness days. To add *only business days*, change the following code in the PCM_OP_BILL_POL_CALC_PYMT_DUE_T policy opcode from this:

```
if (due_date_adjust) {
    fm_utils_add_n_days(due_date_adjust, &due_t);
}
```

To this:

```
if (due_date_adjust) {
    fm_bill_pol_add_n_bus_days(ctxp, n, "billing_calendar_name", &due_t, ebufp);
}
```

For information about the **fm_bill_pol_add_n_bus_days** function, see the chapter about billing in *BRM Opcode Guide*.

Associating Due Date Adjustments with Payment Terms

To specify an adjustment for bills associated with a particular payment term, add the appropriate **Payment_Term id** element to a **Due_date_adjustment** element in the billing run configuration file:

```
<Due_date_adjustment length = n >
    <Payment_Term id = x />
</Due_date_adjustment >
```

where:

- n is a positive integer that represents the number of days to add to the due date of a bill.
- x is the ID of any payment term defined in the **/config/payment_term** object. See "Managing Payment Terms".

You must specify both n and x. For example, the following 5-day adjustment applies only to bills associated with payment terms 1001 and 1002:

```
<Due_date_adjustment length = 5 >
    <Payment_Term id = 1001 />
    <Payment_Term id = 1002 />
</Due_date_adjustment >
```

Calculating Due Dates Based on Both Payment Terms and Adjustments

When due date adjustments are associated with payment terms, bill due dates are calculated as follows:

- **Payment terms that add days**

  billing cycle end date + *payment term* + *adjustment* = *due date*

  For example, if the billing cycle end date is April 1, 2001, the payment term 1001 is “add 7 days to the billing cycle end date," and the due date adjustment is 5 days, this calculation is used:

  April 1 + 7 days + 5 days = April 13

- **Payment terms that specify a particular day of month**

  payment term + *adjustment* = *due date*

  For example, if payment term 1002 is “second Tuesday of the month," the due date adjustment is 5 days, and billing is run on April 1, 2004, this calculation is used:

  April 8 (second Tuesday of April 2004) + 5 days = April 13

Specifying a Default Due Date Adjustment

To specify a due date adjustment for bills in a billing run that are not associated with a payment term or whose payment term is not associated with an adjustment, add a default adjustment:

```
<Due_date_adjustment length = n />
```

where n is a positive integer that represents the number of days to add to the due date of a bill.

For example, if the following adjustments are included in the same XML file, the 7-day adjustment applies to bills associated with any payment term except payment terms 1001 and 1002:

```
<Due_date_adjustment length = 5 >
    <Payment_Term id = 1001 />
    <Payment_Term id = 1002 />
</Due_date_adjustment >
<Due_date_adjustment length = 7 />
```

Sample Billing Run Configuration File

The following sample **pin_bill_run_control.xml** file contains these due date adjustments:

- A 5-day adjustment for bill units associated with payment terms 1001 and 1002
- A 7-day adjustment for all the other bill units in the billing run

Due date adjustments are configured in the **bold** elements.

```
<DOMList>
    <DOM>---03</DOM>
    <DOM>---07</DOM>
    <DOM>---15</DOM>
</DOMList>
<BillSegmentList>
    <BillSegment>101</BillSegment>
    <BillSegment>102</BillSegment>
</BillSegmentList>
<DueDateAdjustment Length=5 >
    <PaymentTerm id = 1001 />
    <PaymentTerm id = 1002 />
</DueDateAdjustment >
<DueDateAdjustment Length=7/>
```

For information about the **DOMList** and **BillSegmentList** parent elements, see "Including Only Specified Billing DOMs in Billing Runs" and "Including Only Specified Billing Segments in Billing Runs".

Validating Your Billing Run Configuration File Edits

After editing the XML file, you use the file name as a parameter when you run the **pin_bill_day** script. See "Specifying Due Date Adjustments in a Billing Run".

The script passes the file name to the pin_bill_accts utility, which validates the contents of the file against its schema definition. If the contents do not conform to the schema definition, the utility returns an error. The schema definition is in this file:

*BRM_home***/xsd/pin_bill_run_control.xsd**

The XML file is not directly linked to its schema definition file. Instead, it is linked to this XSD *reference* file:

*BRM_home***/apps/pin_billd/business_configuration.xsd**

15

About Bill Suppression

Learn how to implement bill suppression in Oracle Communications Billing and Revenue Management (BRM), which allows you to postpone finalizing a bill until the end of a future billing cycle.

Topics in this document:

About Suppressing Bills

BRM finalizes a bill at the end of each billing cycle. When a bill is finalized, the status of its bill items is changed from pending to open so that they stop accumulating charges and so that payments can be applied to them. In addition, a due date is added to the bill. A new bill is created to handle bill items for the next billing cycle.

Bill suppression, however, enables you to postpone finalizing a bill until the end of a *future* billing cycle. When a bill is suppressed, it is extended to include bill items for the next cycle, and the bill continues to accumulate charges until the end of that cycle.

Note

Charges accrued during all cycles in which a bill is suppressed do not age, get invoiced, go into collections, or have a due date set for them until suppression ends and the bill is finalized.

You use bill suppression to avoid sending out unnecessary bills and incurring wasteful expenses. For example, if the cost to create and mail a bill is greater than the balance due, you can suppress the bill until its balance due is greater than its production costs.

Bills can be suppressed in several ways. See these topics:

- About Automatic Bill Suppression
- About Manual Bill Suppression
- About Manual Account Suppression

All types of bill suppression can be overridden by exceptions. See "Exceptions to Bill Suppression".

About Automatic Bill Suppression

At the end of a billing cycle, BRM can automatically suppress bills whose balance is *less* than a user-specified minimum required to finalize a bill. Such bills are suppressed for one billing cycle. If their balance is still below the minimum at the end of that cycle, they are suppressed for another billing cycle.

Note

- Bills with negative balances are not suppressed.
- If both bill suppression and open item purging are enabled for a bill unit (**/billinfo** object), the bills for the bill unit are always suppressed because its bill total will always be **0**. For more information about open item purging, see "Enabling Open Items to Be Purged" in *BRM System Administrator's Guide*.

If the number of consecutive billing cycles for which a bill is suppressed reaches your specified maximum number of cycles, the bill is generated even if its balance is still below the minimum. This ensures that an excessive amount of time does not pass between customer bills.

To implement automatic bill suppression, you specify minimum balance amounts and maximum cycle limits for each customer segment that includes accounts whose bills you want to suppress automatically. A customer segment's specifications apply to all the bill units in the accounts that belong to the segment.

For example, a customer segment for low-usage accounts with bad payment histories might have a bill-generation threshold of only $5 and a limit of only three consecutively suppressed cycles, whereas a customer segment for high-usage accounts with good payment histories might have a bill-generation threshold of $15 and a limit of six consecutively suppressed cycles.

Note

If an account belongs to more than one customer segment, the *lowest* minimum balance and the *lowest* maximum cycle settings associated with the customer segments apply to the account. These settings can be from different customer segments.

For example, account X belongs to customer segments A and B. Segment A's minimum balance is $5 and its maximum cycle setting is 4. Segment B's minimum balance is $10 and its maximum cycle setting is 2. Thus, account X's minimum balance is $5 (from segment A) and its maximum cycle limit is 2 (from segment B).

For more information, see "Automatically Suppressing Bills".

About Manual Bill Suppression

Manual bill suppression enables you to suppress individual bills programmatically or through a custom user interface on a case-by-case basis.

For example, if you use automatic bill suppression, you can use manual bill suppression to suppress bills whose balance does not qualify for automatic suppression, as in this situation: Customer A's account belongs to customer segment X. The minimum balance required to finalize bills associated with accounts in customer segment X is $10. Midway through the current billing cycle, customer A's balance is $105, so his bill does not qualify for automatic bill suppression and will be finalized at the end of the billing cycle. Because customer A is having cash flow problems, however, he calls a customer service representative (CSR) and asks her to suppress his bill for two billing cycles. Using an interface that interacts with the manual bill suppression feature, she manually suppresses his bill for the requested number of cycles.

Note

Unlike automatic bill suppression, the default manual bill suppression feature does not use customer segments.

For more information, see the chapter about billing in *BRM Opcode Guide*.

About Manual Account Suppression

Manual *account* suppression enables you to suppress accounts on request. With this feature, customers who will not be using their account for an extended period of time can retain all their services and connection IDs without accumulating any of the charges usually associated with their account.

Optionally, charges associated with one account-level bundle can accumulate during account suppression. You can use this bundle to handle any special fees you want to charge while an account is suppressed. For example:

- Charge a purchase fee for suppressing an account.
- Charge a low monthly cycle fee for retaining a suppressed account's services and connection IDs.

Note

Unlike automatic bill suppression, account suppression does not use customer segments.

To prevent owner accounts from being suppressed, add the appropriate logic to the PCM_OP_BILL_POL_CHECK_SUPPRESSION policy opcode. For more information, see the chapter about billing in *BRM Opcode Guide*.

Suppressed Accounts versus Inactive Accounts

A suppressed account differs from an inactive account as shown in Table 15-1:

Table 15-1 Differences between Suppressed and Inactive Accounts

Condition	Account Type	Status
Is the account inactive?	**Suppressed account**	**No.**
Is the account inactive?	Inactive account	Yes.
Are the account's services inactive?	**Suppressed account**	**Yes.**
Are the account's services inactive?	Inactive account	Yes.
Are the account's bills finalized?	**Suppressed account**	**No.** Accrued charges do not age, get invoiced, or go into collections.

Table 15-1 (Cont.) Differences between Suppressed and Inactive Accounts

Condition	Account Type	Status
Are the account's bills finalized?	Inactive account	Yes. Charges accrued before the account is inactivated age, get invoiced, and go into collections.
Can new charges accrue in the account?	**Suppressed account**	**Yes.** Optionally, charges associated with one account-level bundle can accrue.
Can new charges accrue in the account?	Inactive account	No.
Does the status of the account's child accounts change?	**Suppressed account**	**No.** Bills for any nonpaying child bill units of the parent account's paying parent bill units are finalized. Their charges continue to accrue in the suppressed parent account's bill that is not yet finalized.
Does the status of the account's child accounts change?	Inactive account	**Yes.** All child accounts that have nonpaying bill units are inactivated.
Is charge sharing affected?	Inactive account	**Yes.** Charge sharing is suspended. Formerly sponsored charges accrue in member account bills while the owner account is inactive.
Is discount sharing affected?	Inactive account	**Yes.** Member account events impact member accounts' balances, not the inactive owner account's balance.
Is charge sharing affected?	**Suppressed account**	**No.** Member accounts' sponsored charges continue to accrue in the suppressed owner account's unfinalized bill.
Is discount sharing affected?	**Suppressed account**	**No.** Member account events continue to impact the balance of the suppressed discount sharing group owner's unfinalized bill.

Exceptions to Bill Suppression

All types of bill suppression can be overridden by exceptions. The default exceptions are listed in Table 15-2:

Table 15-2 Exceptions to Bill Suppression

Exception	Description
Payment received	The receipt of a payment requires that a bill be finalized to record the payment against the bill. By default, this exception is disabled. To enable it, see the chapter about billing in *BRM Opcode Guide*.
Adjustment or credit applied	If an adjustment or a credit is made to an account, the bill is finalized to notify the customer about the change in the balance.

Table 15-2 (Cont.) Exceptions to Bill Suppression

Exception	Description
Maximum consecutive cycle suppressions exceeded	The maximum number of consecutive billing cycles for which a bill can be suppressed is specified for the customer segment to which the bill's account belongs. See "Associating Bill Suppression Information with Customer Segments". **Note** • If the maximum number of consecutive cycles for the customer segment is 0, the bill can *never* be suppressed. • If the maximum number of consecutive cycles for the customer segment is missing, the bill is not suppressed. It is considered as zero cycles.
First or last bill	An account's first and last bills are always finalized at the end of their billing cycles, even if their balance is below the minimum balance required to finalize bills.
Account closed	Bills associated with closed accounts cannot be suppressed.

To add or delete exceptions, see the chapter about billing in *BRM Opcode Guide*.

Note

The **Bill Now** functionality and on-purchase billing for bundles and packages do not override bill suppression. Although they finalize a suppressed bill, they do *not*:

- Reset the counter in the **/billinfo** object that tracks consecutively suppressed billing cycles (PIN_FLD_NUM_SUPPRESSED_CYCLES).
- End manual bill or manual account suppression.

How Exceptions Affect Manual Bill and Account Suppression

Exceptions do not end manual bill or manual account suppression. Rather, they cause the PCM_OP_BILL_MAKE_BILL opcode to do the following:

- Finalize the manually suppressed bill at the end of its current billing cycle.
- Reset the counter in the **/billinfo** object that tracks the bill's *consecutively suppressed* billing cycles (PIN_FLD_NUM_SUPPRESSED_CYCLES) to **0**.
- Decrement the counter in the **/billinfo** object that tracks the bill's *remaining manually suppressed* cycles (PIN_FLD_SUPPRESSION_CYCLES_LEFT) by 1.

For example, a bill is manually suppressed for 10 billing cycles. At the end of the fifth cycle, however, it is finalized because of an exception. At that time, the bill's *consecutively suppressed* cycles counter is reset to **0**, and its *remaining manually suppressed* cycles counter is decremented by 1. Because the latter counter was also decremented by 1 at the end of the four previous suppressed billing cycles, its value is now 5, which indicates that the bill should be suppressed for 5 more cycles.

Automatically Suppressing Bills

To implement manual bill suppression and manual account suppression, see *BRM Opcode Guide*.

To set up automatic bill suppression:

1. Set up customer segments in your system and add accounts to them. See "Creating and Managing Customer Segments" in *BRM Managing Customers*.

   Note

   Whether or not you set up customer segments, all accounts belong to customer segment 0. Thus, to implement automatic bill suppression *without* creating customer segments, perform step 2 for customer segment 0. The suppression specifications associated with customer segment 0 apply to *all* the accounts in your system.

2. For each customer segment that includes accounts whose bills you want to suppress automatically, specify the following information:
   - Minimum balance required for a bill to be finalized.
   - Maximum number of consecutive billing cycles that a bill can be suppressed.

   See "Associating Bill Suppression Information with Customer Segments".

Editing the Bill Suppression Configuration File

In the *BRM_home***/sys/data/config/pin_bill_suppression.xml** file, you specify the following information for each customer segment that includes accounts whose bills you want to suppress automatically:

- Minimum balance required for a bill to be finalized.
- Maximum number of consecutive billing cycles that a bill can be suppressed.

To edit this configuration file, open it in an XML editor or a text editor.

In the file, the **CustomerSegmentArray** parent element must contain a **CustomerSegment** child element for each customer segment to which you want to add bill suppression information. A **CustomerSegment** child element looks like this:

```
<CustomerSegment ID="int">
    <MinBillAmount>decimal</MinBillAmount>
    <MaxSuppressionCycles>int</MaxSuppressionCycles>
</CustomerSegment>
```

To specify bill suppression information for a customer segment, add a **CustomerSegment** child element to the **CustomerSegmentArray** parent element. In the child element, specify values for the items listed in Table 15-3:

Table 15-3 Customer Segment Bill Suppression Elements

XML Element or Attribute	Description	Possible Values
ID	The ID of the customer segment. Customer segments are defined in an array in the **/config/customer_segment** object. The index of each array entry is the ID of a customer segment. The IDs of the customer segments to which an account belongs are specified in the **PIN_FLD_CUSTOMER_SEGMENT_LIST** field of the **/account** object.	Any nonnegative integer. **Note:** • Suppression data associated with nonexistent customer segment IDs is ignored until the IDs are defined in the **/config/customer_segment** object. • Customer segment ID 0 is the default customer segment. All accounts belong to this customer segment.
MinBillAmount	Minimum balance required to finalize a bill. If the balance is less than this amount, the bill is automatically suppressed.	Any positive number with two decimal places. **Note:** Although balances are stored in account currency, this value is not converted to a particular currency. For example, if this value is **5.00**, it represents 5 US dollars, 5 Australian dollars, 5 euros, and so on.
MaxSuppressionCycles	Maximum number of consecutive billing cycles for which a bill can be suppressed.	Any integer greater than 0.

Sample Bill Suppression Configuration File

The following is a sample **pin_bill_suppression.xml** file:

```
<?xml version="1.0" encoding="UTF-8"?>

<BusinessConfiguration
   xmlns="http://www.portal.com/schemas/BusinessConfig"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.portal.com/schemas/BusinessConfig
business_configuration.xsd">

<!-- Sample file. Modify according to guidelines -->
   <BillSuppressionConfiguration>
      <CustomerSegmentList>
         <CustomerSegment ID="1001">
            <!-- Bad customer -->
            <MinBillAmount>5.55</MinBillAmount>
            <MaxSuppressionCycles>2</MaxSuppressionCycles>
         </CustomerSegment>
         <CustomerSegment ID="1002">
            <!-- Good customer -->
            <MinBillAmount>99.99</MinBillAmount>
            <MaxSuppressionCycles>5</MaxSuppressionCycles>
         </CustomerSegment>
      </CustomerSegmentList>
   </BillSuppressionConfiguration>
```

```
</BusinessConfiguration>
```

Validating Your Bill Suppression Configuration File Edits

After editing the contents of the XML file, you use the "load_pin_bill_suppression" utility to load the contents of the file into the **/config/suppression** object in the database. See "Associating Bill Suppression Information with Customer Segments".

Before loading the contents of the file, the utility validates the contents against the file's schema definition. If the contents do not conform to the schema definition, the load operation fails. The schema definition is in this file:

*BRM_home***/xsd/pin_bill_suppression.xsd**

The XML file is not directly linked to its schema definition file. Instead, it is linked to the *BRM_home***/sys/data/config/business_configuration.xsd** reference file.

Associating Bill Suppression Information with Customer Segments

To implement bill suppression, edit the bill suppression configuration file (**pin_bill_suppression.xml**) and then load its contents into the **/config/suppression** object in the BRM database.

1. Open the **pin_bill_suppression.xml** file in an XML editor or a text editor.

   By default, the file is in the *BRM_home***/sys/data/config** directory, where *BRM_home* is the directory where you installed BRM components.

2. In the file, specify the following information for each customer segment that contains accounts whose bills you want to suppress:

   - Minimum balance required for a bill to be finalized.
   - Maximum number of consecutive billing cycles that a bill can be suppressed.

   See "Editing the Bill Suppression Configuration File".

3. Save the file.

4. Use the following command to run the "load_pin_bill_suppression" utility from the directory in which the **pin_bill_suppression.xml** file is located:

   ```
   load_pin_bill_suppression pin_bill_suppression.xml
   ```

   > **Note**
   >
   > When you run the utility, the **pin_bill_suppression.xml** and **business_configuration.xsd** files must be in the same directory. By default, both files are in *BRM_home***/sys/data/config**. See "Validating Your Bill Suppression Configuration File Edits".

   If you do not run the utility from the directory in which **pin_bill_suppression.xml** is located, include the complete path to the file, for example:

   ```
   load_pin_bill_suppression BRM_home/sys/data/config/pin_bill_suppression.xml
   ```

For more information, see "load_pin_bill_suppression".

5. Stop and restart the Connection Manager (CM).
6. To verify that the bill suppression information was loaded, display the **/config/suppression** object by using Object Browser or the **robj** command with the **testnap** utility.

   For general instructions on using **testnap**, see "Using the testnap Utility to Test BRM" in *BRM Developer's Guide*. For information about reading an object and writing its contents to a file, see "Reading an Object and Writing Its Contents to a File" in *BRM Developer's Guide*.

16

Creating Custom Bill Items

Learn how to create custom bill items and how Oracle Communications Billing and Revenue Management (BRM) assigns custom bill items to events.

Topics in this document:

About Custom Bill Items

Bill items enable you to track a customer's balance for a type of event. For example, a bill item tracks all charges for service usage or all charges for cycle fees during a billing cycle.

By default, BRM tracks balances in the following bill items: cycle arrears items, cycle forward items, cycle forward arrears items, cycle tax items, cycle incentive items, and usage items.

You can create custom bill items to further aggregate charges and to provide more descriptive information in your invoices, reports, and CSR applications. For example, if you charge customers for password changes, you can track password changes separately and list the charges on invoices under “password change" rather than “usage."

About Defining Custom Bill Items

When you create a custom bill item, you define the following:

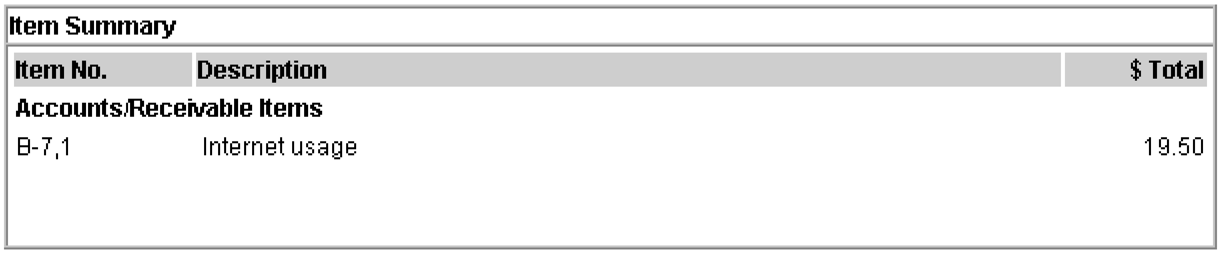
- **The bill item name**. This is the item name displayed on customer invoices, reports, and CSR applications.
- **How to track charges.** You specify whether a bill item stores one charge only or accumulates multiple charges. See "Tracking Charges in Bill Items".
- **How to store the item in the database.** You can either pre-create a custom **/item** object in the database or have BRM create one for you. See "About Creating /item Objects".
- **The type of events you want the bill item to track**. You do this by assigning bill items either to an event and service combination or to event attributes. See "About Assigning Custom Bill Items to Events".

Tracking Charges in Bill Items

When you create a custom bill item, you specify whether the item accumulates charges or tracks each charge separately.

Cumulative bill items accumulate charges throughout the billing cycle. All events of the same type are consolidated into a single **/item** object. For example, if a customer has three broadband sessions during a billing cycle, BRM stores all of the charges in one **/item** object. The customer's invoice also lists one item with the total rolled-up charge for all three sessions, as shown in Figure 16-1:

Figure 16-1 Rolled-Up Broadband Usage Charges on Invoice

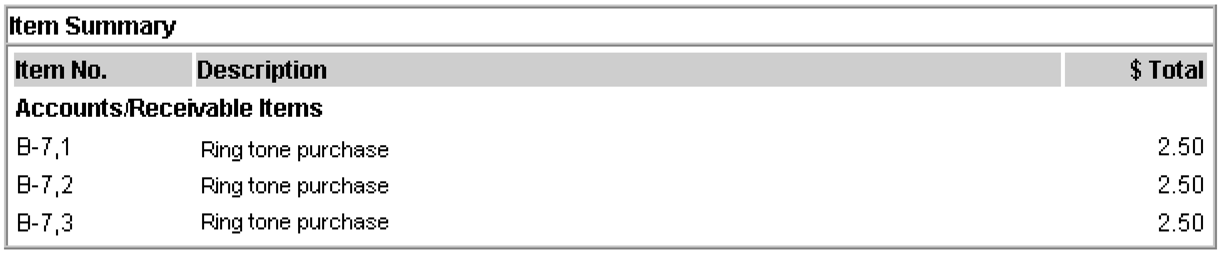


Item Summary

Item No.	Description	$ Total
Accounts/Receivable Items		
B-7,1	Internet usage	19.50

Individual bill items store a charge for a single event, such as purchasing a charge offer. A separate **/item** object is created for each event of the same type. For example, if a customer purchases three ringtones during a billing cycle, BRM stores the charges in three separate **/item** objects. The customer's invoice also lists three separate bill items and their charges, as shown in Figure 16-2:

Figure 16-2 Individual Bill Items



Item Summary

Item No.	Description	$ Total
Accounts/Receivable Items		
B-7,1	Ring tone purchase	2.50
B-7,2	Ring tone purchase	2.50
B-7,3	Ring tone purchase	2.50

You specify whether a custom bill item is cumulative or individual in the **config_item_types.xml** file. See "Mapping Item Tags to Item Types".

About Creating /item Objects

You create your custom bill items in the database by subclassing the **/item** object. For example, you can create an **/item/password** object for storing password charges.

For usage events, you can specify whether BRM pre-creates the custom bill item before any event occurs or creates it during the rating process. In this case, the item is created when a service object is created for an account and when billing is run.

About Assigning Custom Bill Items to Events

You assign custom bill items to events in either of two ways:

- Assign bill items to a specific event and service combination. See "About Using Event and Service Combinations to Assign Bill Items".
- Assign bill items to events based on event attributes. See "About Using Event Attributes to Assign Bill Items".

About Using Event and Service Combinations to Assign Bill Items

You can assign a bill item to each event and service combination that you support. For example, you can map the event and service combinations to a bill item object as shown in Table 16-1:

Table 16-1 Event and Service Combinations for Bill Item Object

Event and Service Combination	Item Object
/event/session/telco/gsm **/service/telco/gsm/***	**/item/gsm_usage**
/event/session/telco/gprs **/service/telco/gprs/***	**/item/gprs_usage**
/event/session/email **/service/email**	**/item/email_usage**

In Table 16-1, BRM separates charges for GSM usage, GPRS usage, and email usage into three **/item** objects and displays each item separately on customer invoices.

You can assign one bill item to one event and service combination or assign one bill item to multiple event and service combinations. For example, you can assign **/item/voice** to any event and service combination that provides voice service.

You can also create multiple item configurations (sets of item-to-event-and-service mappings) to apply to different types of bill units. This enables you to avoid creating unnecessary bill items in your BRM system. See "Improving Performance by Using Multiple Item Configurations" in *BRM System Administrator's Guide*.

Note

A large number of items per account or bill unit (**/billinfo** object) can decrease system performance. Additionally, account creation and billing failures can occur when there are a large number of item types for an account or service that results in the maximum lengths for PIN_FLD_ITEM_POID_LIST and PIN_FLD_NEXT_ITEM_POID_LIST fields to be exceeded. Item POIDs are appended to PIN_FLD_ITEM_POID_LIST and PIN_FLD_NEXT_ITEM_POID_LIST in the **/account** object when a new item type is created for an account or service. The following options are recommended when creating custom item types:

- Limit the number of item types such that if a customer uses all the event and service combinations defined in the **config_item_tags.xml** and **config_item_types.xml**, the number of item poids for an account or service does not exceed 2000 bytes. See "Assigning Item Tags Based on Event and Service Combinations".
- Create item types with PRECREATE set to FALSE. By setting PRECREATE to FALSE, the items are created only when the particular event occurs for the service. This enables minimal items to be created during account creation or billing. See "Mapping Item Tags to Item Types".

To assign event and service combinations to bill items, you perform these tasks:

- Map event and service combinations to item tags by editing the **config_item_tags.xml** file. See "Assigning Item Tags Based on Event and Service Combinations".
- Map item tags to item types by editing the **config_item_types.xml** file. See "Mapping Item Tags to Item Types".

You can configure how BRM assigns event and service combinations to bill items by adding the **– fm_act attach_item_to_event** *n* entry in the CM pin.conf file, where *n* can be **0**, **1**, or **2**. If the value of *n* is:

- **0**: BRM does not assign events without balance impact to any item. The events that have balance impact are assigned to items according to the event and service combinations defined in the appropriate **/config/item_tags** object. This is the default value.
- **1**: BRM assigns any event to items. This includes events without any balance impact.
- **2**: BRM assigns any event to items. If the event has a service and the event and service combination is not defined in a **/config/item_tags** object, BRM assigns the event to the default item (**/item/misc**) on the account and not on service.

To add the **attach_item_to_event** entry:

1. Open the Connection Manager (CM) configuration file (*BRM_home***/sys/cm/pin.conf**, where *BRM_home* is the directory in which you installed BRM components).
2. Add the following entry:

   ```
   - fm_act attach_item_to_event n
   ```

   where *n* can be **0**, **1**, or **2**.
3. Save and close the file.
4. Stop and restart the CM.

For information about configuration files, see "Using Configuration Files" in *BRM System Administrator's Guide*.

About Using Event Attributes to Assign Bill Items

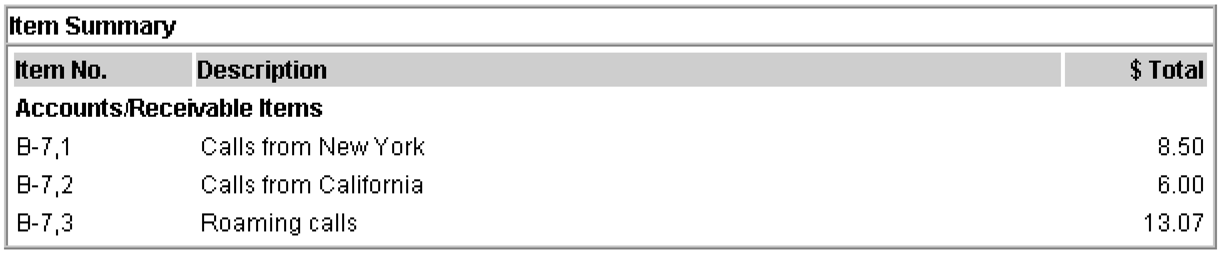
You can provide even more granularity in your reports, invoices, and CSR applications by assigning items by event attribute. This enables you to assign multiple items to the same event and service combination.

For example, for the event and service combination of **/event/session/telco/gsm** and **/service/telco/gsm/***, you can separate events by the following:

- Calls that originated in New York in a custom item object named **/item/new_york**.
- Calls that originated in California in a custom item object named **/item/california**.

In the example shown in Figure 16-3, the customer's invoice displays an itemized list of GSM usage:

Figure 16-3 Itemized List of GSM Usage



Item Summary		
Item No.	**Description**	**$ Total**
Accounts/Receivable Items		
B-7,1	Calls from New York	8.50
B-7,2	Calls from California	6.00
B-7,3	Roaming calls	13.07

To assign items based on event attributes, you perform the following tasks:

- Map event attributes to an item tag by customizing a policy opcode. See "Assigning Item Tags Based on Event Attributes".
- Map item tags to item types by editing the **config_item_types.xml** file. See "Mapping Item Tags to Item Types".

How BRM Assigns Custom Bill Items to Events

BRM assigns bill items to events during the rating process by performing the following tasks:

1. BRM assigns an *item tag* based on the event and service combination.
2. If customized to do so, the BRM custom API takes the assigned item tag as input and then assigns a different item tag based on the event's attributes.
3. BRM assigns an *item type* based on the item tag.

Cumulative Custom Item for Taxes

The following example shows a custom item that stores all taxes from all tax suppliers in a single bill item for each billing cycle.

1. Open the *BRM_home***/sys/data/pricing/example/config_item_tags.xml** file.
2. Add the following entry:

```
<ItemTagElement>
<ItemTag>cycle_tax</ItemTag> <EventType>/event/billing/cycle/tax</EventType>
<ServiceType>/account</ServiceType> </ItemTagElement>
```

3. Save and close the file.
4. Open the *BRM_home***/sys/data/pricing/example/config_item_types.xml** file.
5. Add the following entry:

   ```
   <ItemTypeElement>
   <ItemTag>cycle_tax</ItemTag>
   <ItemDescription>Cycle Tax</ItemDescription>
   <ItemType precreate="false" type="cumulative">/item/cycle_ tax</ItemType>
   </ItemTypeElement>
   ```

6. Save and close the file.

Setting Up BRM to Assign Custom Bill Items to Events

To assign items to events and to update the items in the BRM database for billing and tracking:

1. Create the custom bill item in the database by subclassing the **/item** storable class. See "Creating Custom Fields and Storable Classes" in *BRM Developer's Guide*.

   Note

   Before creating a bill item, you must know the event and service type whose balance impacts you want stored with the new item.

2. Map event and service combinations to item tags in the appropriate item configuration. See "Assigning Item Tags Based on Event and Service Combinations".
3. Map event attributes to item tags. See "Assigning Item Tags Based on Event Attributes".
4. Map item tags to item types in the appropriate item configuration. See "Mapping Item Tags to Item Types".

Assigning Item Tags Based on Event and Service Combinations

You map event and service combinations to custom item tags by editing the **config_item_tags.xml** file. You then load the item tags into a **/config/item_tags** object by using the **load_config_item_tags** utility. See "load_config_item_tags".

Every item tag in the item tags file must have a corresponding item type defined in the **config_item_types.xml** file. If you make changes to the **config_item_tags.xml** file after you load it into the database, you must make corresponding changes to the item types and load the **config_item_types.xml** file again. See "Mapping Item Tags to Item Types".

To assign item tags:

1. Open the *BRM_home***/sys/data/pricing/example/config_item_tags.xml** file in a text editor.
2. Verify that the value of the following optional tag is correct:

   ```
   <Name>Item_Configuration_Name</Name>
   ```

   Where *Item_Configuration_Name* is the name of the item configuration to which the item tags defined in this XML file belong.

The name of the BRM system's default item configuration is **Default**. If the value of this tag is **Default**, or if this tag is not included in the file, your custom tags are added to the default item configuration.

To add a new item configuration to your system, enter a unique item configuration name in this XML file and in the corresponding **config_item_types.xml** file.

To modify an existing item configuration in your system, enter that configuration's name in this XML file and in the corresponding **config_item_types.xml** file.

For more information about item configurations, see "Improving Performance by Using Multiple Item Configurations" in *BRM System Administrator's Guide*.

3. Add custom tags to the file by following the instructions in the file.

> **Note**
>
> Tag names must be unique.

For example, to store GSM calls in a custom bill item, add the following entry:

```
<ItemTagElement>
    <ItemTag>GSM</ItemTag>
    <EventType>/event/delayed/session/telco/GSM/*</EventType>
    <ServiceType>/service/telco/GSM/*</ServiceType>
</ItemTagElement>
```

where:

- The **ItemTag** element specifies the unique name for the item tag.
- The **EventType** element specifies the parent **/event** storable class.
- The **ServiceType** element specifies the parent **/service** storable class.

4. Save the file. You can save the file with a different name and location or use the original file.

5. Run the following command:

```
load_config_item_tags config_item_tags_file
```

where *config_item_tags_file* is the name and path of your **config_item_tags.xml** file.

> **Note**
>
> The **load_config_item_tags** utility replaces the entire contents of the **/config/item_tags** object whose PIN_FLD_NAME value matches the Name value in the **config_item_tags.xml** file with the contents of that file. If you are updating a set of item tags, you cannot load new items only. You must load complete sets of items each time you run the **load_config_item_tags** utility.

6. Stop and restart CM. See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

To verify that the item tags were loaded, use Object Browser in Developer Center or the **testnap** utility's **robj** command to display the specified **/config/item_tags** object. See "Reading an Object and Writing Its Contents to a File" in *BRM Developer's Guide*.

Assigning Item Tags Based on Event Attributes

You assign item tags based on event attributes by customizing the BRM API.

Setting Up Online Charging to Assign Items Based on Event Attributes

You set up online charging to assign items to events based on event attributes by using the PCM_OP_BILL_POL_GET_ITEM_TAG policy opcode. By default, this policy opcode does nothing. However, you can customize it to find events with specific flist fields, assign the appropriate item tag, and then return the item tag in the PIN_FLD_ITEM_TAG output flist field. See *BRM Opcode Guide*.

Mapping Item Tags to Item Types

You map the *item tags* that you defined in the **config_item_tags.xml** file or policy opcode to *item types* by using the **config_item_types.xml** file. You then load the mappings into the appropriate **/config/item_types** object by using the **load_config_item_types** utility.

Note

- The **load_config_item_types** utility replaces the entire contents of the specified **/config/item_types** object whose PIN_FLD_NAME value matches the Name value in the **config_item_types.xml** file with the contents of that file. If you are updating a set of mappings, you cannot load new mappings only. You must load complete sets of mappings each time you run the **load_config_item_types** utility.
- If the **config_item_types.xml** file does not contain a PIN_FLD_NAME value, the **/config/item_types** object whose PIN_FLD_NAME value is **Default** is replaced.

To map item tags to item types:

1. Open the *BRM_home***/sys/data/pricing/example/config_item_types.xml** file in an XML editor or a text editor.
2. Verify that the value of the following tag is correct:

   ```
   <Name>Item_Configuration_Name</Name>
   ```

   where *Item_Configuration_Name* is the name of the item configuration to which the item types defined in this XML file belong.

   The name of the BRM system's default item configuration is **Default**. If the value of this tag is **Default**, or if this tag is not included in the file, your custom mappings are added to the default item configuration.

   To add a new item configuration to your system, enter a unique item configuration name in this XML file and in the corresponding **config_item_tags.xml** file.

   To modify an existing item configuration in your system, enter that configuration's name in this XML file and in the corresponding **config_item_tags.xml** file.

   For more information about item configurations, see "Improving Performance by Using Multiple Item Configurations" in *BRM System Administrator's Guide*.

3. Map the item tags you created to custom item types by following the instructions in the file.

   For example, to map the item tag **new_york** to the item type **/item/new_york**, add the following entry:

```
<ItemTypeElement>
    <ItemTag>new_york</ItemTag>
    <ItemDescription>Calls from New York</ItemDescription>
    <ItemType precreate="false" type="cumulative">/item/new_york</
ItemType>
</ItemTypeElement>
```

   where:

   - The **ItemTag** element specifies the name of the item tag.
   - The **ItemDescription** element specifies the item name that is displayed in customer invoices, reports, and CSR applications.
   - The **precreate** element specifies whether BRM pre-creates the item for the service type: **true** specifies to pre-create the item in the database, and **false** specifies to create the item when the event occurs.

     Note

     BRM pre-creates items for usage events only. It does not pre-create items for purchase or cycle fee events.

   - The **type** element specifies whether to track balances separately or to consolidate balances: **cumulative** specifies that this bill item stores charges for all events of the same type in a billing cycle; **individual** specifies to create a separate item for each event. See "Tracking Charges in Bill Items" for more information.

     Note

     If you have set **precreate** to **false** for the usage or delayed events that are rated using Pipeline Manager, do not set the **type** to **individual**.

   - The **ItemType** element specifies the name of the custom **/item** object. In this example, BRM stores the item charges in the **/item/new_york** object.

4. Save the file. You can save the file with a different name and location or use the original file.
5. Run the following command:

```
load_config_item_types config_item_types_file
```

   where *config_item_types_file* is the name and path of your **config_item_types.xml** file.

> **Caution**
>
> The **load_config_item_types** utility replaces the entire contents of the **/config/item_types** object whose PIN_FLD_NAME value matches the Name value in the **config_item_types.xml** file with the contents of that file. If you are updating a set of mappings, you cannot load new item types only. You must load complete sets of mappings each time you run the **load_config_item_types** utility.

6. Stop and restart the CM. See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

To verify that the item tags were loaded, use Object Browser in Developer Center or the **testnap** utility's **robj** command to display the specified **/config/item_types** object. See "Reading an Object and Writing Its Contents to a File" in *BRM Developer's Guide*.

Assigning Bill Items to Event Balance Impacts

All events contain, or can contain, a balance impact. You can use custom **/item** types to separately track charges in individual balance impacts of an event. For example, even though only one BRM event is recorded for a service, if you charge for both connection time and the amount of bytes transferred during a session, the two charges can be tracked separately.

PIN_FLD_ITEM_TAGS is an array in the output flist of the PCM_OP_BILL_POL_GET_ITEM_TAG policy opcode. The PIN_FLD_ITEM_TAGS enables you to create an item tag for one or more balance impacts. You choose from an event, which balance impacts to use, and what item tags they are assigned to. The item type is assigned using the matching element ID of the item tag and balance impact.

The following example, lists the account POID and custom item tags in flist array format. Balance impacts with element IDs **2** and **3** have item types assigned based on the item tags **TransferVolume** and **ConnectionPeriod**. All other balance impacts have an item type assigned based on the item tag **SessionUsage**.

```
0     PIN_FLD_POID                POID [0] 0.0.0.1 /account 182477 0
0     PIN_FLD_ITEM_TAG             STR [0] SessionUsage
0     PIN_FLD_ITEM_TAGS          ARRAY [2] allocated 20, used 3
1     PIN_FLD_ITEM_TAG             STR [0] TransferVolume
0     PIN_FLD_ITEM_TAGS          ARRAY [3] allocated 20, used 3
1     PIN_FLD_ITEM_TAG             STR [0] ConnectionPeriod
```

> **Note**
>
> If a PIN_FLD_ITEM_TAGS array element is not specified for a balance impact, the balance impact will have an item assigned based on the PIN_FLD_ITEM_TAG element at the top level of the policy opcode output.

To assign bill items to event balance impacts:

1. Create your custom balance impact item tags and types. See "Setting Up BRM to Assign Custom Bill Items to Events".
2. Map your custom balance impact item tags to your custom balance impact item types. See "Mapping Item Tags to Item Types".

3. Customize the PCM_OP_BILL_POL_GET_ITEM_TAG policy opcode with the appropriate business logic to:
   a. Determine the element ID of the required balance impact of the event in the input flist of the PCM_OP_BILL_POL_GET_ITEM_TAG policy opcode.
   b. Create a PIN_FLD_ITEM_TAGS array element in the output flist of the PCM_OP_BILL_POL_GET_ITEM_TAG policy opcode with the element ID being the same as the balance impact ID from step a.
   c. Set the field PIN_FLD_ITEM_TAGS of the PIN_FLD_ITEM_TAGS array element to the custom value of the item tag that you require for the balance impact.

      For information about customizing policy opcodes, see "Using the Policy Opcode Source Files" in *BRM Developer's Guide*.
4. Assign your custom items to event balance impacts during the rating process.

By default, any item tag specified for a sharing group owner's balance impact is ignored and an item of type **/item/sponsor** is used. A custom item type can be assigned for a sharing group owner's balance impact only in the following situations:

- The **SplitSponsorItemByMember** business parameter is enabled. See "Splitting Sponsored Charges into Multiple Items".
- A custom item tag is specified for the sharing group owner's balance impact in the PIN_FLD_ITEM_TAGS array.

Creating Custom Sponsored Bill Items

By default, BRM accumulates the charges for all charge sharing member services and accounts belonging to one owner in a single **/item** object.

You can create custom sponsored bill items that divide the accumulated charges across the members of the charge sharing group.

To create custom sponsored bill items:

1. Enable the **SplitSponsorItemByMember** business parameter. See "Splitting Sponsored Charges into Multiple Items".

   The charges are broken down into:
   - One **/item/sponsor** object for each charge sharing member service instance.
   - One **/item/sponsor** object for account-level charges for all member accounts.

   The sponsored items point to the owner account object and to the sharing member service. If the shared charges are at the member account level, the service pointer is NULL.
2. Create your custom balance impact **/item/sponsor** tags and types. See "Setting Up BRM to Assign Custom Bill Items to Events" and "Assigning Bill Items to Event Balance Impacts".

> **Note**
>
> When configuring a custom item type for sharing group owner's balance impacts, specify the base type without the component **sponsor** in the item type string. For example, to use **peak_usage** for sponsored peak usage charges, configure the tag as **/item/peak_usage**. BRM automatically uses the correct sponsor subtype **/item/sponsor/peak_usage** at the time of rating.

3. Assign the custom sponsored items to event balance impacts during the rating process.

Splitting Sponsored Charges into Multiple Items

By default, splitting sponsored charges into multiple sponsored items is disabled. You can enable splitting sponsored charges for online charging, offline charging, or both.

If you split sponsored charges for offline charging, you do not have the option of disabling the pre-updater step of the Rated Event (RE) Loader. This is because the pre-updater stored procedure assigns sponsored items to events when the splitting option is enabled.

To enable this feature, run the **pin_bus_params** utility to change the **SplitSponsorItemByMember** business parameter. For information about this utility, see *BRM Developer's Guide*.

To enable splitting sponsored charges into multiple items:

1. Go to *BRM_home***/sys/data/config**.
2. Create an XML file from the **/config/business_params** object:

   ```
   pin_bus_params -r SplitSponsorItemByMember -bus_params_billing.xml
   ```

3. In the XML file, change **disabled** to **enabled**:

   ```
   <SplitSponsorItemByMember>disabled</SplitSponsorItemByMember>
   ```

4. Do one of the following:
   - To enable splitting of sponsored charges for both online charging and offline charging, change **disabled** to **enabled**.
   - To enable splitting of sponsored charges for only online charging, change **disabled** to **onlyRealTime**.
   - To enable splitting of sponsored charges for only offline charging, change **disabled** to **onlyBatch**.

     ```
     <SplitSponsorItemByMember>disabled</SplitSponsorItemByMember>
     ```

5. Save the file as **bus_params_billing.xml**.
6. Load the XML file into the BRM database:

   ```
   pin_bus_paramsbus_params_billing.xml
   ```

7. Stop and restart the CM.

17
Creating Corrective Bills

Learn how to update or correct a bill after it has been sent to a customer in Oracle Communications Billing and Revenue Management (BRM).

Topics in this document:

About Corrective Bills

Correcting a bill is necessary when a bill or invoice sent to a customer requires an update. Corrections can include:

- Corrections to a bill resulting from changes that do not affect the amount owed by the customer. For example, a change to the invoice address on the customer's account or a correction to the language on the customer's profile.
- Corrections to the charges on a bill resulting from adjustments, for example, settled disputes that require some item adjustment, catalog corrections such as retroactive tariff updates, and so on.

A *corrective bill* is a bill that is generated after corrections are made to a regular bill or a bill that was previously corrected. You ensure that the adjustments are appropriately allocated and then generate the corrective bill. When a corrective bill is generated, it is assigned a new due date based on the current payment setup for the account.

You can create a series of corrective bills for a bill. For example, you can generate a corrective bill for a regular bill, and then a corrective bill for the first corrective bill, and so on.

A corrected invoice is the invoice associated with a corrective bill and, based on your billing configuration, contains the entire bill information or only the corrections and adjustments. For information on corrected invoices, see "Corrective Invoicing" in *BRM Designing and Generating Invoices*.

You use Billing Care or Customer Center to generate corrective bills for an individual bill. When you have corrections for a set of bills or accounts, you use the **pin_make_corrective_bills** utility to create the corrective bills. See "Billing Accounts By Using the pin_make_corrective_bill Utility".

Note

You cannot use the BRM billing scripts to run corrective billing.

BRM can generate corrective bills for the following:

- Original bills.
- Bills created by using Bill Now
- On-purchase billing.
- Parent and nonpaying child bills in a bill unit (**/billinfo** object) hierarchy.
- A prior corrective bill. That is, you can correct a subsequent corrective bill to address the errors on a prior corrective bill.

After BRM generates a corrective bill for a prior bill, BRM accepts payments it receives for the regular bill, the corrective bill, or both, but it applies the payments to the latest (original or corrective) bill only. For example, BRM has a regular bill B1-124 for which it generates two corrective bills (CB1-28 and CB1-30) in one bill period. For BRM, CB1-30 is the *last* bill and B1-124 and CB1-28 are *prior* bills. By default, all payments that BRM receives for B1-124 or CB1-28 or CB1-30 will be applied against CB1-30.

When configuring how to correct bills, note how corrective billing works with the following features:

- **In-progress bills.** BRM creates a corrective bill for a finalized bill only. If a bill is currently in progress, BRM does not permit a corrective bill to be generated for that bill.
- **Bill unit hierarchy.** When charges are corrected for a bill in a bill unit hierarchy, a corrective bill is made for parent and child bill units. A paying parent bill unit includes the corrective bill for a nonpaying child bill unit. You cannot create a corrective bill for only a nonpaying child bill unit.
- **Rerating accounts that use open item accounting.** If you rerate events in billing cycles other than the current billing cycle for accounts which have open item accounting, you should generate corrective bills for those previous bills. In such a scenario, it is recommended that you generate corrective bills and corrected invoices for all the previous bills that were affected by the rerating. When you generate corrected invoices for such corrective bills, the rerated events are displayed in the corrected invoices. They will not be displayed in the regular bill for the current cycle. If you do not create a corrective bill for the previous bill, these rerated events are included in the regular bill for the current cycle.
- **Discount and charge sharing.** You must rerate accounts following updates to the discount and charge sharing for those accounts in a billed period. BRM includes the resulting adjustments in the corrective bills you generate for the billed period.
- **Collections.** BRM does not initiate payment collection actions for bills that have been replaced by a corrective bill. See "Post-Processing Actions for Corrective Bills" for more information.
- **A/R actions.** Disputes, settlements, and write-offs impact the content of a corrective bill. To prevent the generation of corrective bills for certain A/R actions, set up a custom validation policy for BRM to use when you generate the corrective bills. Such a validation could be used to prevent BRM from generating corrective bills when the prior bill is in dispute and a settlement has not been reached.
- **Adjustments created by rerating.** BRM creates unallocated items for any adjustment items that are created as a result of rerating. If you require that corrective bills should contain the aggregation and allocation of automatic adjustments to the items on the original bill, you must enable the **AllocateReratingAdjustments** business parameter before you rerate the original bills.

Corrective Bills and Billing Cycles

You can have multiple corrections for a single bill. BRM generates corrective bills differently based on whether the corrections occur in the same cycle or in multiple cycles.

When there are multiple corrections in a single bill cycle, BRM corrects the most recent corrective bill for the cycle. BRM includes first-time corrections on the most recent corrective bill and also events or items that were previously corrected and billed on previous corrective bills.

When corrections span multiple billed cycles, BRM uses the accounting method specified in the account to determine the cycle for which to generate the corrective bill.

- **Balance forward.** For accounts using the balance forward accounting method, BRM generates a single corrective bill for the last billed cycle.
- **Open item.** For accounts using the open item accounting method, BRM creates corrective bills for each billed cycle which contains corrections.

  For example, an account has corrections in the October bill and the November bill. BRM generates two corrective bills, one for October and one for November. If the October bill was already paid, BRM includes the corrections for the October cycle to the next open bill, which in this case is the bill for December. The corrections for the November cycle are included in the November bill because that bill is still pending payment.

Configuring Corrective Billing

Complete the following to set up the corrective billing process in BRM. Some steps are optional.

- Enable corrective billing in BRM. See "Enabling Corrective Billing".
- Restrict CSR permissions, as necessary. See "Restricting Corrective Billing Permissions".
- Set up a numbering scheme for corrective bill numbers. See "Configuring Bill Numbers for Corrective Bills".
- Create custom reasons for the correction. BRM provides default reasons, but you can add to this list. See "Customizing Correction Reasons".
- Specify the threshold amount for corrective bills. See "Specifying the Minimum Threshold Amount for Corrective Bills".
- Do not permit the processing of payment for prior bills. See "Rejecting Payments for Prior Bills".
- Set up the handling of corrections for partially or fully-paid bills. See "Enabling BRM to Create Corrective Bills for Partially or Fully Paid Bills".

Enabling Corrective Billing

To enable corrective billing, run the **pin_bus_params** utility to change the **EnableCorrectiveInvoices** business parameter. For information on this utility, see "pin_bus_params" in *BRM Developer's Guide*.

To enable corrective billing:

1. Go to *BRM_home***/sys/data/config**.
2. Create an XML file from the **/config/business_params** object:

```
pin_bus_params -r BusParamsBilling bus_params_billing.xml
```

3. In the XML file, change **disabled** to **enabled**:

   ```
   <EnableCorrectiveInvoices>enabled</EnableCorrectiveInvoices>
   ```

4. Save the file as **bus_params_billing.xml**.
5. Load the file into the BRM database:

   ```
   pin_bus_params bus_params_billing.xml
   ```

6. Stop and restart the CM.

Configuring Accounts and Bills for Corrective Billing

When you run the **pin_make_corrective_bills** utility, you can use the command line to specify the bill numbers and account numbers that need corrective bills. In addition, you can include account numbers and bill numbers in a file that the **pin_make_corrective_bills** utility reads.

The default file is **pin_bill_run_control.xml** in the *BRM_home***/apps/pin_billd** directory. See "Managing Large Billing Runs" for a sample billing run configuration file.

To verify that the selected accounts or bills are eligible for corrective billing, run the following command:

```
pin_make_corrective_bills -validate_only
```

Restricting Corrective Billing Permissions

By default, BRM provides all CSRs with permission to create corrective bills.

To restrict the permission for CSR roles:

1. In Permissioning Center, restrict the permissions for the required CSR roles. For more information on setting up permissions, see "Setting Up Permissions in BRM Applications" in *BRM System Administrator's Guide*.
2. Stop and restart the CM.

Configuring Bill Numbers for Corrective Bills

Corrective billing uses a different bill number format. Regular bills use a bill number such as B1-189, but a corrective bill uses a number such as CB1-15. A corrective bill applied to a corrective bill uses the same prefix and the next available sequence; for example, CB1-207.

However, you can configure BRM to apply the regular billing number format to corrective bills. In this case, BRM retains the original bill's prefix and uses the next available sequence number in BRM. For example, for bill number B1-189, the corrective bill number could be B1-2007.

To apply the regular bill numbering format to corrective bills:

1. Go to *BRM_home***/sys/data/config**.
2. Create an XML file from the **/config/business_params** object:

   ```
   pin_bus_params -r BusParamsBilling bus_params_billing.xml
   ```

3. In the file, set the **<GenerateCorrectiveBillNo>** element to **disabled**:

   ```
   <GenerateCorrectiveBillNo>disabled<GenerateCorrectiveBillNo>
   ```

4. Save the file as **bus_params_billing.xml**.

5. Load the XML file into the BRM database:

   ```
   pin_bus_params bus_params_billing.xml
   ```

6. Stop and restart the CM.

Customizing Correction Reasons

You can specify correction reasons:

- When you create a corrective bill from Billing Care or Customer Center. You can select a reason for the correction.
- When you run the **pin_make_corrective_bills** utility. You provide the correction reason.

BRM provides the following default reasons for correcting a bill:

- Update to the invoice address
- Manual adjustment
- Price correction

You can provide custom corrections reasons for bill correction by modifying an existing reason code or adding a custom reason code to the **reasons.en_US** file. BRM assigns the IDs of **1**, **2**, and **3** respectively to these reasons and uses **43** in the version field to specify the domain of the reason.

Here is an example code sample to add a reason code:

```
DOMAIN = "Reason Codes-Bill Correction Reasons";
STR
    ID = 25;
    VERSION = 43;
    STRING = "Corrective bill request for Error in Promotion ABC";
END
```

When you add or modify a reason code, ensure that **ID** is a number greater than **3** and the value for **version** is **43** (see the preceding example). These values are reserved by BRM.

Use the **load_localized_strings** utility to load the modified **reasons.en_US** file. For instructions on providing custom reason codes, see "String Manipulation Functions" in *BRM Developer's Reference*. For instructions on the **load_localized_strings** utility, see "load_localized_strings" in *BRM Developer's Guide*.

Specifying the Minimum Threshold Amount for Corrective Bills

You can specify a minimum amount that A/R actions for a bill should reach for BRM to generate a corrective bill. If the bill to be corrected belongs to a bill unit hierarchy, BRM checks that threshold against the sum of A/R actions for all bill units that had charges contributing to that bill.

To specify the minimum threshold amount for corrective bills:

1. Go to *BRM_home***/sys/data/config**.
2. Create an XML file from the **/config/business_params** object:

   ```
   pin_bus_params -r BusParamsBilling bus_params_billing.xml
   ```

3. Set the **<CorrectiveBillThreshold>** element to the required value. For example:

   ```
   <CorrectiveBillThreshold>15</CorrectiveBillThreshold>
   ```

The default is **0**.

4. Save the file as **bus_params_billing.xml**.
5. Load the XML file into the BRM database:

   ```
   pin_bus_params bus_params_billing.xml
   ```
6. Stop and restart the CM.

Rejecting Payments for Prior Bills

Corrections to bill charges and payments may not always be synchronized operations. That is, you may receive payments where the bill number associated with a payment does not match the last bill, or you may have received a partial or full payment for that original or corrective bill.

You can configure BRM to reject payments for prior bills, and thereby require them to be specific to the bill. To enable this feature, run the **pin_bus_params** utility to change the **RejectPaymentsForPreviousBill** business parameter. For information about this utility, see "pin_bus_params" in *BRM Developer's Guide*.

To reject payments for prior bills and require them to be specific to the bill:

1. Go to *BRM_home***/sys/data/config**.
2. Create an XML file from the **/config/business_params** object:

   ```
   pin_bus_params -r BusParamsBilling bus_params_billing.xml
   ```
3. In the file, set the **<RejectPaymentsForPreviousBill>** element to **enabled**:

   ```
   <RejectPaymentsForPreviousBill>enabled</RejectPaymentsForPreviousBill>
   ```
4. Save the file as **bus_params_billing.xml**.
5. Load the XML file into the BRM database:

   ```
   pin_bus_params bus_params_billing.xml
   ```
6. Stop and restart the CM.

Enabling BRM to Create Corrective Bills for Partially or Fully Paid Bills

If BRM receives a partial or full payment for a bill, it does not generate a corrective bill. Instead, BRM assigns all payments for that bill period to the next open bill.

To enable BRM to generate a corrective bill for a bill that has a full or partial payment, set the **AllowCorrectivePaidBills** parameter in the **billing** instance of the **/config/business_params** object to **enabled**. BRM then generates the corrective bill, and the balance on the corrective bill reflects the payment that was received against the prior bill. The corrective bill also displays the payment details for payments already processed against the prior bill.

To enable BRM to generate corrective bills for fully paid or partially paid bills:

1. Go to *BRM_home***/sys/data/config**.
2. Create an XML file from the **/config/business_params** object:

   ```
   pin_bus_params -r BusParamsBilling bus_params_billing.xml
   ```
3. In the file, set the **<AllowCorrectivePaidBills>** element to **enabled**:

   ```
   <AllowCorrectivePaidBills>enabled</AllowCorrectivePaidBills>
   ```
4. Save the file as **bus_params_billing.xml**.

5. Load the XML file into the BRM database:

```
pin_bus_params bus_params_billing.xml
```

6. Stop and restart the CM.

Customizing Corrective Bills

You can customize corrective bills in the following ways:

- **Customize due dates.** You can apply different due dates to corrective bills by customizing the bill due date calculations. Modify the PCM_OP_BILL_POL_CALC_PYMT_DUE_T policy opcode to retain the same due date as in the prior bill, or add a specific number of offset days.
- **Change the default invoice type.** Use the PCM_OP_CUST_POL_PREP_PAYINFO policy opcode to override the default value (detailed replacement invoice).
- **Define custom validations.** To define custom validations for corrective billing, modify the existing validations in the PCM_OP_BILL_POL_VALID_CORRECTIVE_BILL policy opcode or add custom validations.

For more information, see "Corrective Billing" in *BRM Opcode Guide*.

Billing Accounts By Using the pin_make_corrective_bill Utility

You can generate corrective bills in one of the following ways:

- Allocate adjustments to a selected bill in Billing Care or Customer Center, and then submit the bill for corrective billing.
- Run the **pin_make_corrective_bills** utility which generates the corrective bill.

The **pin_make_corrective_bills** utility calculates the balance due after allocating the adjustments and A/R actions for each account bill unit, and creates a corrective bill for the balance due. It creates corrective bills for bill units whose bills have corrections that fall within the period you specify.

The balance due amount is the amount requested as a payment by the **pin_collect** utility and the amount that is shown on the corrective invoice.

For information about the **pin_make_corrective_bills** utility syntax, see "pin_make_corrective_bills".

Run the **pin_make_corrective_bills** utility when you have allocated all adjustments to prior bills.

When you generate corrective bills for a bill unit hierarchy, you must run the **pin_make_corrective_bills** utility for the topmost parent bill unit.

You must run the **pin_make_corrective_bills** utility before you run **pin_collect** because **pin_collect** needs the balance due amount collected by the **pin_make_corrective_bill** utility.

After you begin the process of generating a corrective bill, you cannot cancel the process.

To generate a corrective bill using **pin_make_corrective_bills**:

1. Go to *BRM_home*/**apps/pin_billd**.
2. Run the **pin_make_corrective_bills** utility:

```
pin_make_corrective_bills
```

For example, you can:

- Use the **-corrective_inv_type** parameter to specify the invoice type (replacement or invoice correction letter).
- Use the **-threshold_amount** parameter to override the threshold amount for generating a corrective bill.
- Use the **-account_no** parameter to specify the accounts to create corrective bills for.
- Use the **-create_if_no_corrections** parameter to generate a corrective bill when there are only simple changes such as invoice address changes. Use this parameter with the **-corrective_inv_type R** parameters.

See "pin_make_corrective_bills".

3. Verify the contents of the corrective bills by using Billing Care or Customer Center.

Post-Processing Actions for Corrective Bills

After you generate corrective bills, you can do the following:

- Generate corrective invoices by running the **pin_inv_accts** utility. You can generate summary or detailed corrective invoices for the corrective bills you generated. See "pin_inv_accts" in *BRM Designing and Generating Invoices*.
- Export corrective invoices for use with custom programs, (such as DOC1) to generate and publish the corrective invoice documents. Use the **pin_inv_export** utility to export invoices to a format you can use with other programs, such as DOC1. See "pin_inv_export" in *BRM Designing and Generating Invoices*.
- Collect any balance due for accounts that use credit card and direct debit payment methods by running the **pin_collect** utility. See "pin_collect" in *BRM Configuring and Collecting Payments*.
- Check whether corrective billing affected the state of any prior bills that were in collections by running the **pin_collections_process** utility. For more information, see "pin_collections_process" in *BRM Collections Manager*.

Corrective Billing for Disputes, Settlements and Write-offs

Disputes, settlements, and write-offs impact the content of a corrective bill. If you must prevent the generation of corrective bills for certain A/R actions, set up a custom validation policy for BRM to use when you generate the corrective bills. Such a validation could be used to prevent BRM from generating corrective bills when the prior bill is in dispute and a settlement has not been reached.

Disputes

Corrective bills can be generated for a dispute entered at the bill level, item level or event level. When BRM generates the corrective bill for a prior bill in dispute, the corrective bill contains the details of the dispute, the disputed bill item under the corrected items, and the bill balance reduced by the disputed amount.

Settlements

Corrective bills can be generated for a dispute settled at the bill level, item level or event level. When BRM generates the corrective bill when a bill item on the prior bill is in settlement, the corrective bill contains the details of the settlement, the settled bill item under the corrected items, and the bill balance reduced or increased by the settlement.

A dispute may be settled for the full amount of the dispute in the customer's favor. If you submitted a corrective bill at the time of such a dispute, the corrective bill at settlement time may become a duplicate as there may not be any additional corrections to the second bill.

Write-Offs

You perform write-offs after the account is past due and considered delinquent in collections. When BRM generates the corrective bill for a write-off on a bill item on the prior bill, the corrective bill contains the details of the write-off, the written-off bill item under the corrected items, and the bill balance reduced by the write-off amount.

18 Running Trial Billing

Learn how to run trial billing to validate billing results in Oracle Communications Billing and Revenue Management (BRM) without billing your customers.

Topics in this document:

About Trial Billing

Trial billing is a process that simulates BRM billing. You use trial billing to validate billing results without billing customers.

The trial billing utility simulates the billing functions of the **pin_bill_accts** utility. It can additionally create and store trial invoices in the BRM database.

You can also collect revenue assurance data for trial bills and display the data by generating Revenue Assurance Billing reports.

In case of billing discrepancies, you can make corrections, such as adjustments or payment allocations, or you can run rerating. When you are satisfied with the results from trial billing, run actual billing to generate the final bills.

To perform trial billing, you run the "pin_trial_bill_accts" utility.

Table 18-1 summarizes the similarities and differences between regular billing and trial billing.

Table 18-1 Comparisons Between Regular Billing and Trial Billing

Billing with pin_bill_accts	Trial Billing with pin_trial_bill_accts
Performs billing on an account's billing day of month.	Performs trial billing before or after an account's billing day of month.
Creates cycle forward, usage item, and bill objects in the BRM database.	Creates cycle forward and usage item objects, but the objects are not recorded in the BRM database.
Calculates and totals the balance impacts for the previous billing cycle.	Calculates and totals the balance impacts for any billing cycle.

Table 18-1 (Cont.) Comparisons Between Regular Billing and Trial Billing

Billing with pin_bill_accts	Trial Billing with pin_trial_bill_accts
Calculates and updates account balances.	Calculates but does not update account balances.
Does not create invoices.	Creates and stores trial invoices in the BRM database unless you specify not to.
Balance due in the bill object shows the exact amount that is due.	Balance due in the trial invoices may not show the exact amount due at time of actual billing. This is because events that have a balance impact can occur after you run trial billing.

Note

With regular billing, invoices are created separately by running the **pin_inv_accts** utility. The trial billing utility simulates billing and *additionally* creates and stores trial invoices. For this reason, a trial billing run that generates invoices takes longer than actual billing. You have the option to not generate trial invoices if you do not need them. See "Creating Trial Bills without Generating Trial Invoices".

About Trial Billing for Bill Unit Hierarchies

Bill unit hierarchies establish a parent-child relationship where the parent bill unit is responsible for paying the charges of its non-paying child bill units that are on the same billing cycle. Trial billing for bill unit hierarchies differs from regular billing in how charges are calculated and rolled up.

In regular billing, charges for nonpaying child bill units are calculated first and then rolled up to the paying parent bill unit. BRM creates a database lock for each nonpaying bill unit, computes the balance impacts, and then releases the lock.

In trial billing, the paying parent and all its nonpaying child bill units are locked simultaneously. The parent bill unit and the nonpaying child bill units remain locked until the balance impacts of all nonpaying bill units are computed and rolled up to the parent bill unit.

Note

- As with regular billing, BRM considers the invoicing threshold value for bill unit hierarchies during trial billing. If it exceeds the threshold value (the number of nonpaying bill units is greater than the threshold value), a separate invoice is generated for each nonpaying bill unit by using multiple threads.
- When hierarchical bill units are selected for trial billing, access to the parent and child bill units is prevented, and system performance is decreased due to the database locks.

About Trial Billing for Sharing Groups

In sharing groups, sharing occurs when an account is added to a sharing group and a discount is added to that account. The sharing group owner account and member account can have different billing cycles. When a sharing group owner is selected for trial billing, the trial invoice might not contain all the fees from member accounts.

In regular billing, when a sharing group owner account is billed, BRM processes the member accounts to determine the sponsored fees to apply to the owner account. The sponsored cycle events of member accounts are accumulated in the **/item/sponsor** object of the owner account and appear on the owner's bill.

In trial billing, when a sharing group owner account is selected, trial billing is not run for member accounts. In this case, the owner's trial invoice might not contain the recurring charges for the sponsored cycle events of the member accounts because such charges are generated by running billing. The owner's trial invoice would, however, contain the members' usage charges because those charges are not generated by billing.

Trial billing generates temporary bills for validation, ensuring accurate charge calculation during the final billing process. If any discrepancies arise in the trial bills or charges, customers can adjust the charges and rerun trial billing to validate them.

In wholesale bill unit hierarchies, trial billing should be run for all non-paying subordinate accounts and paying parent accounts. When customers make corrections or adjustments to a few bills, they can rerun trial billing for only the affected accounts, minimizing processing time and ensuring efficient performance across the system.

About Trial Invoices

A *trial invoice* is a statement of charges and the balance that is due. You use trial invoices to validate billing charges before creating final bills for your customers. Trial invoices are generated when you run the trial billing utility (**pin_trial_bill_accts**). For information about creating trial invoices, see "Running Trial Billing".

Note

- Trial invoices are created only for billing cycles that have not been billed yet.
- Trial invoices are optional. If you do not need trial invoices, you can specify not to generate them: for example, when the revenue assurance data collected from trial billing is sufficient for validating the billing results.
- Session or activity charges incurred between the dates of actual billing and trial billing do not appear on trial invoices.

You can design your own invoice templates for displaying trial invoices, and you can customize the information displayed on trial invoices in the same ways you do for regular invoices.

You can also purge and export trial invoices. See "Purging Trial Invoices" and "Exporting Trial Invoices".

Note

Trial invoices are *always* stored in the primary BRM database schema. In a multischema system, regular invoices can be stored in a separate schema.

About Collecting Revenue Assurance Data from Trial Billing

When you enable trial billing to generate revenue assurance data, the data is automatically generated when you run the **pin_trial_bill_accts** utility. You can use this data to validate overall billing results. To view the data, run a Revenue Assurance Billing Summary report.

You enable trial billing to generate revenue assurance data by setting an entry in the trial billing configuration file (*BRM_home***/apps/pin_trial_bill/pin.conf**). For more information, see "Enabling Billing Utilities to Generate Revenue Assurance Data" in *BRM Collecting Revenue Assurance Data*.

You can also split revenue assurance data collected from trial billing into more detailed categories by running **pin_trial_bill_accts** with the **-split** parameter. The split data is based on the type of billable item and its associated services. To view split revenue assurance data, run a Revenue Assurance Billing Detail report.

You can customize how revenue assurance data is split by configuring item subtypes for the type of revenue, such as one-time usage, recurring charges, adjustments, and payments.

For information about revenue assurance data, see "About Collecting Revenue Assurance Data from Billing" in *BRM Collecting Revenue Assurance Data*.

Configuring Trial Billing

You can configure the following options for trial billing:

- Specifying Accounts for Trial Billing
- Specifying Bill Units, Billing Segments, and DOMs for Trial Billing
- Creating Trial Bills without Generating Trial Invoices

Specifying Accounts for Trial Billing

To run trial billing for a small number of random accounts, set the **pin_trial_bill_accts** flag in the Connection Manager (CM) configuration file (*BRM_home***/sys/cm/pin.conf**) to the maximum number of accounts to bill:

```
- pin_trial_bill_accts threshold numberOfAccounts
```

To specify a list of specific accounts to be trial billed, use the **-f** parameter:

```
pin_trial_bill_accts -end 4/1/2002 -f inputFile
```

Note

You can also use the **-f_control** parameter as an alternative way of specifying accounts for trial billing. See "Specifying Bill Units, Billing Segments, and DOMs for Trial Billing".

The input file lists the account POIDs and bill unit POIDs in flist array format. Each array element corresponds to one account and bill unit.

Note

- Trial billing stops and reports a warning message when it encounters an account or bill unit with inactive status.
- All accounts listed in the input file will be trial billed.
- If a selected account has a nonpaying bill unit, **pin_trial_bill_accts** does not create a trial bill for it. Nonpaying bill units are trial billed only when their parent accounts are selected for trial billing.
- If an account that has not been billed for some time is selected for trial billing, a trial invoice is generated for each billing cycle that was skipped. The invoice data is stored in shared memory until all the billing cycles have been processed. If the account skipped many billing cycles, this could result in a PIN_ERR_NO_MEM error, indicating that the system does not have enough shared memory to process that account.

Specifying Bill Units, Billing Segments, and DOMs for Trial Billing

You can run trial billing for bill units associated with specified accounting DOMs, specified billing segments, or a combination of DOMs and segments. You list the DOMs and billing segments in a trial-billing-run configuration file. You can also use this file to specify a list of accounts and bill units for trial billing.

You create a trial-billing-run configuration file by copying and modifying the BRM billing run configuration file (*BRM_home*/**apps/pin_billd/pin_bill_run_control.xml**). You then specify this file when you run **pin_trial_bill_accts** with the **-f_control** parameter.

Note

- Using a trial-billing-run configuration file to specify accounts for trial billing is an alternative way of specifying accounts in flist format by using the **-f** parameter. See "Specifying Accounts for Trial Billing".
- The **pin_bill_run_control.xml** file is the same file used to split a regular billing run into multiple runs. See "Configuring a Split Billing Run".

To specify accounts and bill units, DOMs, or billing segments for trial billing:

1. Open the **pin_bill_run_control.xml** file in an XML editor or a text editor.

   By default, the file is in the *BRM_home*/**apps/pin_billd** directory.

2. Edit the file to specify the bill units to trial bill:
   - To specify accounts and their bill units, see "Specifying Bill Units for Trial Billing".
   - To specify accounting DOMs, see "Specifying Accounting DOMs for Trial Billing".
   - To specify billing segments, see "Specifying Billing Segments for Trial Billing".

3. Save the file under a different name and close it. Give the file a meaningful name: for example, if trial billing a group of accounts, include the account range or reason for billing in the file name; if trial billing specific DOMs, include the DOM range in the file name.

> **Note**
>
> When you run **pin_trial_bill_accts**, the configuration file you create and the default *BRM_home***/apps/pin_billd/business_configuration.xsd** file must be in the same directory.

4. Run **pin_trial_bill_accts** with the **-f_control** parameter:

   ```
   pin_trial_bill_accts -end 4/1/2002 -f_control filename
   ```

   where *filename* is the name of the trial-billing-run configuration file that you created. If *filename* is in a different directory from which you run **pin_trial_bill_accts**, you must also include the entire path for the file.

For more information, see "pin_trial_bill_accts".

Specifying Bill Units for Trial Billing

To generate trial bills for a set of accounts and bill units, add a **BillingList** parent element in the trial-billing-run configuration file for each account and bill unit to include. In the **BillingList** parent element, add an **Account** child element that specifies the POID of the account, and add a **Billinfo** child element that specifies the POID of the bill unit.

> **Note**
>
> - Trial billing stops and reports a warning message when it encounters an account or bill unit with inactive status.
> - To trial bill specific accounts and bill units, you must include both the account POID and bill unit POID. If only one is specified, the account or bill unit is not trial billed.

For example, the following **BillingList** parent element generates trial bills only for the bill unit with POID 64295 that belongs to the account with POID 17763:

```
<BillingList>
    <Account>17763</Account>
    <Billinfo>64295</Billinfo>
</BillingList>
```

To specify multiple accounts or multiple bill units from the same account, add a **BillingList** parent element for each bill unit. For example, the following entries generate two trial bills for the account with POID 17763 and one trial bill for the account with POID 25147:

```
<BillingList>
    <Account>17763</Account>
    <Billinfo>64295</Billinfo>
</BillingList>
<BillingList>
    <Account>17763</Account>
    <Billinfo>68439</Billinfo>
</BillingList>
<BillingList>
    <Account>25147</Account>
    <Billinfo>314552</Billinfo>
</BillingList>
```

Specifying Accounting DOMs for Trial Billing

To generate trial bills for bill units associated with specified accounting DOMs, add a **DOMList** parent element in the trial-billing-run configuration file. In the **DOMList** parent element, add a **DOM** child element that specifies the accounting DOM for each day whose bill units you want to include.

For example, the following **DOMList** parent element generates trial bills only for bill units whose accounting DOM is 1 or 15:

```
<DOMList>
    <DOM>---01</DOM>
    <DOM>---15</DOM>
</DOMList>
```

If the **DOMList** parent element is omitted, bill units associated with any accounting DOM can be included in the billing run.

Specifying Billing Segments for Trial Billing

To run trial billing for specified billing segments, you must first set up billing segments in your system and then associate them with bill units. For more information, see "Load Balancing Billing Runs".

To generate trial bills for bill units associated with specified billing segments, add a **BillSegmentList** parent element in the trial-billing-run configuration file. In the **BillSegmentList** parent element, add a **BillSegment** child element that specifies the billing segment ID for each billing segment whose bill units you want to include. The segment ID is the ID of any billing segment defined in the **/config/billing_segment** object in your BRM system.

For example, the following **BillSegmentList** parent element generates trial bills only for bill units associated with billing segments 101, 102, and 103:

```
<BillSegmentList>
    <BillSegment>101</BillSegment>
    <BillSegment>102</BillSegment>
    <BillSegment>103</BillSegment>
</BillSegmentList>
```

If the **BillSegmentList** parent element is omitted, bill units associated with any billing segment can be included in the billing run.

Rerunning Trial Billing After Making Corrections

After making corrections to specific billing accounts in a hierarchy, you can rerun trial billing to validate that it is generating the correct results. Rerun billing only on the bills you changed by running the **pin_trial_bill_accts** utility with the **-f_control** and **-pay_type** parameters.

To rerun trial billing:

1. Go to *BRM_home*/**apps/pin_trial_bill**.
2. Run the **pin_trial_bill_accts** utility:

   ```
   pin_trial_bill_accts [-pay_type [10007,10001]] [-f_control inputFile]
   ```

Running the utility without any parameters initiates trial billing for all accounts that are due for billing. The utility also allows running trial billing for a subset of accounts. For example, you can:

- Use the **-hierarchy** parameter to rerun trial billing for all accounts in a hierarchy.
- Use the **-f_control** parameter to rerun trial billing for a specified account in the input file.

Rerunning Trial Billing for All Child Accounts Under Specific Parent Accounts

To rerun trial billing for all child accounts under specified parent billing accounts:

1. Open the **pin_bill_run_control.xml** file in an XML editor or a text editor.

   By default, the XML file is in the *BRM_home*/**apps/pin_billd** directory.
2. Edit the file to add the required parent accounts.

   For example, these entries specify to rerun trial billing of parent accounts.

   ```
   <BillingList>
           <Account>101010</Account>
           <Billinfo>111111</Billinfo>
   </BillingList>

   <BillingList>
           <Account>202020</Account>
           <Billinfo>222222</Billinfo>
   </BillingList>
   ```
3. Run the following command to create trial bills for all child accounts in the hierarchy:

   ```
   pin_trial_bill_accts -pay_type 10007 -f_control inputFile -hierarchy
   ```

   The **pin_trial_bill_accts** utility runs trial billing for all child accounts under the specified parents.
4. Run the following command to create trial bills for the parent accounts in the hierarchy:

   ```
   pin_trial_bill_accts -pay_type 10001 -f_control inputFile -hierarchy
   ```

   Note

   - Each time you run this command, you can add a maximum of 25 parent accounts. However, there is no restriction on the number of child accounts within each hierarchy.
   - The command validates that all subordinates under the parent accounts have completed trial billing. If any subordinate account has not completed trial billing, the **pin_trial_bill_accts** utility reports an error.

Rerunning Trial Billing for Specific Child Accounts Under Parent Accounts

To rerun trial billing for specific child accounts under parent accounts:

1. Open the **pin_bill_run_control.xml** file in an XML editor or a text editor.

   By default, the XML file is in the *BRM_home*/**apps/pin_billd** directory.
2. Edit the file to add the required child accounts in a billing hierarchy.

   For example, these entries specify to rerun trial billing of child accounts in a billing hierarchy.

   ```
   <BillingList>
           <Account>100000</Account>
           <Billinfo>200000</Billinfo>
   </BillingList>

   <BillingList>
           <Account>300000</Account>
           <Billinfo>400000</Billinfo>
   </BillingList>
   ```
3. Run the following command to create trial bills for the child accounts in the input file:

   ```
   pin_trial_bill_accts -pay_type 10007 -f_control inputFile
   ```
4. Run the following command to create trial bills for the parent accounts in the input file:

   ```
   pin_trial_bill_accts -pay_type 10001 -f_control inputFile
   ```

   Note

   - When you run trial billing for the parent accounts, ensure that the *inputFile* includes POIDs and **/billinfo** object details associated with each parent account. This information is required for accurate trial billing.
   - If only a couple of children's accounts are trial billed, only the charges of those child accounts will be rolled up, and then the parent account will be trial billed.

Running Trial Billing

To run trial billing, use the "pin_trial_bill_accts" utility:

1. Go to *BRM_home*/**apps/pin_trial_bill**.
2. Run the **pin_trial_bill_accts** utility:

   ```
   pin_trial_bill_accts
   ```

   Running the utility with no parameters runs trial billing for all accounts that are due for billing. You can run trial billing for a subset of accounts. For example, you can:

   - Use the **-start** and **-end** parameters to run trial billing for bills in a specified date range. See "Running Trial Billing With Date Ranges".
   - Use the **-pay_type** parameter to run trial billing for a specific payment type.
   - Use the **-bill_only** to run trial billing without generating trial invoices and collect revenue assurance data.
   - Use the **-split** parameter to generate detailed revenue assurance data.

- Use the **-active**, **-inactive**, and **-closed** parameters to run trial billing for a specific account status.

See "pin_trial_bill_accts".

Running Trial Billing With Date Ranges

The **start** and **end** dates determine which accounts are selected for trial billing and which billing cycles for those accounts are trial billed.

The **end** date is used as the search criteria for retrieving accounts for trial billing. The search selects all accounts with a billing date less than the **end** date. You can specify either an absolute date, a number of days before or after the current date, or the current date (by specifying **0**).

> **Note**
>
> If you do not specify an **end** date, **pin_trial_bill_accts** uses the current date for the **end** date.

The **start** date determines the billing cycles for which trial invoices are generated. You can specify either an absolute date, a number of days before or after the current date, or the current date (by specifying **0**).

If you specify a **start** date, trial billing is run only when a billing cycle is completed between the start date and the end date. Trial billing is not run on partial cycles.

> **Note**
>
> If you do not specify a **start** date or if you specify **0** for the current date, **pin_trial_bill_accts** generates trial invoices for all billing cycles that were completed between the current date and the **end** date and that have not already been billed. More than one trial invoice might be generated for accounts that have not been billed for one or more billing cycles.

Examples:

- Create trial invoices for accounts whose billing date is on or before **4/1/2002** (current date is 3/15/2002):

  ```
  pin_trial_bill_accts -end 04/01/2002
  pin_trial_bill_accts -end +17
  ```

- Create trial invoices for accounts whose billing date is on or before **3/1/2002** (current date is 3/15/2002):

  ```
  pin_trial_bill_accts -end 03/01/2002
  pin_trial_bill_accts -end -14
  ```

- Create trial invoices for accounts whose billing date is on or before the current date:

  ```
  pin_trial_bill_accts -end 0
  pin_trial_bill_accts
  ```

- Create trial invoices for accounts with complete billing cycles between **4/1/2002** and **5/15/2002** (current date is 5/15/2002):

```
pin_trial_bill_accts -start 04/01/2002 -end 5/15/2002
pin_trial_bill_accts -start -44 -end 0
```

Running Trial Billing According to Payment Type

Generates trial invoices for accounts with the specified payment method ID. If you do not specify the payment method ID, the **pin_trial_bill_accts** utility generates trial invoices for all payment methods.

The *ID* can be any payment method ID from Table 18-2.

Table 18-2 Payment Method ID

Payment Method	ID
credit card	10003
debit card	10002
direct debit	10005
guest	10010
invoice	10001
prepaid	10000
SEPA	10018
subordinate	10007
undefined	0

For example, to generate trial invoices for all bills paid by the credit card payment method, use the following syntax:

```
pin_trial_bill_accts -pay_type 10003
```

For hierarchical bill units, you must run nonpaying child bill units before running paying parent bill units. For example, to generate trial invoices for hierarchical bill units whose payment method is by check, use the following syntax:

```
pin_trial_bill_accts -pay_type 10007
pin_trial_bill_accts -pay_type 10012
```

Note these limitations:

- The **-active**, **-closed**, or **-inactive** parameters are not supported for nonpaying child bill units when combined with the **-pay_type** parameter. These parameters limit the number of nonpaying child bill units that are invoiced, which leads to errors when their paying parent bill unit is trial billed.
- Do not set a threshold value. Setting a threshold value limits the number of nonpaying child bill units that are invoiced, which leads to errors when their paying parent bill unit is trial billed.

  Note

  The threshold entry is located in the *BRM_home*/**apps/pin_trial_bill/pin.conf** file.

- The **-f**, **-f_control**, and **-bill_only** parameters are not supported for all bill units when used with the **-pay_type** parameter.

Creating Trial Bills without Generating Trial Invoices

If the revenue assurance data collected from trial billing provides enough information for you to validate your billing charges and you do not need the specific information provided in invoices, you can run the trial billing utility with the **-bill_only** parameter. This parameter suppresses the creation of trial invoices. Generating trial invoices takes longer than regular billing; therefore, running trial billing without generating trial invoices improves the performance of trial billing and is equivalent in performance to running regular billing.

Purging Trial Invoices

To purge trial invoices, run the **pin_trial_bill_purge** utility.

By default, **pin_trial_bill_purge** purges invoices with the **/invoice/trial** POID type. If a custom **/invoice** subclass exists, invoices with **/invoice/***custom_subclass***/trial** POID type are purged.

> **Note**
>
> If the **/invoice** class is partitioned, running the purge utility (**partition_utils -o**) on **/invoice** purges both regular invoices and trial invoices.

For a description of the syntax and parameters of this utility, see "pin_trial_bill_purge".

For more information about trial invoices, see "About Trial Invoices".

Exporting Trial Invoices

To export trial invoices, run the **pin_inv_export** utility and use the **-trial** parameter.

You can export trial invoices in the same formats that are supported for regular invoices. To export trial invoices, follow the same procedures you use for exporting regular invoices. For more information, see "Exporting Invoices" in *BRM Designing and Generating Invoices*.

For a description of the syntax and parameters of this utility, see "pin_inv_export" in *BRM Designing and Generating Invoices*.

19
Suspending Billing

Learn how to suspend billing for a select number of accounts in Oracle Communications Billing and Revenue Management (BRM).

Topics in this document:

- About Suspending Billing of Accounts and Bills
- Suspending Billing of Closed Accounts
- Suspending Billing of an Account's Bill

About Suspending Billing of Accounts and Bills

By default, BRM generates bills for *all* bill units in all accounts. When you run billing in BRM, active accounts as well as inactive and closed accounts are billed. However, you may have accounts or bill units that you do not want to bill. You can suspend billing for those accounts and bill units. You can later resume spending on suspended accounts and bill units.

Note

- By excluding accounts or bill units that do not need to be billed, you can reduce the time it takes to complete your billing run.
- If you suspend an account, all nonpaying child bill units of the suspended account's paying parent bill units are also suspended.
- To *suppress* billing, see "About Suppressing Bills". Unlike bill and account suspension, bill suppression does not inactivate bill units.
- For information about another way to reduce the duration of your billing run, see "Splitting a Billing Run into Multiple Runs".

Suspending Billing of Closed Accounts

By default, you can configure BRM to enable or disable billing of closed accounts. When billing of closed accounts is *disabled*, BRM excludes closed accounts from the billing run when the following conditions are met:

- The account's total balance due for every bill unit is zero.

  Note

  The total balance due includes the totals from the nonpaying child bill units of an account's paying parent bill units.

- The account has had no billable activity since the previous bill.

To enable billing of closed accounts:

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

   ```
   pin_bus_params -r BusParamsBilling bus_params_billing.xml
   ```

   This command creates an XML file named **bus_params_billing.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_billing.xml.out**, set **StopBillClosedAccounts** to **enabled**:

   ```
   <StopBillClosedAccounts>enabled</StopBillClosedAccounts>
   ```

   The default value is **disabled**.

   > **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_billing.xml.out** file to **bus_params_billing.xml**.
6. Use the following command to load your changes into the **/config/business_params** object:

   ```
   pin_bus_params bus_params_billing.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 Connection Manager (CM).

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

Suspending Billing of an Account's Bill

If you have additional bill units in accounts that you do not want to bill, such as inactive bill units, you can customize BRM to suspend billing of those bill units.

20

Setting Up Billing in a Multischema Environment

Learn how to set up billing in an Oracle Communications Billing and Revenue Management (BRM) multischema environment.

Topics in this document:

Setting Up Billing to Run in a Multischema Environment

You can run billing in a multischema BRM environment on one database schema at a time by using one instance of the billing utilities or on multiple schemas simultaneously by using multiple instances of the billing utilities. In other words, to bill accounts on a specific database schema, you must run the billing utilities from that database schema.

For instance, to bill accounts that reside on the primary and secondary database schemas, you can run the billing utilities from the primary schema to bill the accounts that reside on the primary database schema and run another instance of the billing utilities from the secondary schema to bill the accounts that reside on the secondary database schema.

Note

When running the **pin_bill_day** script with the **-file** option, ensure that the accounts specified in the billing run configuration file reside on the same database schema where **pin_bill_day** is run. If the file contains accounts from different database schemas, **pin_bill_day** reports an error.

Running Billing on One Schema at a Time

Running billing utilities on multiple database schemas one at a time requires that you edit the billing utility configuration file before each time you run the billing utilities. Perform the following steps before each time you run billing:

1. Open the billing utility configuration file (*BRM_home*/**apps/pin_billd/pin.conf**) in a text editor.
2. Change the value of the **userid** entry to the database schema against which you want to run billing. For example, to run billing on the 0.0.0.2 schema, change the **userid** entry as follows:

```
- - userid 0.0.0.2 /service/pcm_client 1
```

3. Change the value of the **login_name** entry to an account that resides in the schema against which you want to run billing. For example, to run billing using the account root.0.0.0.2, change the **login_name** entry as follows:

   ```
   - nap login_name root.0.0.0.2
   ```

4. Save the file.
5. Run the billing utilities.

Running Billing on Multiple Schemas Simultaneously

Running billing on multiple database schemas simultaneously requires that you create parallel instances of the billing utility configuration files, each of which is configured for a particular schema. Then you run all instances of your billing utilities.

1. For each schema that you want to run billing on, create a subdirectory under *BRM_home***/apps/pin_billd**.

   For example, *BRM_home***/apps/pin_billd/db1** for database 1, *BRM_home***/apps/pin_billd/db2** for database2, and so on.
2. Copy the *BRM_home***/apps/pin_billd/pin.conf** file into each new subdirectory.
3. In each billing subdirectory, do the following:
   a. Open the **pin.conf** file.
   b. Change the database number in the **login_name** entry to a database account that resides in the schema against which you want to run billing. For example, to run billing against schema 0.0.0.2, change the **login_name** entry as follows:

      ```
      - nap login_name root.0.0.0.2
      ```

   c. Save the file.
4. Run the billing utilities from the new subdirectories.

21

Remitting Funds to Third Parties

Learn how to set up Oracle Communications Billing and Revenue Management (BRM) to remit a share of revenues to third parties.

Topics in this document:

About Remittance

Use the BRM remittance feature to share the revenue you receive with third parties, such as resellers or service providers. You can direct BRM to calculate the amount of remittance in various ways. For example, you can pay third parties a percentage of subscriber fees or a flat amount per new subscriber.

Note

Settlements is a widely used industry term for *remittance*.

Table 21-1 lists some scenarios for using the BRM remittance feature:

Table 21-1 Remittance Scenarios

Scenario	Description
Service or Product Provider Payments	An Internet service provider (ISP) offers a service or product from a different company and needs to remit part of the revenue from the service or product to that company. For example, an ISP offers online games to its subscribers for an extra fee and pays a portion of those fees to the company that provides the games.
Sales Commissions	An ISP pays a commission for each subscriber its sales person signs up. **Note:** Sales commissions are supported only if the profile of the subscriber account contains information about a salesperson. See "Using Remittance for Sales Commissions".

Table 21-1 (Cont.) Remittance Scenarios

Scenario	Description
Telephony Settlements	IP telephony calls connect with gateways and other telephony networks. A portion of the revenues collected for these calls go to the other carriers. For example, you must pay a call termination fee to the carrier that completes a call. **Note:** Implementing this example in BRM requires customizing the remittance fields file. See "Defining Custom Remittance Fields".

About Remittance Products

Using the remittance feature requires you to create special products as part of your price list. BRM uses a remittance product as the basis for calculating the remittance amount to be paid to a third party.

You use Pricing Center to create a remittance product and specify the following:

- The product applies to Account, instead of to a service.
- The event type is Remittance Event.
- The rate is measured by one of these metrics:
  - Number
  - Usage Time
  - Usage Size
  - Amount
  - A custom metric you create and load into BRM. For information, see "Configuring Ratable Usage Metrics (RUMs)" in *PDC Creating Product Offerings*.

Note

* You cannot mix remittance and non-remittance products in a deal or plan.
* You can include one or more remittance products in a deal.

For more information, see "Creating a Remittance Product ".

About Defining Remittance Specifications

A remittance specification defines a single remittance arrangement that specifies which third party receives remittance when particular events occur. A specification also includes the product BRM uses to calculate remittance when the criteria are met.

You define specifications in the remittance specification file. Each specification includes the following information:

- Account number of the account that receives payment.
- Status of the events that contribute to remittance. You can specify that you pay third parties only when the events have been billed or paid, or you can specify that you pay without reference to the billing status.

- Name of the product that determines the rate that BRM uses to calculate remittance.

> **Note**
>
> Whenever a product name changes, you must update the remittance specification and reload it into the database.

- Remittance criteria that specify which events trigger payments to the remittance account. See "About Remittance Criteria".

> **Note**
>
> – Remittance is not supported for pipeline-rated events.
>
> – You cannot use the same combination of remittance account and remittance product in more than one specification.
>
> – You cannot see the balance owed to a remittance account until you run the remittance utility.

About Remittance Criteria

You define a remittance criterion by assigning a value to a remittance field. Each field represents an attribute of a storable class.

The following remittance fields are available by default:

- service type
- product name
- event type
- name of a profile associated with an account

For example, you can specify that all cycle forward events for the product Internet Access and **/service/ip** contribute to remittance.

> **Note**
>
> You must define an event type as one of your remittance criteria.

These fields are defined in the remittance fields file and then loaded into the BRM database.

A technical person can also create additional custom fields. Do this if you want remittance to depend on criteria other than the defaults. For example, if you want remittance to depend on a telephony gateway, define custom fields in this file.

Customizing the remittance fields file requires an understanding of BRM storable classes. For more information, see "Defining Custom Remittance Fields".

About Calculating Remittance

You run the remittance utility, "pin_remittance", to calculate the amount you must pay to third parties.

When an event occurs that meets the defined criteria, BRM stores the remittance information about the event. BRM later uses the stored information to calculate payments when you run the remittance utility.

When BRM rates an event, it runs an opcode to evaluate the criteria defined in your remittance specification. If the event meets a set of criteria, BRM stores information relevant to remittance, such as the remittance account and product.

The remittance utility uses that information to calculate the amount to pay each remittance account.

The remittance utility does the following:

- Collects the remittance information that BRM previously stored in separate objects.
- Creates a new event for each combination of remittance account and product.
- Calculates the amount to pay each account for each event by rating the event and stores that data for reporting purposes.

Typically, you run the remittance utility monthly. You can run it separately or as part of the monthly remittance script, **pin_remit_month**.

Before running the remittance utility, you should first run billing on all accounts *except* remittance accounts. This is especially true if you defined your remittance specifications so that events that contribute to remittance must be billed or paid before you pay the third parties. For more information, see "Running Billing Scripts".

You then do the following, either by running **pin_remit_month** or as separate steps:

1. Run the remittance utility to calculate the amount owed to each remittance account.
2. Run the billing utility on only the remittance accounts.
3. Run invoicing on only the remittance accounts.

For more information on how to calculate remittance, see "Calculating Remittance ".

Setting Up Remittance

To set up remittance, use the following steps. Each step includes a link to a detailed procedure.

1. Create one or more remittance products in Pricing Center, based on the remittance event, and include them in a deal. See "Creating a Remittance Product ".
2. Create an account in Customer Center for each third party that you want to receive funds. See "Creating a Remittance Account".
3. Load the remittance fields file into the BRM database. You do this whether or not you add custom fields to the remittance fields file. This makes the fields available to the remittance specification. See "Loading the Remittance Fields File".
4. Create your remittance specifications. Each specification matches a remittance product and account with a set of remittance criteria. See "Defining Remittance Specifications ".
5. Load the remittance specification. This makes the specification available to BRM so it can begin collecting remittance information. See "Loading the Remittance Specifications".

Creating a Remittance Product

To create a remittance product, follow the procedure in Pricing Center Help. The following steps are specific to a remittance product:

1. In the Product Creation Wizard or in the General Product Info tab, select **/account** instead of a service in the **Applies To** field. This indicates that the remittance product is not connected with a service.
2. In the Event Map, select **Remittance Event** for **Event**.
3. In the Event Map, select one of the following metrics for **Measured By**: Number, Usage Time, Usage Size, and Amount. See Table 21-2.

Table 21-2 Examples of Metrics

Metric	Use To	Example
Number	Calculate remittance for a given event type based on a flat fee per occurrence.	You want to remit $5 for each cycle forward event.
Usage Time	Calculate remittance based on a flat fee for the duration of an event.	You want to remit $1 for each hour of Internet usage.
Usage Size	Calculate remittance based on the size of the event.	You want to remit $1 for each 5 megabytes (MB) of storage space a customer uses each month.
Amount	Calculate remittance based on a percentage of the rated dollar amount.	You want to remit 25% of a customer's total monthly fees for Internet usage to the customer's service provider.

Note

- Do not use the metric **Occurrence**. Pricing Center enables you to use it in your remittance product, but this metric will not work with remittance.
- You can also create custom metrics. See "Configuring Ratable Usage Metrics (RUMs)" in *PDC Creating Product Offerings*.

4. In Rate Plan Properties, specify the balance impact as follows:
   - If Number, Usage Time, or Usage Size is the metric, specify a negative value. For example, if the metric is Number, the balance impact might be -5 US dollars.

     The number is negative because you want BRM to credit the account that owns this product.
   - If Amount is the metric, specify a negative value that represents a percentage. For example, -.05 represents -5%.

Note

- Pricing Center lets you use positive values in the product balance impact. If you do this, BRM debits, rather than credits, the remittance account.
- Pricing Center does not validate your remittance product to ensure that you used a valid metric or entered a balance impact that makes sense.

Creating a Remittance Account

You create an account for each third party that you pay remittance. That account can only purchase plans and deals that contain remittance products.

Follow the normal procedure for creating accounts in Customer Center. See the discussion about creating a consumer or business account in Customer Center Help.

Use Invoice for the payment method. The invoices will show a negative balance due for remittance accounts.

Most other payment methods do not make sense for remittance. In particular, using credit card or direct debit as the payment method causes errors when you run the remittance utility, **pin_remittance**.

Loading the Remittance Fields File

The remittance fields file makes fields from storable classes available for setting up remittance criteria. You must load this file into BRM before you define your remittance specifications.

This file contains default fields you can use to define specifications that cover many common remittance scenarios. Custom fields can also be added. For more information, see "About Defining Remittance Specifications" or the description in the remittance fields file, **pin_remittance_flds**.

To load the remittance fields file:

1. Go to a directory with a valid configuration file.

   Typically, you go to the directory that contains the remittance fields file: **/sys/data/pricing/example**. The "load_pin_remittance_flds" utility uses the configuration file for information on how to connect to the BRM database. See "Connecting BRM Utilities" in *BRM System Administrator's Guide*.

2. Enter this command:

   ```
   BRM_home/bin/load_pin_remittance_flds file_name
   ```

   where *BRM_home* is the directory where you installed BRM components, and *file_name* is the name and path of the **pin_remittance_flds** file.

   You do not need to specify a file name if you use the default **pin_remittance_flds** file and you run the command from the same directory where the file resides.

To verify that the **pin_remittance_flds** file was loaded, you can display the **/config/remittance_flds** object by using the Object Browser, or use the **robj** command with the **testnap** utility.

For general instructions on using **testnap**, see "Using the testnap Utility to Test BRM" in *BRM Developer's Guide*. For information about reading an object and writing its contents to a file, see "Reading an Object and Writing Its Contents to a File" in *BRM Developer's Guide*.

Defining Remittance Specifications

You define the conditions that cause BRM to pay funds to a third party in a remittance specification. For more information, see "About Defining Remittance Specifications".

To create a remittance specification:

1. Open the remittance specification file (*BRM_home*/**sys/data/pricing/example/pin_remittance_spec**) in any text editor.

   The next steps refer to the following example of a simple specification:

   ```
   ACCOUNT_BEGIN

      remittance_account_number  0.0.0.1-9617
      remittance_type            B
      remittance_product_name    Product 6a - Flat Fee Remittance

      CRITERIA_BEGIN
         field   service_type    = /service/ip
         field   product_name    = Product 1a - Internet Access
         field   event_type      = /event/session/dialup
      CRITERIA_END

   ACCOUNT_END
   ```

2. Enter ACCOUNT_BEGIN to start a new specification.
3. Enter the number of the account that receives the remittance. For example:

   ```
   remittance_account_number  0.0.0.1-9617
   ```

4. Enter the remittance type, which is one of three values:
   - **B**: Billed. BRM does not credit the remittance account until the event that triggers the remittance has been billed.
   - **P**: Paid. BRM does not credit the remittance account until the event that triggers the remittance has been paid and the corresponding bill item closed.

     A bill item is closed for a BRM-initiated payment when you run **pin_collect**. You can also close a bill item when you transfer amounts between bill items through the Bill Details panel in Customer Center. See "pin_collect" in *BRM Configuring and Collecting Payments*.
   - **U**: Unbilled. BRM credits the remittance account whether or not the event that triggers the remittance has been billed or paid.

   For example:

   ```
   remittance_type            B
   ```

5. Enter the remittance product name. BRM uses this product to determine the remittance rate. The remittance account must own the product you specify. For example:

   ```
   remittance_product_name    Product 6a - Flat Fee Remittance
   ```

6. Enter CRITERIA_BEGIN to start the criteria section of the specification.
7. Enter one or more remittance criteria. These criteria are a series of statements. Each statement includes a field from the remittance fields file, an operator, and a value. For example:

   ```
   field   event_type    = /event/session/dialup
   ```

In this statement:

- **field** identifies the item that follows (service_type) as a field from the remittance fields file. It must start every line within the list of criteria.
- **event_type** is a field from the remittance fields file. Typically, you use at least three default fields as criteria: **event_type**, **service_type**, and **product_name**.

  > **Note**
  >
  > Whenever the product name changes, you must update the **pin_remittance_spec** file and reload it into the database.

  You can also use the default field **profile_name** or a custom field. For information on using custom fields, see "About Adding Custom Remittance Criteria".

  You must include the **event_type** field as one of your criteria.
- **=** (equal sign) is the operator. This is the only valid operator for **service_type**, **product_name**, **event_type**, and **profile_name**. For a list of operators you can use with other fields, see the **pin_remittance_spec** file.
- **/service/ip** is the value for **service_type**.

8. Enter CRITERIA_END at the end of the criteria.
9. Enter ACCOUNT_END at the end of the specification.

> **Note**
>
> If you want the same remittance account to receive payment for additional remittance products, then create a separate remittance specification for each product. You cannot use the same combination of remittance account and remittance product in more than one specification.

For more details on creating remittance specifications, see the instructions in the **pin_remittance_spec** file.

Loading the Remittance Specifications

Load the remittance specifications into your database by following one of these procedures:

- For single-schema systems, see "Loading Remittance Specifications on Single-Schema Systems".
- For multischema systems, see "Loading Remittance Specifications on Multischema Systems".

Loading Remittance Specifications on Single-Schema Systems

Use the utility "load_pin_remittance_spec" to load the remittance specification file into BRM:

1. Go to a directory that contains a valid configuration file.

   Typically, you go to the directory that contains the remittance criteria file: *BRM_home***/sys/data/pricing/example**. The **load_pin_remittance_spec** utility uses information in the configuration file to connect to the BRM database. See "Connecting BRM Utilities" in *BRM System Administrator's Guide*.

2. Enter this command:

   ```
   BRM_home/bin/load_pin_remittance_spec file_name
   ```

   where *file_name* is the name and path of the **pin_remittance_flds** file.

   You do not need to specify a file name if you use the default file name **pin_remittance_spec** and you run the command from the directory in which the file resides.

3. Stop and restart the Connection Manager (CM). See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

   Note

   You must follow this procedure every time you change the remittance specification file.

To verify that the **pin_remittance_spec** file was loaded, display the **/config/remittance_spec** object by using the Object Browser, or use the **robj** command with the **testnap** utility.

For general instructions on using **testnap**, see "Using the testnap Utility to Test BRM" in *BRM Developer's Guide*. For information about reading an object and writing its contents to a file, see "Reading an Object and Writing Its Contents to a File" in *BRM Developer's Guide*.

Loading Remittance Specifications on Multischema Systems

To load remittance specifications on a multischema system, perform the following procedure on your primary BRM installation system:

Note

You must follow this procedure every time you change the remittance specification file.

1. Combine the remittance specifications for all database schemas into a consolidated remittance specification file.
2. Go to a directory that contains a valid configuration file (**pin.conf**) for connecting to your BRM database schemas. See "Connecting BRM Utilities" in *BRM System Administrator's Guide*.
3. Run the **load_pin_remittance_spec** utility by entering this command:

   ```
   BRM_home/bin/load_pin_remittance_spec file_name
   ```

   where *file_name* is the name of your consolidated remittance specification file. For information, see "load_pin_remittance_spec".
4. Stop and restart the CM. See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.
5. Stop and restart the CM and all primary and secondary Data Managers (DMs). See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide.*

6. (Optional) To verify that the **pin_remittance_spec** file was properly loaded, display the **/config/remittance_spec** object by using the Object Browser or the **robj** command with the **testnap** utility.

   For general instructions on using **testnap**, see "Using the testnap Utility to Test BRM" in *BRM Developer's Guide*. For information about reading an object and writing its contents to a file, see "Reading an Object and Writing Its Contents to a File" in *BRM Developer's Guide*.

Running Remittance

Before running remittance, you should run billing for all non-remittance accounts. This ensures that BRM will calculate all remittance that depends on events being billed or paid. For more information, see "Running Billing Scripts".

Use the following steps each time you run remittance. Each step includes a link to a detailed procedure:

1. Calculate remittance. Then run billing and invoicing for the remittance accounts. See "Calculating Remittance ".
2. Run a BRM report that summarizes the amount due to each remittance account. See "Creating Remittance Reports".
3. Change the balance of each remittance account to reflect payments you made. See "Changing the Balance of a Remittance Account".

When you run remittance, you can use the **pin_remittance** utility **-b** parameter to choose whether to trigger billing of remittance accounts before calculating remittance. By default, the **pin_remit_month** script runs the **pin_remittance** utility with the **-b** parameter which ensures that remittance is calculated before the remittance account is billed.

If you do not use the **-b** parameter, remittance owed to the account in the current billing cycle is not credited to it until the next billing cycle.

Calculating Remittance

Use the **pin_remittance** utility to calculate the amount you must pay to each third party. You can run remittance as part of a monthly remittance script or you can run it separately.

For information on how BRM calculates remittance, see "About Calculating Remittance".

Running the Monthly Remittance Script

To calculate remittance:

1. Run the daily, weekly, and monthly billing scripts.

   By default, these scripts run billing on non-remittance accounts only. You should keep this default. You want to bill remittance accounts after calculating remittance, so the bills and invoices for these accounts are up-to-date.

   For information on running these scripts, see "Running Billing Scripts".
2. Run the monthly remittance script:

   ```
   pin_remit_month
   ```

   This script does the following:

- Runs the remittance utility, **pin_remittance**. For more information on this utility, see "Running the Remittance Utility Separately".
- Runs billing for remittance accounts.
- Runs invoicing for remittance accounts.

> **Note**
>
> You can run **pin_remit_month** at any time interval that is appropriate for your business. Running it monthly is one common approach.

Running the Remittance Utility Separately

To run the utilities in **pin_remit_month** separately:

1. Change to a directory with a valid configuration file. See "Connecting BRM Utilities" in *BRM System Administrator's Guide*.
2. Run the remittance utility:

   ```
   pin_remittance
   ```

   By default, **pin_remittance** does the following:

   - Calculates remittance for all remittance accounts.
   - Includes events that occurred up to midnight the previous day in its calculations.
   - Creates a remittance report in a text file. See "Creating Remittance Reports".

   For information on changing these defaults and on the utility's syntax, see "pin_remittance".

   The remittance utility is located in *BRM_home***/bin**.
3. Run the billing utility on inactive, closed, and active remittance accounts:

   ```
   pin_bill_accts -inactive -remit only
   pin_bill_accts -close -remit only
   pin_bill_accts -active -remit only
   ```

   For more information, see "Billing Accounts By Using the pin_bill_accts Utility" or "pin_bill_accts".
4. Run the invoice utility:

   ```
   pin_inv_accts -pay_type 10001
   ```

   For more information, see "Generating Invoices" in *BRM Designing and Generating Invoices*.

Creating Remittance Reports

You can get two reports for remittance:

- The **pin_remittance** utility creates a report that lists the amount remitted to each account each time you run the utility. The report is in a text file named **rem_***date***.rep**, where *date* is the end date for which remittance events are included in the calculation. By default, the end date is the current date.

  Table 21-3 contains an example of this report:

Table 21-3 Remittance Report Example

Acct No.	Start Date	End Date	Amount Remitted
0.0.0.1-9929	06/07/2020	08/13/2020	-676.2
0.0.0.1-10057	06/07/2020	08/13/2020	-9382.25
0.0.0.1-10185	06/07/2020	08/13/2020	0

Note

The negative values in the report represent balances owed to third parties.

You use this report to review the amounts that **pin_remittance** calculated and to verify that the information is correct.

- You can create a report that provides a summary of remittance due to each account for your payables department. You can generate this report for different time periods, account numbers, states, countries, and item types. For more information, see "pin_remittance".

Note

In a multischema system, the Remittance report is accurate only when each service provider account's associated remittance objects, remittance events, content connector events, and user accounts are in the same database schema. If they are not all in the same schema, some data is not included in the reports.

Changing the Balance of a Remittance Account

When you send funds to a remittance account, you use an adjustment in BRM to change the account's balance. For example, if the account shows a balance of -50, and you pay $50 to the third party, you must create an adjustment of $50 to change the balance to 0.

For information on adjustments, see "Making Adjustments " in *BRM Managing Accounts Receivable*.

Using Remittance with Multiple Database Schemas

If you have multiple BRM database schemas, you must run the remittance utility (**pin_remittance**) for each schema. You can do this in either of these ways:

- On one schema at a time, using one instance of the remittance utility. See "Running Remittance on One Schema at a Time".
- On multiple schemas simultaneously, using multiple instances of the remittance utility. See "Running Remittance on Multiple Schemas Simultaneously".

Depending on your setup, a single event can contribute remittance to more than one remittance account. In a multischema environment, those remittance accounts can be in different schemas. All remittance account balances are updated only when you run the remittance utility for all schemas.

Running Remittance on One Schema at a Time

Running the remittance utility on multiple database schemas one at a time requires that you edit the remittance utility configuration file every time you run the remittance utility. Perform the following procedure *before* you run remittance:

1. Open the remittance utility configuration file (*BRM_home*/**apps/pin_remit/pin.conf**).
2. Change the value of the **userid** entry to the schema against which you want to run remittance.

   For example, to run remittance on schema number **0.0.0.2**, change the **userid** entry as follows:

   ```
   - - userid 0.0.0.2 /service/pcm_client 1
   ```
3. Change the value of the **login_name** entry to an account that resides in the schema against which you want to run remittance.

   For example, to run remittance using the **root.0.0.0.2** account, change the **login_name** entry as follows:

   ```
   - nap login_name root.0.0.0.2
   ```
4. Save and close the file.
5. Run the remittance utility. See "Running Remittance".

Running Remittance on Multiple Schemas Simultaneously

Running remittance on multiple database schemas simultaneously requires that you create parallel instances of the remittance utility configuration file, each of which is configured for a particular schema. Then, you run all instances of your remittance utility.

1. For each schema you want to run remittance on, create a subdirectory in *BRM_home*/**apps/pin_remit**.

   For example, *BRM_home*/**apps/pin_remit/db1** for schema 1, *BRM_home*/**apps/pin_remit/db2** for schema 2, and so on.
2. Copy the *BRM_home*/**apps/pin_remit/pin.conf** file into each new subdirectory.
3. In each subdirectory, do the following:
   a. Open the **pin.conf** file.
   b. Change the schema number in the **login_name** entry to an account that resides in the schema against which you want to run remittance.

      For example, to run remittance against schema number **0.0.0.2**, change the **login_name** entry as follows:

      ```
      - nap login_name root.0.0.0.2
      ```
   c. Save and close the file.
4. Run the remittance utility from the new subdirectories. See "Running Remittance".

Improving Remittance Performance

If remittance-related events occur frequently in your BRM system, it can affect your system's performance. You can improve performance by increasing the time interval for refreshing the status of remittance accounts and products.

BRM caches remittance account-product status information and refreshes the information based on the time interval specified in the CM configuration file.

By default, this interval is set to 300 seconds (five minutes). If the status of an account or product changes, BRM does not get the status change for calculating remittance until the next interval. For more information on what happens when BRM calculates remittance, see "About Calculating Remittance".

To change the time interval for refreshing the status of remittance accounts and products:

1. Open the CM configuration file (*BRM_home***/sys/cm/pin.conf**).
2. Change the value of the **remit_cache_refresh_interval** entry:

   ```
   - fm_remit remit_cache_refresh_interval 300
   ```

   The interval value is in seconds, with a default of 300 seconds. You can change it as follows:

   - To improve remittance performance, increase the interval to refresh the status of accounts and products less frequently. The longer the interval, the more you must increase it to get equivalent performance improvements.

     For example, increasing the interval from 5 to 10 gives you a much greater performance improvement than increasing it from 300 to 305.
   - To refresh account and product status information more frequently, reduce the interval. This could affect your BRM system's performance.
   - If you want BRM to be immediately aware of status changes, comment out this entry by adding a # symbol at the beginning of the line. If you do this, BRM reads the status information from the database each time an event matches your remittance criteria.
3. Save and close the file.
4. Stop and restart the CM. See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

Using Remittance for Sales Commissions

You can use the BRM remittance feature to pay commissions to salespeople, but this requires customizing remittance. One approach is to do the following:

1. For each new subscriber account, have a programmer create a customer profile and include the name of the salesperson in that profile. For information on creating and using profiles, see "About Storing Customer Profile Information" in *BRM Developer's Guide*.
2. Create a custom field in the remittance fields file, for example, **profile_value**. For more information, see "Defining Custom Remittance Fields" or the description in the **pin_remittance_flds** file.
3. Create a separate remittance specification for each salesperson. For each specification, include the following criteria:
   - **profile_name**: This is a default remittance field. Set it to the name of the profile that contains the salesperson's name.

     You must include profile_name as one of the criteria if you are also using an attribute of a profile as one of your criteria.
   - **profile_value**: Set the custom remittance field you created in step 2 to the salesperson's name.

Example of Setting Up a Remittance Specification

This is an example of how to set up a simple remittance specification. In this example, the remittance account receives 5% of purchase and cycle forward fees from subscribers to a specific product.

1. In Pricing Center, create a product with the settings shown in Table 21-4.

Table 21-4 Example Settings for a Remittance Product

Field	**Value**
Name	Remittance Product 1
Description	Credit to remittance accounts 5% of the applicable rated purchase and cycle forward events paid by subscribers to a particular Internet access product.
Product Type	Subscription
Applies To	Account
Event	Remittance Event
Measured By	Amount
Rate Plan Structure	Single Rate Plan
Balance Impact:	None
Resource	US Dollar
Scaled Amount	-.05
Units	None

2. In Pricing Center, create a deal and plan, as shown in Table 21-5.

Table 21-5 Example Deal and Plan

Field	**Deal**	**Plan**
Name	Remittance Deal 1	Remittance Plan 1
Include	Remittance Product 1	Remittance Deal 1

3. In Pricing Center, add Remittance Plan 1 to the CSR-new plan list and commit the plan list to your BRM database.
4. In Customer Center, create an account that includes the settings shown in Table 21-6.

Table 21-6 Settings for Account

Field	**Value**
Plan	Remittance Plan
Name	Service Provider 1 (representing the name and contact information for a service provider you are sharing revenues with)
Payment Method	Invoice

This account will receive 5% of rated events as defined in the product Remittance Product 1.

5. Load the default remittance field file into BRM. Go to *BRM_home***/data/pricing/examples** and enter:

   ```
   load_pin_remittance_flds pin_remittance_flds
   ```

6. In a text editor, open the remittance specification file, *BRM_home***/sys/data/pricing/example/pin_remittance_spec**.
7. Add the following to the end of the file:

   ```
   ACCOUNT_BEGIN

      remittance_account_number  0.0.0.1-8422
      remittance_type            B
      remittance_product_name    Remittance Product 1

      CRITERIA_BEGIN
         field   service_type    = /service/ip
         field   product_name    = Product 1a - Internet Access
         field   event_type      = /event/billing/product/fee/purchase
      CRITERIA_END

      CRITERIA_BEGIN
         field   service_type    = /service/ip
         field   product_name    = Product 1a - Internet Access
         field   event_type      = /event/billing/product/fee/cycle/
   cycle_forward_monthly
      CRITERIA_END

   ACCOUNT_END
   ```

8. Load the remittance specification file into BRM. Go to *BRM_home***/data/pricing/examples** and enter:

   ```
   load_pin_remittance_spec pin_remittance_spec
   ```

9. Stop and restart the CM. See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

You have now implemented this remittance specification. When you run **pin_remittance**, BRM credits 5% of the purchase and cycle forward fees from the Internet Access product to the remittance account you created.

About Customizing Remittance

BRM provides a default set of fields in storable classes mapped to remittance fields, which you can use to specify remittance criteria.

You can customize remittances in the following ways:

- Define custom remittance criteria to specify how to remit funds to third parties. For more information, see "About Adding Custom Remittance Criteria".
- Calculate remittance based on custom ratable usage metrics (RUMs).

  For information on creating RUMs, see "Configuring Ratable Usage Metrics (RUMs)" in *PDC Creating Product Offerings*.

  For information on using custom RUMs to calculate remittance, see "Calculating Remittance Using Custom RUMs".

About Adding Custom Remittance Criteria

To add custom remittance criteria, you must specify the BRM fields used to define these criteria, and map each BRM field to a corresponding remittance field in the **pin_remittance_flds** file. BRM then validates each criterion based on the fields specified in the **pin_remittance_flds** file.

To add custom remittance criteria, you must perform these tasks:

1. Create a remittance product in Pricing Center. For more information, see "Creating a Remittance Product ".
2. Create a remittance account in Customer Center. For more information, see "Creating a Remittance Account".
3. Select the criteria for remittance. For example, you may want to add a new criteria called **company_name** in addition to the criteria already provided.
4. Define the remittance field for every criterion selected, if a BRM field is not already defined in the **pin_remittance_flds** file, and map each field to the corresponding remittance criterion.
5. In the **pin_remittance_spec** file, specify the criteria that determine which third party receives remittance and the product BRM uses to calculate remittance.

   The **pin_remittance** utility calculates the remittance amount for all events that meet the criteria you define in this file. See "pin_remittance".

Defining Custom Remittance Fields

The **pin_remittance_flds** file contains default fields that help you calculate remittance. You must load this file into the BRM system to define remittance specifications. For more information, see "Loading the Remittance Fields File".

You can add more fields to this file to define custom criteria in the **pin_remittance_spec** file.

Each field in the remittance fields file makes an attribute of the BRM storable class available for defining remittance criteria in the **pin_remittance_spec** file. To specify custom fields in this file, you must know about BRM fields and storable classes. For more information, see "About Storable Classes and Storable Objects" in *BRM Developer's Guide*.

A custom remittance field definition contains four columns separated by one or more spaces:

```
<remittance field name> <base class> <substruct> <BRM field>
```

Remittance Field Name

This name is the field identifier. Provide this name when you specify remittance criteria in the remittance specification file.

Precede the name with the word “field" followed by a space. For example, **"field origination_gw"**.

Do not use blank spaces within the name itself. You cannot specify a name already in use for one of the reserved attributes.

Base Class of the Attribute

BRM supports attributes for these base storable classes:

- **EVENT**

- **ACCOUNT**
- **PROFILE**

You must always specify the base class for an attribute.

Substruct Name

The name of the substruct within which the field is located. This must be a valid substruct name as specified in the BRM data dictionary. For example, for a telco call event, the PIN_FLD_CALLING_FROM field is contained within the substruct PIN_FLD_TELCO_INFO. If there are no substructs, specify **NONE**. You cannot use arrays or fields within arrays.

Attribute Name

The name of the attribute specified in the BRM data dictionary. For example, for the origination gateway of a telco call event, specify the PIN_FLD_CALLING_FROM field.

This example shows the remittance field definition for other call origination:

```
field origination    EVENT    PIN_FLD_TELCO_INFO    PIN_FLD_CALLING FROM
```

- The first column is the remittance field name, which identifies the field defined in the **pin_remittance_spec** file.
- The second column is the base class of the attribute. You can specify only EVENT, ACCOUNT, or PROFILE.

  The third column is the name of the substruct in the BRM database that contains the field.
- If the attribute is contained within a substruct name, specify the BRM name of the substruct; otherwise specify NONE.
- The fourth column is the BRM field name for the attribute.

To add a **brand_name** field to indicate the brand, add the following line beneath the reserved fields:

```
field    brand_name    ACCOUNT NONE    PIN_FLD_GL_SEGMENT
```

Note

Do not change or delete the reserved attributes in the remittance fields file. However, you can edit this file to add fields.

This example shows you how to add custom remittance fields:

```
service_type             RESERVED
event_type               RESERVED
product_name             RESERVED
profile_name             RESERVED
field Origination        EVENT      PIN_FLD_TELCO_INFO         PIN_FLD_CALLING FROM
field Destination        EVENT      PIN_FLD_TELCO_INFO         PIN_FLD_CALLED TO
field Brand_name         ACCOUNT    NONE                  PIN_FLD_GL_SEGMENT
field Profile_value      PROFILE    PIN_FLD_LDAP_INFO     PIN_FLD_DN
```

Specifying Custom Remittance Criteria

In the remittance specification file, specify the custom criteria you want to use in remittance calculation and then define the corresponding fields in the remittance fields file.

For syntax and instructions on specifying single and multiple sets of criteria for an account, see the **pin_remittance_spec** file.

Specifying Remittance Criteria for Sales Commissions

Use multiple criteria specification to remit sales commissions to salespersons. For example, to remit $1 to every salesperson for each cycle of the salesperson's IP accounts, do the following:

1. For each salesperson, create a remittance account (remittance product and plan can be shared).
2. When you create customer accounts, use the salesperson's name in the profile.
3. Create custom criteria for each salesperson.

This example shows how to use multiple criteria specification to remit sales commissions:

```
ACCOUNT_BEGIN
remittance_account_number0.0.0.1-10001
#sales person A's remittance account
remittance_typeP
remittance_product_name   SalesCommissionProduct
CRITERIA_BEGIN
field service_type=/service/ip
field event_type =/event/billing/product/fee/cycle/cycleforward_monthly
fieldprofile_name= SalesA
CRITERIA_END
ACCOUNT_END
ACCOUNT_BEGIN
remittance_account_number0.0.0.1-10002
#salesperson B's remittance account
remittance_typeP
remittance_product_name   SalesCommissionProduct
CRITERIA_BEGIN
fieldservice_type=/service/ip
fieldevent_type=/event/billing/product/fee/cycle/cycle_forward
monthlyfieldprofile_name= SalesB
CRITERIA_END
ACCOUNT_END
ACCOUNT_BEGIN
remittance_account_number0.0.0.1-10003
#salesperson C's remittance account
remittance_typeP
remittance_product_name   SalesCommissionProduct
CRITERIA_BEGIN
fieldservice_type =/service/ip
fieldevent_type =/event/billing/product/fee/cycle/
cycle_forward_monthlyfieldprofile_name = SalesC
CRITERIA_END
ACCOUNT_END
```

For more information about using remittance for sales commissions, see "Using Remittance for Sales Commissions".

About Using Custom Ratable Usage Metrics to Calculate Remittance

Remittance products use the **/event/billing/remittance** event type. To calculate remittance for an event type, set its ratable usage metric (RUM) to the appropriate name. For a list of RUMs you can use to create a remittance product, see "Creating a Remittance Product ".

BRM supplies a set of default ratable usage metrics (RUMs) for remittance products, but you can also calculate remittance based on custom RUMs.

For information on how to create and load custom RUMS into the BRM database, see "About Setting Up RUMs for Real-Time Rating" in *BRM Setting Up Pricing and Rating*.

Calculating Remittance Using Custom RUMs

The BRM system recognizes the ratable quantity only for default RUMs. To use a custom RUM, modify the PCM_OP_REMIT_POL_SPEC_QTY policy opcode. See "Customizing Remittance".

How Remittance Works

Use the following opcodes to manage remittance:

- PCM_OP_REMIT_GET_PROVIDER. See "Retrieving Remittance Accounts".
- PCM_OP_REMIT_REMIT_PROVIDER. See "Calculating the Remittance Amount".
- PCM_OP_REMIT_VALIDATE_SPEC_FLDS. See "Verifying the Remittance Specification File".
- PCM_OP_REMIT_POL_SPEC_QTY. See "Customizing Remittance".

Retrieving Remittance Accounts

Use PCM_OP_REMIT_GET_PROVIDER to retrieve a list of remittance accounts that are owed for a particular event.

PCM_OP_REMIT_GET_PROVIDER performs the following:

1. Determines whether an event meets any criteria that you specified in the **pin_remittance_spec** file. See "Defining Remittance Specifications ".
2. Retrieves the following remittance information from the event itself and the **/config/remittance_spec** object:
   - Event object POID
   - Item object POID
   - Remittance account object POID
   - Remittance product object POID
   - Quantity to rate for the remittance calculation

Note

If the event uses a custom RUM, PCM_OP_REMIT_GET_PROVIDER calls PCM_OP_REMIT_POL_SPEC_QTY to retrieve the quantity. See "Customizing Remittance".

3. Creates a **/remittance_info** object.
4. Returns the POID of the **/remittance_info** object.

Calculating the Remittance Amount

Use PCM_OP_REMIT_REMIT_PROVIDER to calculate the remittance amount owed to third-party companies. This opcode retrieves remittance data from **/remittance_info** objects and then calculates the total remittance amount owed to third parties.

This opcode is called directly by the **pin_remittance** utility. For more information, see "pin_remittance".

Verifying the Remittance Specification File

Use PCM_OP_REMIT_VALIDATE_SPEC_FLDS to validate remittance criteria before loading it into the BRM database. This opcode validates that the criteria fields you specify in the **pin_remittance_spec** file are defined in the **/config/remittance_flds** object.

When the **pin_remittance_spec** file passes the validation, PCM_OP_REMIT_VALIDATE_SPEC_FLDS returns the PIN_FLD_RESULT field set to PIN_RESULT_PASS. This notifies the calling application to load the specification data into the **/config/remittance_spec** object.

When the file fails validation, PCM_OP_REMIT_VALIDATE_SPEC_FLDS returns the PIN_FLD_RESULT field set to PIN_RESULT_FAIL. This notifies the calling application to stop the loading process.

This opcode is called directly by the **load_pin_remittance_spec** utility. For more information, see "load_pin_remittance_spec".

Customizing Remittance

Use the PCM_OP_REMIT_POL_SPEC_QTY policy opcode to retrieve the remittance quantity to rate for a custom RUM. However, you can modify this policy opcode to retrieve the remittance quantity for custom RUMs.

22

Billing Utilities

Learn about the syntax and commands for the Oracle Communications Billing and Revenue Management (BRM) billing utilities.

Topics in this document:

load_config_item_tags

Use this utility to load item tags into the BRM database when you create custom bill items. See "Creating Custom Bill Items".

Location

BRM_home/**bin**

Syntax

```
load_config_item_tags [-v] [-d] [-h] config_item_tags_file
```

Parameters

-v
Displays information about successful or failed processing as the utility runs.

> **Note**
>
> This parameter is always used 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:
> **load_config_item_tags** *any_other_parameter* **–v >** *filename*.**log**

-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.

-h
Displays the syntax and parameters for this utility.

config_item_tags_file
The name and location of the file that contains your item tag definitions. The sample **config_item_tags.xml** file is in the *BRM_home***/sys/data/pricing/example** directory.

Results

The **load_config_item_tags** utility notifies you when it successfully creates the **/config/item_tags** object.

If the **load_config_item_tags** utility does not notify you that it was successful, look in the utility log file (**default.pinlog**) to find any errors. The log file is either in the directory from which the utility was started or in a directory specified in the configuration file.

load_config_item_types

Use this utility to load bill item tag-to-bill item type mappings into the BRM database when creating custom bill items. See "Creating Custom Bill Items".

Location

*BRM_home***/bin**

Syntax

```
load_config_item_types [-v] [-d] [-h] config_item_types_file
```

Parameters

-v
Displays information about successful or failed processing as the utility runs.

Note

This parameter is always used 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:
load_config_item_types *any_other_parameter* **–v >** *filename*.**log**

-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.

-h
Displays the syntax and parameters for this utility.

config_item_types_file
The name and location of the file that contains your custom item definitions. You can use the sample file, *BRM_home*/**sys/data/pricing/example/config_item_types.xml**.

Results

The **load_config_item_types** utility notifies you when it successfully creates the **/config/item_types** object.

If the **load_config_item_types** utility does not notify you that it was successful, look in the utility log file (**default.pinlog**) to find any errors. The log file is either in the directory from which the utility was started, or in a directory specified in the configuration file.

load_pin_bill_suppression

Use this utility to load bill suppression information for customer segments into the BRM database. See the following topics:

- Associating Bill Suppression Information with Customer Segments
- Editing the Bill Suppression Configuration File

Location

BRM_home/**bin**

Syntax

```
load_pin_bill_suppression [-t] [-v] [-d] [-h] filename
```

Parameters

-t
Runs the utility in test mode to validate the XML file against its schema definition (see "Validating Your Bill Suppression Configuration File Edits"). This option does *not* load data into the **/config/suppression** object or overwrite any existing data in the object.

> **Note**
>
> To avoid load errors based on XML content problems, run the utility with this option *before* loading data into the object.

-v
Displays information about successful or failed processing as the utility runs.

> **Note**
>
> This parameter is always used 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:
> **load_pin_bill_suppression** *any_other_parameter* **–v >** *filename***.log**

-d
Creates a log file for debugging purposes. If no log file name is specified in the utility's **pin.conf** file, name the file **default.pinlog**. Use this parameter for debugging when the utility seems to have run with no errors but the data has not been loaded into the database.

-h
Displays the syntax and parameters for this utility.

filename
The name and location of the bill suppression configuration file. The default file is *BRM_home***/sys/data/config/pin_bill_suppression.xml**, but the utility can take any XML file name as a parameter if the file's contents conform to the appropriate schema definition. See "Validating Your Bill Suppression Configuration File Edits".
If you run the command in a different directory from where *filename* is located, you must include the entire path for the file. In addition, *filename* must be in the same directory as the default *BRM_home***/sys/data/config/business_configuration.xsd** file.

Results

This utility notifies you only if it encounters errors. Look in the **default.pinlog** file for errors. This file is either in the directory from which the utility was started or in a directory specified in the utility configuration file.

To verify that the bill suppression information was loaded, display the **/config/suppression** object by using one of the following features:

- Object Browser
- **robj** command with the **testnap** utility

For information about reading an object and writing its contents to a file, see "Reading an Object and Writing Its Contents to a File" in *BRM Developer's Guide*.

Note

You must stop and restart the Connection Manager (CM) to make new bill suppression data available.

load_pin_billing_segment

Use this utility to load billing segment definitions into the BRM database. See "Setting Up Billing Segments".

Location

BRM_home/**bin**

Syntax

```
load_pin_billing_segment [-t] [-v] [-d] [-h] filename
```

Parameters

-t
Runs the utility in test mode to validate the XML file against its schema definition (see "Validating Your Billing Segment Configuration File Edits"). This option does *not* load data into the **/config/billing_segment** object or overwrite any existing data in the object.

Note

To avoid load errors based on XML content problems, run the utility with this option *before* loading data into the object.

-v
Displays information about successful or failed processing as the utility runs.

Note

This parameter is always used 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:
load_pin_billing_segment *any_other_parameter* **–v >** *filename*.**log**

-d
Creates a log file for debugging purposes. If no log file name is specified in the utility's **pin.conf** file, name the file **default.pinlog**. Use this parameter for debugging when the utility seems to have run with no errors but the data has not been loaded into the database.

-h
Displays the syntax and parameters for this utility.

filename
The name and location of the billing segment configuration file. The default file is *BRM_home***/sys/data/config/pin_billing_segment.xml**, but the utility can take any XML file name as a parameter if the file's contents conform to the appropriate schema definition. See "Validating Your Billing Segment Configuration File Edits".
If you run the command in a different directory from where *filename* is located, you must include the entire path for the file. In addition, *filename* must be in the same directory as the default *BRM_home***/sys/data/config/business_configuration.xsd** file.

Results

This utility notifies you only if it encounters errors. Look in the **default.pinlog** file for errors. This file is either in the directory from which the utility was started or in a directory specified in the utility configuration file.

To verify that the billing segment definitions were loaded, display the **/config/billing_segment** object by using one of the following features:

- Object Browser
- **robj** command with the **testnap** utility

For information about reading an object and writing its contents to a file, see "Reading an Object and Writing Its Contents to a File" in *BRM Developer's Guide*.

load_pin_calendar

Use this utility to load billing calendars into the BRM database. See "Editing the Billing Calendar Configuration File ".

Location

*BRM_home***/bin**

Syntax

```
load_pin_calendar [-t] [-d] [-v] [-h] filename
```

Parameters

-t
Runs the utility in test mode to validate the XML file against its schema definition (see "Validating Your Billing Calendar Configuration File Edits"). This option does *not* load data into **/config/calendar** objects or overwrite any existing data in the objects.

> **Note**
>
> To avoid load errors based on XML content problems, run the utility with this option *before* loading data into the object.

-d
Creates a log file for debugging purposes. Use this parameter for debugging when the utility seems 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 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:
> **load_pin_calendar** *any_other_parameter* **–v >** *filename*.**log**

-h
Displays the syntax and parameters for this utility.

filename
The name and location of the billing calendar configuration file. The default file is *BRM_home*/**sys/data/config/pin_calendar.xml**, but the utility can take any XML file name as a parameter if the file's contents conform to the appropriate schema definition. See "Validating Your Billing Calendar Configuration File Edits".
If you run the command in a different directory from where *filename* is located, you must include the entire path for the file. In addition, *filename* must be in the same directory as the default *BRM_home*/**sys/data/config/business_configuration.xsd** file.

Results

If the utility does not notify you that it was successful, look in the **default.pinlog** file to find any errors. This file is either in the directory from which the utility was started or in a directory specified in the utility configuration file.

load_pin_payment_term

Use this utility to load payment terms into the BRM database. See "Editing the Payment Terms Configuration File ".

Location

BRM_home/**bin**

Syntax

```
load_pin_payment_term [-t] [-h] [-v] [-d] filename
```

Parameters

-t
Runs the utility in test mode to validate the XML file against its schema definition. This option does *not* load data into the **/config/payment_term** object or overwrite any existing data in the object.

> **Note**
>
> To avoid load errors based on XML content problems, run the utility with this option *before* loading data into the object.

-h
Displays the syntax and parameters for this utility.

-d
Creates a log file for debugging purposes. Use this parameter for debugging when the utility seems 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 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:
> **load_pin_payment_term** *any_other_parameter* **–v >** *filename*.**log**

filename
The name and location of the payment terms configuration file. The default file is *BRM_home***/sys/data/config/pin_payment_term.xml**, but the utility can take any XML file name as a parameter if the file's contents conform to the appropriate schema definition.
If you run the command in a different directory from where *filename* is located, you must include the entire path for the file. In addition, *filename* must be in the same directory as the default *BRM_home***/sys/data/config/business_configuration.xsd** file.

Results

If the utility does not notify you that it was successful, look in the **default.pinlog** file to find any errors. This file is either in the directory from which the utility was started or in a directory specified in the utility configuration file.

load_pin_remittance_flds

Use this utility to load remittance field definitions into the BRM database. A remittance field corresponds to an attribute of a storable class and is used to define criteria in remittance profiles. Loading these field definitions is a prerequisite to using the BRM remittance feature.

You define remittance fields in the *BRM_home***/sys/data/pricing/example/pin_remittance_flds** file. Even if you do not modify the default version of this file, you must load this file before you can define a remittance specification file.

To connect to the BRM database, the **load_pin_remittance_flds** utility needs a configuration file in the directory from which you run the utility. See "Connecting BRM Utilities".

> **Caution**
>
> When you run **load_pin_remittance_flds**, it overwrites remittance fields currently in the BRM database. If you are updating remittance fields, you cannot load only new fields. Ensure that you load a complete remittance fields file, including fields that have not changed.

Location

*BRM_home***/bin**

Syntax

```
load_pin_remittance_flds [-t] [-v] [-d] [-h] remittance_flds
```

Parameters

-t
Runs the utility in test mode. This option does *not* save the field definitions to the BRM database. You can use this parameter to verify if the file will load correctly.

-v
Displays information about successful or failed processing as the utility runs.

> **Note**
>
> This parameter is always used 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:
> **load_pin_remittance_flds** *any_other_parameter* **–v >** *filename***.log**

-d
Creates a log file for debugging purposes. If no log file name is specified in the utility's **pin.conf** file, the file is named **default.pinlog**. Use this parameter for debugging when the utility seems to have run with no errors but the data has not been loaded into the database.

-h
Displays the syntax and parameters for this utility.

remittance_flds
The name and location of the remittance fields file. The default is *BRM_home***/sys/data/pricing/example/pin_remittance_flds**.

Results

This utility notifies you only if it encounters errors. Look in the **default.pinlog** file for errors. This file is either in the directory from which the utility was started or in a directory specified in the utility configuration file.

load_pin_remittance_spec

Use this utility to load remittance specifications into the BRM database. In a remittance specification, you define the criteria that determine which third party receives remittance and which product BRM uses to calculate remittance. You must load this file to use the BRM remittance feature.

load_pin_remittance_spec checks the validity of products in the remittance specification.

You define remittance specifications in the *BRM_home***/sys/data/pricing/example/pin_remittance_spec** file, or another file that uses the same format.

Caution

When you run **load_pin_remittance_spec**, it overwrites remittance specifications in the BRM database. If you update remittance information, you cannot load only new remittance specifications. Ensure that you load your complete specification file, including unchanged specifications.

Location

BRM_home/**bin**

Syntax

```
load_pin_remittance_spec [-t] [-v] [-d] [-h] filename
```

Parameters

-t
Runs the utility in test mode. This option does *not* save the specification file to the BRM database. You can use this parameter to verify if the file will load correctly.

-v
Displays information about successful or failed processing as the utility runs.

Note

Use this parameter 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*.**lo**g with the name of the log file:
load_pin_remittance_spec *any_other_parameter* **–v >** *filename*.**log**

-d
Creates a log file for debugging purposes. If no log file name is specified in the utility's **pin.conf** file, name the file **default.pinlog**. Use this parameter for debugging when the utility seems to have run without errors, but the data has not been loaded into the database.

-h
Displays the syntax and parameters for this utility.

filename
The name and location of the remittance specification file. The default is *BRM_home*/**sys/data/pricing/example/pin_remittance_spec**.

Results

This utility notifies you only if it encounters errors. Look in the **default.pinlog** file for errors. This file is either in the directory from which the utility was started or in a directory specified in the utility configuration file.

Note

You must stop and restart the Connection Manager (CM) to make the new specification available.

pin_bill_accts

Use this utility as part of your daily billing to calculate the balance due for each account and to create a bill for the balance due. See "Billing Accounts By Using the pin_bill_accts Utility".

Location

BRM_home/**bin**

Syntax

```
pin_bill_accts  [-active|-inactive|-close]
                [-pay_type ID]
                [-end mm/dd/yy|numberOfDays]
                [-remit [include|exclude|only]]
                [-sponsorship|-discount]
                [-retry]
                [-cycle_charge_only] [-finalize_bill]
                [-preinvoice bill|accept|reject]
                [-file billing_run_config_file_name]
                [-from_file]
                [-split]
                [-test]
                [-verbose]
                [-help]
```

Parameters

-active | -inactive | -close
Specifies the status of the accounts to be billed.

-pay_type *ID*
Calculates the balance due for accounts with the specified payment method. You specify the payment method by using the payment method ID, such as 10007 for nonpaying (subordinate) accounts, as shown in Table 22-1.

Note

If you use **-pay_type 10007** Subordinate, you must run the **pin_bill_accts** utility twice. The first time you run the utility with this parameter to calculate the child account's balance, and the second time you run the utility without the parameter to roll up the balance due to the parent account and then bill the parent account.

This example creates bills by using **-pay_type 10007** for nonpaying (subordinate) accounts:

```
pin_bill_accts     -active      -pay_type 10007
pin_bill_accts     -active
```

Payment Method	ID
credit card	10003
debit card	10002
direct debit	10005
guest	10010
invoice	10001
prepaid	10000
SEPA	10018
subordinate	10007
undefined	0

-end *mm/dd/yy|numberOfDays*
Specifies the due date for accounts to be billed. You can specify a specific due date (for example, **-end 01/31/21** includes accounts with a due date on or before January 31, 2021) or you can specify the due date as number of days from the current date (for example, **-end 5** includes accounts with a due date on or before 5 days from the current date).

Note

The **end** time parameter cannot be greater than the system time. For example, if the current system date is 1/15/2005 and the **end** time specified is 1/31/2005, **pin_bill_accts** fails with “bad config:time" error message.

-remit
Specifies whether remittance accounts should be billed. Use one of these options with this parameter:

- **-remit include**: Include remittance accounts in billing.
- **-remit exclude**: Exclude remittance accounts from billing.
- **-remit only**: Bill only remittance accounts.

If you do not specify this parameter, remittance accounts are excluded from billing. For information on remittance, see "Remitting Funds to Third Parties".

-sponsorship
Specifies how charge sharing groups are billed. When you specify this option, the **pin_bill_accts** behavior is determined by the value of the **BillingFlowSponsorship** parameter, which can be one of the following:

- **0:** Bills owner and member accounts in any order.
- **1:** Bills owner accounts before member accounts.
- **2:** Bills member accounts before owner accounts.

Before using this option, ensure that **BillingFlowSponsorship** is set to the correct value. See "Configuring Billing for Groups and Hierarchies".

Note

You cannot use this parameter with the **-discount** parameter.

-discount
Specifies how discount sharing groups are billed. When you specify this option, **pin_bill_accts** behavior is determined by the value of the **BillingFlowDiscount** parameter, which can be one of the following:

- **0:** Bills owner and member accounts in any order.
- **1:** Bills owner accounts before member accounts.
- **2:** Bills member accounts before owner accounts.

Before using this option, ensure that **BillingFlowDiscount** is set to the correct value. See "Configuring Billing for Groups and Hierarchies".

Note

You cannot use this parameter with the **-sponsorship** parameter.

-retry
Runs billing for accounts that were previously not billed by **pin_bill_accts** due to some error. After the errors have been resolved, use this option to bill the failed accounts.
Use **pay_type 10007** with the **-retry** option to bill failed accounts that own nonpaying bill units first, and then run it without the **pay_type** option to bill all other failed accounts as follows:

```
pin_bill_accts -active -retry -pay_type 10007
pin_bill_accts -active -retry
```

-cycle_charge_only
Runs billing on accounts with **/billinfo** bill states **0** and **1**. All cycle fees and billing-time discounts are applied, but totals are not calculated, and the bill is not finalized. At the end of the billing run, the **/billinfo** bill state is set to **2**. See "About Bill Unit States and the pin_bill_accts Utility".

Note

After running **pin_bill_accts** with this parameter, use the **-finalize_bill** parameter to finalize the bill.

-finalize_bill
Runs billing on accounts with **/billinfo** bill state **2**. Totals are calculated, and items, bills, and so on are updated. At the end of the billing run, the **/billinfo** bill state is set to **0**. See "About Bill Unit States and the pin_bill_accts Utility".

Note

- Use this parameter to finalize a bill after running the **pin_bill_accts** utility with the **-cycle_charge_only** parameter.

-preinvoice
Runs a proforma invoice before the final billing of an account is run. Use this to generate, accept, or reject the proforma invoice. Use one of these options with this parameter:

- **-preinvoice bill**: Run a proforma invoice when billing or regenerate a proforma invoice.
- **-preinvoice accept**: Accept the proforma invoice.
- **-preinvoice reject**: Reject the proforma invoice.

-file *billing_run_config_file_name*
Specifies the name and location of a billing run configuration file. The default file is **pin_bill_run_control.xml** in the *BRM_home***/apps/pin_billd** directory.
The billing run configuration file can contain either a list of account and bill unit POIDs or a list of billing segments and accounting days of months (DOMs).

Note

- When you run **pin_bill_accts** with a billing run configuration file, do not run it as a **cron** job. If you do, depending on the restrictions in the configuration file, some bill units might never be billed.
- When running **pin_bill_accts** with the **-file** option, ensure that the accounts specified in the billing run configuration file reside on the same database schema where **pin_bill_accts** is run. If the file contains accounts from different database schemas, **pin_bill_accts** reports an error. See "Setting Up Billing in a Multischema Environment".

-from_file
Specifies to bill the accounts listed in the input TXT file. For information about the syntax of the input TXT file, see "Configuring the Text File for Proforma Invoices" in *Designing and Generating Invoices*.

-split
Generates detail revenue assurance data if you have enabled its collection in the billing utilities configuration file. (See "Enabling Billing Utilities to Generate Revenue Assurance Data" in *BRM Collecting Revenue Assurance Data*.) You can view the detailed data by generating Revenue Assurance Billing Detail report. The details are based on item types.

Note

If you specify both the **-split** and **-file** parameters and the input file for the **-file** parameter includes a list of accounts and bill units, the Revenue Assurance Billing Detail report does not segregate the data based on the billing segment and billing day of month (DOM).

For more information, see "About Collecting Revenue Assurance Data from Billing" in *BRM Collecting Revenue Assurance Data*.

-test
Tests the utility, but does not affect accounts. Use this parameter to see which accounts will be billed, without actually creating bills for the balances due.

-verbose
Displays information about successful or failed processing as the utility runs.

> **Note**
>
> This parameter is always used with other parameters and commands. It is not position dependent. For example, you can enter **-verbose** 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_bill_accts** *any_other_parameter* **–verbose >** *filename*.**log**

-help
Displays the syntax and parameters for this utility.

Results

If the **pin_bill_accts** utility does not notify you that it was successful, look in the utility log file (**default.pinlog**) to find any errors. The log file is either in the directory from which the utility was started, or in a directory specified in the configuration file.

When it is called internally by the **pin_bill_day** script, the **pin_bill_accts** utility logs error information in the **pin_mta.pinlog** file.

Error Handling

When the **pin_bill_accts** utility detects that the cycle fee processing was not completed for the **/billinfo** object, **pin_bill_accts** stops with an error and sets the PIN_FLD_BILLING_STATUS field of the **/billinfo** object to PIN_BILL_ERROR. In addition, it sets the PIN_FLD_BILLING_STATUS_FLAGS field of the **/billinfo** object to PIN_BILL_FLAGS_CF_NOT_APPLIED (bit value 0x1000).

> **Note**
>
> If any nonpaying child bill unit caused the failure, **pin_bill_accts** updates the billing statuses set in the PIN_FLD_BILLING_STATUS and PIN_FLD_BILLING_STATUS_FLAGS fields of the **/billinfo** object for both child and parent bill units.

pin_cust_convert_wholesale_hierarchy

Use this utility to convert one or more hierarchies to wholesale billing. See "Converting Existing Bill Unit Hierarchies to Wholesale Billing" for information about the composition of the input file. See "Configuring Wholesale Billing" for more information about wholesale billing.

Location

BRM_home/**bin**

Syntax

```
pin_cust_convert_wholesale_hierarchy -file convert_wholesale_input.xml
```

Parameters

-file *convert_wholesale_input.xml*
Specifies the name and location of an input file containing information about the hierarchies to be converted.
The input file can contain either a list of **/billinfo** POIDs or a list of **/BusinessProfile** POIDs for the hierarchies to be converted.

Results

Each thread of this application will call the PCM_OP_CUST_CONVERT_WHOLESALE_HIERARCHY opcode to convert the hierarchy for a parent account bill unit.

Error Handling

This application checks for multiple bill units on a single parent account, and if found, does not proceed for that account. Accounts with multiple parent bill units are not allowed in wholesale billing.

pin_cycle_fees

Use this utility to perform the following tasks:

- Charge cycle forward fees.
- Identify cycle forward fees that have reached the end of free billing periods. For example, if a customer signs up for one month of free service, the **pin_cycle_fees** utility finds when the free period is over and applies the cycle forward fee balance impact to the customer's account.
- Cancel charge offers with an expired pending cancellation date. For example, if a charge offer is set to cancel at a future date, the **pin_cycle_fees** utility cancels it.
- Bill charge offers with a delayed purchase start time.
- Split events for backdated operations. When the backdate window includes multiple billing cycles, the **pin_cycle_fees** utility splits the events by cycle and applies the charges separately for each cycle.

See "Using the pin_cycle_fees Utility".

Location

BRM_home/**bin**

Syntax

```
pin_cycle_fees   -regular_cycle_fees|-defer_cycle_fees|-defer_cancel|-defer_purchase
                 [-start mm/dd/yy|number_of_days]
                 [-end mm/dd/yy|number_of_days]
```

```
[-verbose] [-test] [-help]
[-file filename.xml | -f filename.txt]
```

Parameters

-regular_cycle_fees
Charges cycle forward fees.

> **Note**
>
> The **-regular_cycle_fees** parameter replaces the functionality of the **pin_cycle_forward** utility.

-defer_cycle_fees
Identifies and applies cycle forward fees that have reached the end of free billing periods.

-defer_cancel
Cancels expired charge offers.

-defer_purchase
Bills charge offers with a delayed purchase start time.

-start [*mm/dd/yy or yyyy|number_of_days*]
-end [*mm/dd/yy or yyyy|number_of_days*]
Start and end dates.
If a start date is specified, the entire day is included. The end date is automatically the current date if you do not specify a value for the **-end** parameter.
If an end date is specified, that entire day is included, ending at, but not including, the 0th (first) second of the next day (00:00:00 a.m.). The end date cannot be a future date.

-test
Tests the utility, but does not affect accounts. Use this parameter to see which accounts have reached the end of free billing without actually applying cycle forward fees or canceling the expired charge offers.

-verbose
Displays information about successful or failed processing as the utility runs.

> **Note**
>
> This parameter is always used with other parameters and commands. It is not position-dependent. For example, you can enter **-verbose** 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_cycle_fees** *any_other_parameter* **–verbose >** *filename*.**log**

-help
Displays the syntax and parameters for this utility.

-file *filename*.xml
Processes the accounts or services in the specified **.xml** file.

The utility uses a sample file named **PinBillRunControl.xml** file, which is present in the *BRM_home*/**apps/pin_billd** directory. The XML file must list the account POIDs to process in the following format:

```
<CycleFeeList>
     <Account>12345</Account>
</CycleFeeList>
```

When the **StagedBillingFeeProcessing** business parameter is set to **1**, **2**, or **3**, you can also specify to apply cycle fees to services. In the XML file, include a service POID with each account POID. For example:

```
<CycleFeeList>
     <Account>12345</Account>
     <Service>12345</Service>
</CycleFeeList>
```

For information about the **StagedBillingFeeProcessing** business parameter, see "Applying Cycle Forward Fees in Parallel" in *PDC Creating Product Offerings*.

-f *filename*.txt
Specifies to process the accounts in the **.txt** file containing the list of account POIDs in flist format. For example:

```
0     PIN_FLD_RESULTS       ARRAY [0] allocated 20, used      1
1     PIN_FLD_POID           POID [0] 0.0.0.1 /account 101359   0
0     PIN_FLD_RESULTS       ARRAY [1] allocated 20, used      1
1     PIN_FLD_POID           POID [0] 0.0.0.1 /account 98397    0
```

When the **StagedBillingFeeProcessing** business parameter is set to **1**, **2**, or **3**, you can also specify to apply cycle fees to services. In the TXT file, include a service POID with each account POID. For example:

```
0 PIN_FLD_RESULTS                 ARRAY [0] allocated 20, used 9
1   PIN_FLD_POID                   POID [0] 0.0.0.1 /account 180898 0
1   PIN_FLD_SERVICE_OBJ            POID [0] 0.0.0.1 /service/email 184226 0
0 PIN_FLD_RESULTS                 ARRAY [1] allocated 20, used 9
1   PIN_FLD_POID                   POID [0] 0.0.0.1 /account 203813 0
1   PIN_FLD_SERVICE_OBJ            POID [0] 0.0.0.1 /service/ip 201893 0
```

Results

If the **pin_cycle_fees** utility does not notify you of its success, look in the utility log file (**default.pinlog**) to find any errors. The log file is either in the directory from which the utility was started or in a directory specified in the configuration file.

When it is called internally by the **pin_bill_day** script, the **pin_cycle_fees** utility logs error information in the **pin_mta.pinlog** file.

Error Handling

When the **pin_cycle_fees** utility encounters an error in applying regular, deferred, deferred purchase, or deferred cancellation cycle fees, it sets the PIN_FLD_BILLING_STATUS billing status field of the **/billinfo** object to PIN_BILL_ERROR. In addition, it sets the appropriate bit of the PIN_FLD_BILLING_STATUS_FLAGS field of the **/billinfo** object as follows:

- For regular cycle fees: PIN_BILL_FLAGS_CF_ERROR (bit value 0x100)
- For deferred cycle fees: PIN_BILL_FLAGS_DEF_CF_ERROR (bit value 0x200)
- For deferred purchase cycle fees: PIN_BILL_FLAGS_DEF_PURCHASE_ERROR (bit value 0x400)
- For deferred cancel cycle fees: PIN_BILL_FLAGS_DEF_CANCEL_ERROR (bit value 0x800)

Note

If any nonpaying child bill unit caused the failure, **pin_cycle_fees** updates the billing statuses set in the PIN_FLD_BILLING_STATUS and PIN_FLD_BILLING_STATUS_FLAGS fields of the **/billinfo** object for both child and parent bill units.

When the status of the **/billinfo** object is set to PIN_BILL_ERROR, the **pin_bill_accts** utility, which runs after **pin_cycle_fees**, does not select this **/billinfo** object for billing. If you rerun **pin_bill_accts** with the **-retry** option, the billing opcode stops with an error because the cycle fee processing failed for this **/billinfo** object.

Rerun **pin_cycle_fees** (directly or through the **pin_bill_day** script) before you can run billing on this **/billinfo** object.

pin_make_corrective_bills

Use the **pin_make_corrective_bills** utility to generate corrective bills for prior bills that have corrections or to process corrections on prior corrective bills. See "Creating Corrective Bills".

This utility should be used for corrective billing only. For regular bills, use "pin_bill_accts".

Location

BRM_home/**bin**

Syntax

```
pin_make_corrective_bills  [-bill_no bill1, bill2,...]
                           [-account_no acct1, acct2, acct3,...]
                           [-start mm/dd/yy|number_of_days]
                           [-end mm/dd/yy|number_of_days]
                           [-file file_name]
                           [-item_type /item/A/R_item]
                           [-adj_reason D_id R_id]
                           [-threshold_amount min_amt]
                           [-correction_reason D_id R_id]
                           [-corrective_inv_type  R|L  D|S]
                           [-no_adj_create [Y | N]]
                           [-validate_only]
                           [-help]
```

Parameters

-bill_no *bill1, bill2...*
Specifies the bill numbers for which corrective bills are required.

-account_no *acct1, acct2, acct3...*
Specifies the account numbers for which corrective bills are required.

-start *mm/dd/yy|number_of_days*
Specifies the start time to use in selecting the bills for corrective billing. When you specify the start date as a specific date, for example **-start 01/31/01**, the utility attempts to create corrective bills for all bills finalized on or before January 31, 2001. When you specify the start date as a number of days, for example **-end 5**, the utility attempts to create corrective bills for all bills finalized on or in the 5 days before the current date).
If you omit this parameter, **pin_make_corrective_bills** sets the start date to 1 month before the current date.

-end *mm/dd/yy|number_of_days*
Specifies the due date for accounts to be billed. You can specify a specific due date (for example, **-end 01/31/12** includes accounts with a due date on or before January 31, 2012) or you can specify the due date as number of days from the current date (for example, **-end 5** includes accounts with a due date on or 5 days before the current date).
If you omit this parameter, **pin_make_corrective_bill** sets the end date for the corrective billing for each selected bill to the end date on that bill.

Note

The **end** time parameter cannot be greater than the system time. For example, if the current system date is 1/15/2012 and the **end** time specified is 1/31/2012, **pin_make_corrective_bills** fails with “bad config:time" error message.

-file *file_name*
Specifies that the list of accounts or the list of bill numbers will be read from a file, and provides the associated file name. The input file should be in XML format.

-item_type/item/*A/R_item*
Specifies the A/R item allocated to a bill as the criteria for selecting bills to include in the corrective billing. Enter the A/R item for example, **/item/adjustment**, **/item/disputed**, **/item/writeoff**, **/item/recvd**. Use **-item_type item/any** to select all A/R items. If you omit this parameter, **pin_make_corrective_bills** uses the default value, **/item/adjustment**, to select only those bills with adjustments allocated to them.

-adj_reason *D_id R_id*
Specifies the event-adjustment reasons associated with a bill as the criteria for selecting bills to include in the corrective billing. If you enter this parameter, you must specify both ***D_id*** which is the event adjustment domain number and ***R_id*** the reason number.
For example, **-adj_reason** *D_id R_id*

-threshold_amount *min_amt*
Specifies the minimum amount for A/R actions that should be reached in a bill to generate a corrective bill. **pin_make_corrective_bills** uses the value you input with the **-threshold_amount** parameter. If you omit this parameter, **pin_make_corrective_bills** uses the value currently in the **CorrectiveBillThreshold** business parameter.

-correction_reason*D_id R_id*
Specifies the correction reason to associate with the corrective bills. You must specify both ***D_id*** which is the event adjustment domain number and ***R_id*** the reason number.
For example, **-correction_reason** *D_id R_id*.

-corrective_inv_type R|L D|S
Specify the type of corrective invoice to associate with each corrective bill. **R** indicates **Replacement Invoice** and **L** indicates **Invoice Correction Letter**. **S** indicates **Summary** and **D** indicates **Detail**.
Enter the two values separated by a blank. For example, to generate a detailed replacement invoice, you specify **-corrective_inv_type R D** when you run the **pin_make_corrective_bills** utility.
If you omit one of the entries (as in **-corrective_inv_type R**) or omit the blank between the two entries (as in **-corrective_inv_type RD**), the **pin_make_corrective_bills** application will fail to generate the corrective bill.

-create_if_no_corrections Y|N
Specifies whether to generate the corrective bill when there are no A/R charges for a bill. The default value is **N** which results in **pin_make_corrective_bills** not generating the corrective bill for any bill with no A/R charges.
If you enter **Y** as the value for **-create_if_no_corrections**, set the **-corrective_inv_type** parameter to **R** (to indicate Replacement Invoice). **pin_make_corrective_bills** will not create the corrective bill if you specify **-corrective_inv_type** as **L** when you enter **Y** as the value for **-create_if_no_corrections**.
If you enter **Y** as the value for **-create_if_no_corrections** and you omit the **-corrective_inv_type** parameter, **pin_make_corrective_bills** creates the corrective bill if the **/payinfo** object for the account indicates a replacement invoice as the type of corrective invoice. Otherwise, it fails to create the corrective bill.

-validate_only
Specifies that **pin_make_corrective_bills** should only validate the eligibility of the bills for corrective billing. When you run **pin_make_corrective_bills** with the **-validate_only** parameter, the utility only verifies that the selected accounts or bills are eligible for corrective billing. It does not generate corrective bills.
When you use the **-validate_only** parameter, **pin_make_corrective_bills** lists the information for each bill in the **default.pinlog** log file.

-help
Displays the syntax and parameters for this utility.

Results

The **pin_make_corrective_bills** utility creates an entry in the **default.pinlog** utility log file for any errors it encounters in its process. The **default.pinlog** log file is either in the directory from which the utility was started, or in a directory specified in the configuration file.

Error Handling

If billing fails due to errors in bill units, the **pin_make_corrective_bills** utility automatically updates the billing status field in their **/billinfo** objects to **error**. If any nonpaying child bill units caused the failure, the utility updates the billing status for both child and parent bill units.

pin_remittance

Use this utility to calculate the remittance you owe to third parties, such as service providers and resellers. Typically, you run **pin_remittance** as part of the monthly remittance script, **pin_remit_month**, but you can also run it apart from the script. You should run billing on non-remittance accounts before calculating remittance.

Note

To connect to the BRM database, the **pin_remittance** utility needs a configuration file in the directory from which you run the utility. See "Connecting BRM Utilities".

Location

BRM_home/**bin**

Syntax

```
pin_remittance [-acct account_number] [-end date] [-output file_name]
               [-calconly] [-verbose] [-billing_cycle_alignment]
               [-help]
```

Parameters

-acct *account_number*
Calculates the remittance owed to an account. You can specify only one account number. Ensure that the account owns a remittance product.
If you do not use this parameter, **pin_remittance** runs on all remittance accounts.

-end *date*
Specifies the end date for which events are part of the calculation. **pin_remittance** calculates remittance for events that occurred before midnight of the day before the end date. For example, if the end date is 02/13/2020, **pin_remittance** includes all remittance events that took place through midnight of 02/12/2020.
If you do not specify this parameter, the end date is the current date. The start date is always the date of the previous remittance calculation.
The date format is *mm*/*dd*/*yyyy*.

-output [*file_name*]
Creates a remittance report. If you do not specify a file name, the report's default name is **rem_***date***.rep**, where *date* is the end date. To use a different name, specify that name with this parameter. If a file already exists with the report name you specify, **pin_remittance** overwrites the existing file.

-calconly
Calculates the remittance without writing the results to the BRM database. Use this parameter for testing and verification.

-billing_cycle_alignment
When the utility is run with the **-billing_cycle_alignment** parameter, the opcode's input PIN_FLD_FLAGS field is set to **1**. This sets the opcode's current time and date to whichever of the following is earliest:

- BRM's current time and date
- Value of the utility's optional **-end** *date* parameter
- One second before the account's next billing cycle begins

This ensures that billing is not triggered before remittance is calculated when the utility is run after a remittance account's billing date.

> **Note**
>
> By default, the **pin_remit_month** script runs the **pin_remittance** utility with the **-billing_cycle_alignment** parameter.

-verbose
Displays information about successful or failed processing as the utility runs.

> **Note**
>
> This parameter is always used 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_remittance** *any_other_parameter* **–v >** *filename*.**log**

-help
Displays the syntax and parameters for this utility.

Results

If the **pin_remittance** utility does not notify you that it was successful, look in the utility log file (**default.pinlog**) to find any errors. The log file is either in the directory from which the utility was started or in a directory specified in the configuration file.

pin_rollover

The **pin_bill_day** script runs this Oracle Communications Billing and Revenue Management (BRM) utility to roll over all expired sub-balances that have not been rolled over.

For more information, see "When Rollover Events Occur" in *PDC Creating Product Offerings*.

Location

BRM_home/**bin**

Syntax

```
pin_rollover [-verbose] [-test]
```

Parameters

-verbose
Displays information about successful or failed processing as the utility runs.

-test
Runs in test mode to find the accounts that meet the criteria for roll over, but does not perform any rollover. The test does not affect the balances (currency and noncurrency) of the accounts.

Results

If the **pin_rollover** utility does not notify you that it was successful, look in the utility log file (**default.pinlog**) to find any errors. The log file is either in the directory from which the utility was started or in a directory specified in the configuration file.

pin_rollup_sharing_balances

Use this utility to roll up noncurrency balances from the group members to the sharing group owner.

Location

BRM_home/**bin**

Syntax

```
pin_rollup_sharing_balances -file PinRollupSharingBalInput.xml [-verbose][-
help]
```

Parameters

-file PinRollupSharingBalInput.xml
Specifies the XML file containing the bill units to process. For example:

```
    <BusinessConfiguration
    xmlns=http://www.portal.com/schemas/BusinessConfig
    xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xsi:schemaLocation=http://www.portal.com/schemas/BusinessConfig
BusinessConfiguration.xsd>

    <RollupSharingBalConfiguration>
    <!-- List of Owner accounts -->
    <Billinfo>
    <BillinfoId>311038</BillinfoId>
    <Account>1210</Account>
    </Billinfo>
    <Billinfo>
    <BillinfoId>311039</BillinfoId>
    <Account>1423</Account>
    </Billinfo>
    </RollupSharingBalConfiguration>
    </BusinessConfiguration>
```

[-verbose]
Displays information about successful or failed processing as the utility runs.

[-help]
Displays the syntax and parameters for this utility.

Results

Each thread of this application calls the opcode PCM_OP_SUBSCRIPTION_SHARING_ROLLUP_BALANCE to roll up the member's balance to the owner.

pin_trial_bill_accts

Use this utility to calculate the balance due and create a *trial* bill and invoice for each account.

Note

Trial billing stops and reports a warning message when it encounters an account or bill unit with inactive status.

For information about trial billing, see "About Trial Billing".

Note

- To connect to the BRM database, the **pin_trial_bill_accts** utility needs a configuration file in the directory from which you run the utility. The **pin.conf** file for this utility is in *BRM_home***/apps/pin_trial_bill**. See "Connecting BRM Utilities" in *BRM System Administrator's Guide*.
- Trial billing may stop responding if the Data Manager has too few back ends configured. You should change the default configuration settings for Data Manager and increase the number of back ends. For more information about setting the number of back ends, see "Improving Data Manager and Queue Manager Performance" in *BRM System Administrator's Guide*.

Location

*BRM_home***/bin**

Syntax

```
pin_trial_bill_accts [-start mm/dd/yy|+/- numberOfDays|0]
                     [-end mm/dd/yy|+/- numberOfDays|0]
                     [-f inputFile|-f_control inputFile]
                     [-active|-inactive|-closed]
                     [-bill_only]
                     [-pay_type ID]
                     [-retry]
                     [-split]
                     [-verbose]
                     [-help]
                     [-hierarchy]
```

Parameters

-start *mm/dd/yy* or *yyyy*|+/- *numberOfDays*|0
-end *mm/dd/yy* or *yyyy*|+/- *numberOfDays*|0
The **start** and **end** dates determine which accounts are selected for trial billing and which billing cycles for those accounts are trial billed. See "Running Trial Billing With Date Ranges".

-f *inputFile*
Specifies the text file that contains the list of account POIDs for trial billing.
Example:

Create trial bills for accounts in *inputFile* whose billing date is less than **4/1/2030** (current date is 3/15/2030):

```
pin_trial_bill_accts -end 04/01/2030 -f myListOfAccounts
```

-f_control *inputFile*
Specifies the name and location of a text file that contains additional criteria for selecting accounts and bill units for trial billing. The default file is **pin_bill_run_control.xml** in the *BRM_home***/apps/pin_billd** directory.
This file can contain a list of account and bill unit POIDs or a list of billing segments, accounting days of month (DOMs), or both.
For more information, see "Specifying Bill Units, Billing Segments, and DOMs for Trial Billing".

> **Note**
>
> You should not specify billing segments or accounting DOMs with bill unit POIDs in the input file. If you do, the utility considers only the bill units for trial billing.

The billing run configuration file can contain either a list of account and bill unit POIDs or a list of billing segments and accounting days of months (DOMs).

-hierarchy
Specifies to run trial billing on a set of child and parent billing accounts. For more information, see "Trial Billing for Wholesale Hierarchies".
Example:

```
pin_trial_bill_accts -pay_type 10007 -hierarchy
```

-active|-inactive|-closed
Searches for accounts whose status is active, inactive, or closed. By default, all accounts are searched.

> **Note**
>
> The **active**, **inactive,** or **closed** parameter does not apply when you use the **-f** parameter.

-bill_only
Performs trial billing without generating trial invoices and collects revenue assurance data if you have enabled its collection in the trial billing utility configuration file. For more information, see "About Collecting Revenue Assurance Data from Trial Billing".

-pay_type *ID*
Generates trial billing for accounts with the specified payment method ID. If you do not specify the payment method ID, the **pin_trial_bill_accts** utility generates trial bills for all payment methods. See "Running Trial Billing According to Payment Type".

-retry
Runs trial billing for hierarchical bill units that were previously not billed during a trial invoicing run because of one or more errors. After the errors are resolved, use this parameter to rebill the unbilled hierarchical bill units.
You must run nonpaying child bill units before running paying parent bill units. You must also regenerate trial invoices on the parent bill units that did not contain an error. For example, to

run trial billing on bill units that contained an error and whose payment method is check, use the following syntax:

```
pin_trial_bill_accts -pay_type 10007 -retry
pin_trial_bill_accts -pay_type 10012 -retry
pin_trial_bill_accts -pay_type 10012
```

-split
Generates detailed revenue assurance data if you have enabled its collection in the trial billing utility configuration file. This parameter is valid only if Revenue Assurance Manager is installed.

> **Note**
>
> If you use this parameter, trial invoices are generated unless you also specify the **-bill_only** parameter.

You can view the detailed data by generating a Revenue Assurance Billing Detail report. The details are based on item types.

> **Note**
>
> If you specify both the **-split** and **-f_control** parameters and the input file includes a list of accounts and bill units, the Revenue Assurance Billing Detail report does not segregate the data based on the billing segment and billing day of month (DOM).

For more information, see "About Collecting Revenue Assurance Data from Trial Billing".

-verbose
Displays information about successful or failed processing as the utility runs.

-help
Displays the syntax and parameters for this utility.

Results

If the **pin_trial_bill_accts** utility does not notify you that it was successful, look in the utility log file (**default.pinlog**) to find any errors. The log file is either in the directory from which the utility was started or in a directory specified in the configuration file.

pin_trial_bill_purge

Use this utility to purge trial invoices created with the "pin_trial_bill_accts" utility. For information about trial billing, see "Running Trial Billing".

> **Note**
>
> To connect to the BRM database, the utility needs a configuration file in the directory from which you run the utility. The **pin.conf** file for this utility is in *BRM_home***/apps/pin_trial_bill**. See "Connecting BRM Utilities" in *BRM System Administrator's Guide*.

Location

BRM_home/**bin**

Syntax

```
pin_trial_bill_purge [-start mm/dd/yy|+/- number_of_days|0]
                     [-end mm/dd/yy|+/- number_of_days|0]
                     [-active|-inactive|-closed]
                     [-f inputFile|-all]
                     [-verbose][-help]
```

Parameters

-start *mm/dd/yy* or *yyyy*|+/- *number_of_days*|0
The **start** and **end** dates determine the billing cycles for which trial invoices are purged. Billing cycles must fall entirely within the range created by the start and end dates, except fall between the **start** and **end** dates. You can specify an absolute date or number of days before or after the current date.

> **Note**
>
> If you do not specify a **start** date, **pin_trial_bill_purge** will purge trial invoices for *all* billing cycles before the **end** date for the account. If **end** date is not specified, it uses the current date.

-end *mm/dd/yy* or *yyyy*|+/- *number_of_days*|0
The **end** date is used as the search criteria for retrieving accounts for purging trial invoices. It selects all accounts whose billing date is less than the **end** date. You can specify an absolute date or number of days before or after the current date.

> **Note**
>
> If you do not specify an **end** date, **pin_trial_bill_purge** uses the **current** date for the **end** date.

Examples:
Purge trial invoices for accounts whose billing date is on or before **4/1/2032 (current date is 3/15/2032)**:
pin_trial_bill_purge -end 4/1/2032
pin_trial_bill_purge -end +16
Purge trial invoices for accounts whose billing date is on or before **3/1/2032** (current date is 3/15/2032)
pin_trial_bill_purge -end 3/1/2032
pin_trial_bill_purge -end -14
Purge trial invoices for accounts whose billing date is on or before the current date:
pin_trial_bill_purge -end 0
pin_trial_bill_purge
Purge trial invoices for accounts with complete billing cycles between **2/1/2032** and **3/15/2032** (current date is 3/15/2032):
pin_trial_bill_purge -start 2/1/2032 -end 3/15/2032
pin_trial_bill_purge -start -45 -end 0

-active|-inactive|-closed
Searches for accounts whose status is active, inactive, or closed. By default, all accounts are searched.

> **Note**
>
> These parameters do not apply when you use the **-f** parameter.

-f *inputFile*
Specifies the text file that contains the list of account POIDs.
For example, to purge trial invoices for accounts in *inputFile* whose billing date is less than **4/1/2032** (current date is 4/15/2032):
pin_trial_bill_purge -end 4/1/2032 -f myListOfAccounts

-all
Purges all trial invoices in the database.

-verbose
Displays information about successful or failed processing as the utility runs.

> **Note**
>
> This parameter is always used with other parameters and commands. It is not position dependent. For example, you can enter **-verbose** 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_trial_bill_purge** any_other_parameter **–v >** *filename***.log**

-help
Displays the syntax and parameters for this utility.

> **Note**
>
> If you do not specify a start or end date or a file with list of accounts, **pin_trial_bill_purge** selects all accounts whose billing date is on or before the current date.

Results

If the **pin_trial_bill_purge** utility does not notify you that it was successful, look in the utility log file (**default.pinlog**) to find any errors. The log file is either in the directory from which the utility was started, or in a directory specified in the configuration file.

pin_update_bdom

Use this utility to change the billing day of month (DOM) for the parent and child accounts in a large hierarchy.

Note

To connect to the BRM database, the **pin_update_bdom** utility needs a configuration file in the directory from which you run the utility. The pin.conf file for this utility is in *BRM_home*/**apps/pin_billd**. See "Connecting BRM Utilities" in *BRM System Administrator's Guide*.

Location

BRM_home/**bin**

Syntax

```
 pin_update_bdom    [-billinfo poid] [-count limit] [-file sample.txt] [-f
sample.xml] [-retry] [-verbose] [-help]
```

Parameters

-billinfo *poid*
Runs the job for the specified bill unit.

Note

This is an optional parameter. If not provided, the utility finds all jobs with the status NEW and processes them one by one.

-count *limit*
Specifies the number of accounts to update.

-file *sample*.txt
Specifies the name of a text file containing the parent bill unit POID and the parent account POID to process in flist format. For example:

```
 0 PIN_FLD_RESULTS       ARRAY [0] allocated 20, used 3
1     PIN_FLD_POID           POID [0] 0.0.0.1 /billinfo 202821 6
1     PIN_FLD_ACCOUNT_OBJ    POID [0] 0.0.0.1 /account 202821 0
```

-f *sample*.xml
Specifies the list of parent accounts and details of the bill units. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<BusinessConfiguration
         xmlns="http://www.portal.com/schemas/BusinessConfig"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.portal.com/schemas/BusinessConfig
BusinessConfiguration.xsd">
         <BillRunConfiguration>
                 <BillingList>
                         <Account>250244</Account>
                         <Billinfo>25316</Billinfo>
                 </BillingList>
         </BillRunConfiguration>
</BusinessConfiguration>
```

-retry
Runs a search for **/job/bdom** jobs which are in ERROR status. It identifies and processes them one by one.

-verbose
Displays information about successful or failed processing as the utility runs.

-help
Displays the syntax and parameters for this utility.

Results

Each thread of this application will call PCM_OP_CUST_SET_BILLINFO opcode for each parent and child billinfo to proceed with updating the billing DOM.

Error Handling

If the application runs into errors for a few accounts or if it crashes, the status of the job is set to ERROR. To continue the process for the accounts with the errors, you need to provide the **-retry** option. To do this for a specific job, you can provide the corresponding billinfo POID as input using the **-billinfo** option along with the **-retry** option.

pin_update_journals

Use the **pin_update_journals** utility to process the temporary journals of nonpaying child bill units and then roll them up to the paying parent bill unit. You must run this utility before billing the paying parent bill unit. See "About Rolling Journals Up to the Wholesale Parent".

> **Note**
>
> To connect to the BRM database, the **pin_update_journals** utility needs a configuration file in the directory from which you run the utility. See "Connecting BRM Utilities" in *BRM System Administrator's Guide*.

Location

BRM_home/**bin**

Syntax

```
pin_update_journals [-verbose] [-help]
```

Parameters

-verbose
Displays information about successful or failed processing as the utility runs.

-help
Displays the syntax and parameters for this utility.

Results

If the **pin_update_journals** utility does not notify you that it was successful, look in the utility log file (**default.pinlog**) to find any errors. The log file is either in the directory from which the utility was started or in a directory specified in the configuration file.

Error Handling

When the **pin_update_journals** utility encounters an error while processing A/R items in the temporary tables, it sets the PIN_FLD_BILLING_STATUS billing status field of the paying parent bill unit (**/billinfo** object) to PIN_BILL_ERROR. In addition, it sets the appropriate bit of the PIN_FLD_BILLING_STATUS_FLAGS field of the **/billinfo** object as PIN_BILL_FLAGS_UPDATE_JOURNALS_ERROR (bit value 0x2000).

After you have resolved the processing errors, you can reprocess the A/R items by running the **pin_update_journals** utility again.