

**Oracle® Communications
Billing and Revenue Management**

Configuring and Running Billing

Release 7.5

E16696-15

December 2019

Copyright © 2011, 2019, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Preface	xv
Audience	xv
Accessing Oracle Communications Documentation	xv
Documentation Accessibility	xv
Document Revision History	xv

Part I About Billing

1 About Billing

About Billing Customers	1-1
About Accounting and Billing Cycles	1-2
About the First Cycle Forward Fee in a New Account.....	1-3
About Accounting Cycles	1-3
About Accounting Cycle Dates	1-4
About Billing Cycles	1-4
How BRM Sets the Billing DOM	1-5
About Actions Performed at the End of a Billing Cycle.....	1-5
About Auto-Triggered Billing.....	1-6
About Delayed Billing	1-7
About Delayed Billing and Changing the Billing DOM	1-10
About On-Demand Billing.....	1-10
About Bill Now.....	1-10
About Accounting Types	1-12
How BRM Determines the Accounting Type	1-12
About Changing a Bill Unit’s Accounting Type.....	1-13
About Bill Items	1-13
About Multiple Bills per Cycle	1-13
About Billing Cycles for Hierarchical Accounts with Multiple Bill Units.....	1-14
How BRM Creates a Bill	1-14
About Bill Numbering in BRM	1-16
How Bill Now Works	1-16
How On-Demand Billing Works	1-17
Billing for Sponsorship	1-18
Related Documents	1-18

2 About Corrective Billing

About Bill Corrections	2-1
About Simple Corrections to Bills.....	2-1
About Corrections to Charges in Bills.....	2-1
About Billing Customers.....	2-2
About Corrective Bills	2-2
About Corrective Billing in Subordinate Hierarchies.....	2-2
About Generating and Viewing Corrective Bills in BRM	2-3
Regulating Permissions for CSR Roles	2-3
Validating Bills for the Corrective Billing Process	2-3
Standard Validations Performed by BRM.....	2-3
Policy Validations Performed by BRM.....	2-3
About Numbering Corrective Bills in BRM	2-4
About Maintaining a Bill’s History	2-4
Corrective Invoice Document Types in BRM.....	2-5
About Threshold Amounts for Corrective Billing.....	2-5
Correction Reasons for Corrective Billing	2-6
About Payment Due Dates for Corrective Bills	2-6
Handling Payments for Bills with Corrections.....	2-6
Processing Payments for Previous Bills.....	2-6
Processing Payments Received before Creating a Corrective Bill	2-7
Processing Payments Received before Issuing a Corrective Invoice.....	2-7
Processing Payments after Issuing a Corrective Invoice.....	2-7
About Open Item Accounting and Rerating	2-7
About Handling Updates to Discount Sharing and Charge Sharing	2-8
About Collections.....	2-8
Preventing Corrective Bill Generation to Handle A/R Actions.....	2-8
Disputes.....	2-8
Settlements.....	2-8
Write-Offs.....	2-8
About Automatic Allocation from Rerating	2-8
When BRM Generates Corrective Bills	2-9
How BRM Handles Multiple Bill Corrections in the Same Period.....	2-9
How BRM Handles Bill Corrections Spanning Multiple Periods.....	2-10
Setting Up the Corrective Billing Process in BRM	2-10
Enabling Corrective Billing in BRM	2-11
Restricting CSR’s Permission to Corrective Billing.....	2-12
Specifying the Corrective Invoice Type for a Corrective Bill	2-12
Specifying the Default Invoice Type at the Account Level.....	2-12
Specifying the Invoice Type When Creating the Corrective Bill	2-12
Setting Up Custom Validations for Corrective Bills	2-13
Setting Up Bill Numbers for Corrective Bills	2-13
Retaining the Numbering Format Used for Regular Bills	2-13
Customizing the Bill Numbering Format.....	2-14
Customizing Due Dates for Corrective Bills	2-14
Specifying Correction Reasons.....	2-14
Providing Custom Reason Codes for Bill Corrections	2-14

Specifying the Threshold Amount for Corrective Bills	2-15
Providing Threshold Amounts Based on Customers	2-16
Enabling Automatic Allocation of Adjustments from Rerating.....	2-16
Rejecting Payments for Prior Bills	2-17
Generating Corrective Bills for Partially or Fully Paid Bills	2-18
Generating Corrective Bills	2-19
Generating Corrective Bills Using Customer Center	2-19
Generating Corrective Bills for Bills without Charge Corrections.....	2-19
How PCM_OP_BILL_MAKE_CORRECTIVE BILL Works	2-20
Generating Corrective Bills with the pin_make_corrective_bills Utility	2-21
Post-Processing Actions for Corrective Bills	2-22
Viewing Corrective Bills.....	2-22

Part II Configuring Billing

3 Setting Business Policies for Billing

Setting Default Billing Properties for Account Creation	3-1
Setting the Default Accounting Day of Month (DOM)	3-1
Setting the Default Billing-Cycle Length	3-2
Setting the Default Accounting Type	3-2
Setting the Billing DOM According to the Payment Method.....	3-3
Setting the First Billing Cycle to the Day after Account Creation.....	3-3
Setting the Bill Unit Status When Billing Errors Occur	3-3
Specifying the Minimum Payment to Collect	3-4
Setting the Minimum Amount to Charge.....	3-4
Setting the Minimum Amount for Invoices	3-5
Setting the Minimum Amount for Finalizing Bills.....	3-5
Specifying the Minimum Amount to Refund	3-6
Defining Non-refundable Items.....	3-6
Managing Cycles	3-7
About Time-Stamp Rounding.....	3-7
Specifying How to Handle Partial Accounting Cycles.....	3-8
Short and Long Cycles with New Accounts	3-9
How BRM Calculates Long Billing Cycles	3-9
Rounding Up Long Billing Cycles.....	3-11
Changing How to Handle Partial Billing Cycles.....	3-11
Aligning Account and Cycle Start and End Times	3-11
Including Previous Balances in the Current Amount Due in Open Item Accounting	3-12
Specifying Which Billing Cycle to Assign to Deferred Purchase Fees	3-13
About Billing Cycle Forward Fees in Advance	3-13
About Applying Cycle Forward Fees in Parallel.....	3-13
Configuring BRM to Apply Cycle Forward Fees in Parallel	3-14
About Enforcing Cycle Forward Fee Processing Prior to Billing.....	3-15
About Aggregating Service Charges to Account Level Items.....	3-16
Handling Skipped Billing	3-17
Using the pin_bill_day Script to Apply Parallel Cycle Forward Fees.....	3-17

About Limitations and Impacts of Applying Cycle Forward Fees in Parallel.....	3-18
About Flexible Cycles	3-19
Configuring Flexible Cycles and Cycle Forward Fees	3-19
Charging Cycle Forward Fees Associated with Flexible Cycles.....	3-20
Prorating Cycle Forward Fees Associated with Flexible Cycles.....	3-20
Calculating Product Cycle Fees for Backdating.....	3-20
Customizing Accounting Cycles.....	3-21
Customizing How to Bill Events That Occur Between Billing Cycles	3-21
Enabling Product Priority While Applying Cycle Fees.....	3-22
About Calculating Charges When You Change the Rate	3-23
About Rate Changes in the Current Cycle	3-23
About Rate Changes in a Future Cycle	3-24
Calculating Charges When You Change the Rate in a Cycle	3-24
Configuring BRM to Apply Multiple Rates in a Cycle.....	3-24
Configuring Event Notification for Rate Changes	3-25
Configuring Event Notification to Create Rerating Requests for Rate Changes.....	3-25
Creating Rerating Requests When You Change the Rate	3-26
Recalculating the Cycle Fees When the Rate Changes	3-26
Prorating Different Resources When the Rate Changes.....	3-26
Using 31-Day Billing	3-26
About Setting the Alternate Billing Day of Month.....	3-27
Setting the 31-Day Billing Feature	3-28
Switching to 31-Day Billing during BRM Installation.....	3-28
Switching to 31-Day Billing after You Install BRM.....	3-29
Setting the Forward and Back Billing Options	3-29
Customizing Bill and Invoice Numbers	3-30
Customizing the Format of Bill and Invoice Numbers.....	3-30
Specifying When to Apply Custom Bill Numbers	3-31
Configuring Bill Now.....	3-31
Selecting the Input for Bill Now.....	3-31
Changing the Bill Now Due Date	3-31
Providing Discounts to Closed Accounts.....	3-32
Prorating Cycle Arrears and Cycle Forward Arrears for Bill Now	3-32
Creating Two Bills during the Delayed Billing Period	3-33
Calculating Deferred Taxes with Bill Now	3-34
Customizing Bill Now	3-34
Applying Discounts and Folds with Bill Now.....	3-35
Running Bill Now for a Service.....	3-35
Disabling Auto-Triggered Billing.....	3-35
Disabling Auto-Triggered Billing by Specifying Billing Delay	3-36
Disabling Auto-Triggered Billing by Setting AutoTriggeringLimit.....	3-36
Setting Up Delayed Billing	3-38
Configuring Delayed Billing	3-38
Specifying the Billing Delay Interval	3-38
Specifying Auto-Triggered Billing for Delayed Billing.....	3-39
Specifying When to Apply Cycle Forward Fees and Cycle Rollovers	3-40
Enforcing Partial Billing in the Billing Delay Interval	3-41

Setting Delayed Cycle Start Dates to the 29th, 30th, or 31st	3-42
Billing Cycle Override	3-42
Using Pipeline Manager to Trigger Billing.....	3-43
Enabling a Billing Delay for Rated Event Loader.....	3-43
Enabling Rerating and Rollover Correction Due to Delayed Events.....	3-43
Modifying Business Parameters to Enable Rerating and Rollover Correction	3-45
Configuring Event Notification for Rerating and Rollover Correction	3-46
Configuring the BRM Cutoff Time	3-47
How Billing and Invoicing Are Affected by Changing the Cutoff Time.....	3-48
How Rating Is Affected by Changing the Cutoff Time	3-49
How General Ledger (G/L) Is Affected by Changing the Cutoff Time.....	3-49
How Searches Are Affected by Changing the Cutoff Time.....	3-49
How TimeStamp Fields Are Affected by Changing the Cutoff Time	3-49
Configuring the Billing Cutoff Time	3-50
Setting Up Billing for Sponsorship	3-51
Setting Up Billing to Run in a Multischema Environment	3-54
Running Billing on One Schema at a Time.....	3-54
Running Billing on Multiple Schemas Simultaneously.....	3-55
About Suspending Billing of Accounts and Bills.....	3-55
Suspending Billing of Closed Accounts.....	3-56
Suspending Billing of an Account's Bill	3-56
Setting up Billing in Subordinate Hierarchies.....	3-56

4 About Proration

Calculating Prorated Cycle Fees	4-1
Calculating the Unit Interval	4-1
Calculating Unit Interval When use_number_of_days_in_month Is Not Set or 0.....	4-2
Calculating Unit Interval When use_number_of_days_in_month Is Set to 1.....	4-2
Calculating Unit Interval When Billing Day of Month Is 29, 30, or 31	4-2
Examples of Proration	4-2
Example 1: Use_number_of_days_in_month Is Not Set or Set to 0	4-3
Example 2: Use_number_of_days_in_month Is Set to 1	4-4
Examples Using the 29th, 30th, and 31st for Billing Day of Month.....	4-5
Example 3a: Use Forward Option with use_number_of_days_in_month Set to 0	4-5
Example 3b: Use Forward Option with use_number_of_days_in_month Set to 1	4-6
Example 3c: Use Back Option with use_number_of_days_in_month Set to 1	4-8
Example 3d: Use Back Option with use_number_of_days_in_month Set to 0.....	4-9
Enabling BRM to Apply Proration Rules	4-11
Proration for Special Cases.....	4-12
Special Cases	4-12
Addressing Special Cases	4-13
Calculating Prorated Cycle Fees and Refunds for Customized Products	4-14
Example 4: Applying a Cycle Forward Fee with a Customized Product	4-14
Example 5: Refunding a Prorated Cycle Forward Fee with a Customized Product.....	4-14
About 30-Day-Based Proration	4-15
Examples of 30-Day-Based Proration.....	4-15
Example 6: Prorated Purchase Fee with 31-day Billing Cycle.....	4-16

Example 7: Prorated Cancel Fee with 31-day Billing Cycle.....	4-16
Example 8: Prorated Purchase Fee with 28-Day Billing Cycle.....	4-17
Example 9: Prorated Cancel Fee with 28-Day Billing Cycle.....	4-18
Special Cases.....	4-18
Example 10: Full Purchase Fee Charged When Service Is Provided for 1 Day Less.....	4-18
Example 11: Full Cancel Fee Refunded When Service Has Been Used for 1 Day.....	4-19
Enabling 30-Day-Based Proration.....	4-19
Using Two Events to Prorate Charges for Products Whose Validity Ends in First Cycle.....	4-19
Prorating Cycle Fees after a Discount Purchase or Cancellation.....	4-21
Examples of Cycle Fee Proration.....	4-21
Example 12: Cycle Fee Is Refunded after a Discount Purchase.....	4-21
Example 13: Free Minutes Are Prorated after a Discount Cancellation.....	4-21
Example 14: Canceled Discount Proration Is Not Taken into Account When Product Is Canceled.....	4-22
Prorating Cycle Fees When a Discount's Cycle Start or End Date Is Changed.....	4-22
Examples of Cycle Fee Proration.....	4-22
Example 15: Cycle Fee Is Refunded When a Discount's Cycle Start Date Is Changed.....	4-22
Example 16: Cycle Fee Is Charged When a Discount's Cycle Start Date Is Changed....	4-22
Example 17: Cycle Fee Is Refunded When Discount's Cycle End Date Is Changed.....	4-23
Example 18: Cycle Fee Is Charged When Discount's Cycle End Date Is Changed.....	4-23

5 Managing Bill Units with Your Custom Application

Opcodes Used with Bill Units.....	5-1
Creating /billinfo Objects.....	5-1
Associating Account Balances with /billinfo Objects.....	5-2
Updating /billinfo Objects.....	5-2
Preparing /billinfo Data.....	5-2
Assigning Default Billing Information.....	5-3
Assigning DOMs to /billinfo Objects.....	5-4
Assigning DOM Based on the Billing Segment.....	5-5
Customizing the DOM Assignment Process.....	5-5
Validating /billinfo Data.....	5-7
Validating Billing Segment Information.....	5-7
Suspending and Resuming Billing of Closed Accounts.....	5-8
Suspending Billing of Closed Accounts.....	5-8
Resuming Billing When Closed Accounts Are Reactivated.....	5-8
Deleting /billinfo Objects.....	5-9
Creating /billinfo Object Hierarchy and Sponsorship.....	5-9
Changing /billinfo Hierarchy and Sponsorship.....	5-10
Billing Delays for Moved /billinfo Objects.....	5-10

6 Offering the Best Price to Your Customers

About Offering the Best Price to Your Customers.....	6-1
How BRM Determines the Best Deal.....	6-2
Calculating the Best Price by Using the Best Pricing Opcode.....	6-3
About Calculating the Best Deal When Alternate Deal Has a Best Pricing Configuration....	6-4
About Finding the Best Deal in the Middle of a Billing Cycle.....	6-4

About Rerating Events to Apply the Best Price	6-4
About Rerating Events for a Prior Cycle for Which the Best Deal Was Applied	6-5
Adjusting the Account Balance to Apply the Best Deal	6-5
How BRM Calculates the Best Price for Subscription Groups	6-5
Calculating the Best Price for Subscription Services	6-5
Calculating the Best Price for Member Services	6-6
How BRM Calculates the Best Price in Resource Sharing Groups	6-6
Calculating the Best Price for a Discount Sharing Group	6-6
Calculating the Best Price for a Charge Sharing Group	6-6
About Applying Exclusion Rules for Deals in a Best Pricing Configuration	6-6
Mutually Exclusive Deals and Best Pricing	6-6
Mutually Exclusive Discounts and Best Pricing	6-7
About Keeping Track of Best Pricing Information	6-7
About Configuring BRM to Use Best Pricing	6-7
Enabling Best Pricing	6-7

7 Setting Up Pipeline-Triggered Billing

About Pipeline-Triggered Billing	7-1
Pipeline-Triggered Billing Components	7-2
How Pipeline Manager Triggers Billing	7-3
About the Pipeline-Triggered Billing Modules	7-4
About BillHandler	7-4
Overview of the Immediate Billing Process	7-5
About Suspending Billing-Trigger EDRs	7-6
Configuring Pipeline-Triggered Billing	7-7
Setting Up Pipeline Manager to Trigger Billing	7-7
Setting Up the Billing Batch Applications	7-8
Configuring Batch Controller to Start BillHandler	7-9
Configuring BillHandler	7-9

8 About Bill Cycle Management

About Managing Billing Cycles	8-1
About Billing Segments	8-1
About Accounting DOM Status	8-2
Implementing Bill Cycle Management	8-3
Setting Up Billing Segments	8-3
Editing the Billing Segment Configuration File	8-5
Updating Billing Segments	8-8
Associating Bill Units with Billing Segments	8-10
Changing a Bill Unit's Billing Segment	8-10
Assigning Accounting Days of Month to Bill Units in Billing Segments	8-10
Manually Assigning a Billing DOM	8-11
Automatically Assigning a Billing DOM	8-11
Changing a Bill Unit's Billing DOM	8-11

9 About Bill Run Management

About Managing Billing Runs	9-1
About Reducing Billing Run Loads.....	9-1
Billing Only Specified Accounts and Bill Units	9-1
About Managing Bill Due Dates	9-3
About Payment Terms	9-3
About Billing Calendars.....	9-4
About Billing Run-Time Due Date Adjustments	9-4
Reducing Billing Run Loads	9-4
Configuring Auto-Triggered Billing for Bill Run Management	9-4
Splitting a Billing Run into Multiple Runs	9-5
About Sponsored Charges in Split Billing Runs	9-7
Including Only Specified Billing DOMs in Billing Runs.....	9-7
Including Only Specified Billing Segments in Billing Runs	9-8
Sample Billing Run Configuration File.....	9-8
Validating Your Billing Run Configuration File Edits	9-9
Managing Bill Due Dates	9-9
About Configurable Bill Due Dates and Delayed Billing.....	9-9
Managing Payment Terms.....	9-9
Setting Up Payment Terms.....	9-9
Editing the Payment Terms Configuration File	9-10
Updating Payment Terms.....	9-12
Updating the pin.conf File to Use Payment Terms.....	9-12
Assigning Payment Terms to Bill Units.....	9-12
Managing Billing Calendars	9-13
Setting Up Billing Calendars	9-13
Editing the Billing Calendar Configuration File	9-14
Updating Billing Calendars.....	9-18
Associating Billing Calendars with Payment Terms	9-18
Specifying Due Date Adjustments in a Billing Run	9-18
Editing the Billing Run Configuration File to Adjust Bill Due Dates	9-19
Sample Billing Run Configuration File.....	9-21
Validating Your Billing Run Configuration File Edits	9-21
How BRM Calculates Bill Due Dates	9-22
Customizing Bill Due Date Calculations for Payment Terms	9-22
Functions for Calculating Payment Due Dates.....	9-23
fm_bill_pol_calc_pymt_due_t.....	9-25

10 About Bill Suppression

About Suppressing Bills	10-1
About Automatic Bill Suppression.....	10-2
About Manual Bill Suppression	10-2
About Manual Account Suppression	10-3
Suppressed Accounts versus Inactive Accounts	10-3
Exceptions to Bill Suppression	10-4
How Exceptions Affect Manual Bill and Account Suppression	10-5
Automatically Suppressing Bills	10-5

Setting Up Automatic Bill Suppression	10-6
Associating Bill Suppression Information with Customer Segments	10-6
Editing the Bill Suppression Configuration File.....	10-7
Sample Bill Suppression Configuration File	10-8
Validating Your Bill Suppression Configuration File Edits.....	10-9
Manually Suppressing Bills	10-9
Manually Suppressing Accounts	10-10
How Bill Suppression Works	10-11
How BRM Determines Whether Bills Should Be Suppressed	10-11
How BRM Suppresses Bills	10-14
How BRM Suppresses Accounts.....	10-14
How BRM Ends Manual Account Suppression.....	10-15
Customizing Bill Suppression Exceptions	10-15
Adding Bill Suppression Exceptions.....	10-16
Deleting Bill Suppression Exceptions	10-17

11 Creating Custom Bill Items

About Custom Bill Items	11-1
About Defining Custom Bill Items	11-2
About Tracking Charges in Bill Items	11-2
About Creating /item Objects	11-3
About Assigning Custom Bill Items to Events	11-3
About Using Event and Service Combinations to Assign Bill Items.....	11-3
About Using Event Attributes to Assign Bill Items	11-5
How BRM Assigns Custom Bill Items to Events	11-5
Cumulative Custom Item for Taxes	11-6
How Batch Rating Assigns Custom Bill Items.....	11-6
How Real-Time Rating Assigns Custom Bill Items	11-8
About Bill Items and Universal Event Loader	11-9
Setting Up BRM to Assign Custom Bill Items to Events	11-9
Assigning Item Tags Based on Event and Service Combinations.....	11-10
Assigning Item Tags Based on Event Attributes	11-11
Setting Up Batch Rating to Assign Items Based on Event Attributes	11-11
Setting Up Real-Time Rating to Assign Items Based on Event Attributes	11-12
Mapping Item Tags to Item Types.....	11-12
Verifying Item-Tag-to-Item-Type Mapping.....	11-14
Assigning Bill Items to Event Balance Impacts	11-15
How Batch Rating Assigns Custom Bill Items to Events for Balance Impacts	11-16
Creating a Batch Rating iScript for Balance Impacts	11-17
Creating Custom Sponsored Bill Items	11-18
Splitting Sponsored Charges into Multiple Items	11-19

12 Remitting Funds to Third Parties

About Remittance	12-1
About Remittance Products.....	12-2
About Defining Remittance Specifications.....	12-2

About Remittance Criteria.....	12-3
About Calculating Remittance	12-4
Setting Up Remittance	12-4
Creating a Remittance Product	12-5
Creating a Remittance Account.....	12-6
Loading the Remittance Fields File	12-6
Defining Remittance Specifications	12-7
Loading the Remittance Specifications	12-8
Loading Remittance Specifications on Single-Schema Systems.....	12-9
Loading Remittance Specifications on Multischema Systems	12-9
Running Remittance	12-10
Calculating Remittance	12-10
Running the Monthly Remittance Script	12-10
Running the Remittance Utility Separately.....	12-11
Creating Remittance Reports.....	12-12
Changing the Balance of a Remittance Account.....	12-12
Using Remittance with Brands	12-12
Using Remittance with Multiple Database Schemas	12-13
Running Remittance on One Schema at a Time	12-13
Running Remittance on Multiple Schemas Simultaneously.....	12-14
Improving Remittance Performance	12-14
Using Remittance for Sales Commissions	12-15
Example of Setting Up a Remittance Specification	12-15
About Customizing Remittance	12-17
About Adding Custom Remittance Criteria	12-17
Defining Custom Remittance Fields	12-18
Specifying Custom Remittance Criteria	12-19
About Using Custom Ratable Usage Metrics to Calculate Remittance	12-20
Calculating Remittance Using Custom RUMs	12-21
How Remittance Works	12-21
Retrieving Remittance Accounts.....	12-21
Calculating the Remittance Amount	12-21
Verifying the Remittance Specification File	12-22
Customizing Remittance	12-22

Part III Running Billing

13 Running Billing Utilities

About Billing Your Customers	13-1
Regular Billing Process.....	13-1
Corrective Billing Process	13-2
About the Billing Utilities	13-2
Billing Accounts with the pin_bill_accts Utility	13-3
When to Run the pin_bill_accts Utility	13-3
Increasing Performance of the pin_bill_accts Utility	13-3
Customizing the pin_bill_accts Utility	13-3
Billing Accounts with the pin_make_corrective_bill Utility.....	13-3

When to Run the pin_make_corrective_bills Utility.....	13-4
Increasing Performance of the pin_make_corrective_bill Utility	13-4
Customizing the pin_make_corrective_bill Utility	13-4
Generating Invoices with the pin_inv_accts Utility	13-4
When to Run pin_inv_accts.....	13-5
Viewing Invoices.....	13-5
Emailing or Printing Invoices.....	13-5
Prorating Cycle-Forward Fees and Canceling Products with the pin_cycle_fees Utility	13-5
How the pin_cycle_fees Utility Prorates Cycle Forward Fees	13-6
When to Run the pin_cycle_fees Utility	13-6
Informing Customers That a Free Period Has Ended	13-6
Improving Performance of the pin_cycle_fees Utility.....	13-6
Executing Deferred Actions with the pin_deferred_act Utility	13-6
When to Run the pin_deferred_act Utility	13-7
About Running the Billing Scripts	13-7
Customizing the Billing Scripts.....	13-7
Running Billing.....	13-7
What Time to Run Billing Scripts	13-8
Manually Running the pin_bill_day Script	13-8
Customizing the pin_bill_day Script for Best Pricing Options	13-9
To Bill Subordinates before Member Discounts	13-9
To Bill Member Discounts before Subordinates	13-9
To Apply Cycle Fees for Subordinates and Sponsorship Members in One Run.....	13-10
To Apply Cycle Fees for Subordinates, Sponsorship Members, and Discount-Sharing Group Members in One Run	13-10
Specifying Start and End Times	13-10
Setting Start and End Dates for pin_collect.....	13-11
Editing Billing Utility Configuration Files	13-11
Editing the Billing Scripts	13-11
Changing the Path in the Billing Scripts	13-11
Testing Billing	13-11
Running Daily Billing.....	13-12
Running Weekly Billing.....	13-12
Running Monthly Billing.....	13-12
Handling Billing Failures	13-13
If the Billing Utility Was Interrupted	13-13
If You Miss a Daily Billing Run.....	13-13
Running Billing Utilities Manually	13-13
Monitoring Billing Activity	13-14
Checking for Payment-Processing Errors.....	13-14
Maintaining Transmission Logs for Billing Transactions	13-14

14 About Trial Billing

About Trial Billing.....	14-1
Comparing Billing and Trial Billing.....	14-1
How Trial Billing Works	14-2
Generating Trial Invoices.....	14-3

Collecting Split Revenue Assurance Data	14-3
About Trial Invoices	14-3
How Trial Invoices Are Stored.....	14-5
Using Trial Invoices for Validating Billing Charges	14-5
About Collecting Revenue Assurance Data From Trial Billing	14-6
Running Trial Billing	14-6
Specifying Accounts for Trial Billing	14-7
Specifying Bill Units, Billing Segments, and DOMs for Trial Billing	14-8
Specifying Bill Units for Trial Billing	14-9
Specifying Accounting DOMs for Trial Billing	14-10
Specifying Billing Segments for Trial Billing	14-10
Creating Trial Bills without Generating Trial Invoices	14-10
Hierarchical and Sponsor Groups for Trial Billing	14-11
Purging Trial Invoices	14-12
Exporting Trial Invoices	14-12
Viewing Trial Invoices	14-12
Improving Trial Billing Performance	14-13

Part IV Billing Utilities

15 Billing Utilities

load_config_item_tags.....	15-2
load_config_item_types	15-4
load_pin_bill_suppression.....	15-6
load_pin_billing_segment.....	15-8
load_pin_calendar	15-10
load_pin_payment_term	15-12
load_pin_remittance_flds	15-14
load_pin_remittance_spec	15-16
pin_bill_accts.....	15-18
pin_cycle_fees	15-23
pin_cycle_forward.....	15-26
pin_deferred_act.....	15-27
pin_make_corrective_bills.....	15-30
pin_remittance	15-33
pin_rollover	15-35
pin_trial_bill_accts.....	15-36
pin_trial_bill_purge.....	15-41
pin_update_items_journals.....	15-43

Preface

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

Audience

This document is intended for operations personnel and system administrators.

Accessing Oracle Communications Documentation

BRM documentation and additional Oracle documentation; such as Oracle Database documentation, is available from Oracle Help Center:

<http://docs.oracle.com>

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

<https://edelivery.oracle.com>

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at

<http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit

<http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit

<http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

Document Revision History

The following table lists the revision history for this book.

Version	Date	Description
E16696-01	November 2011	Initial release.
E16696-02	May 2012	Documentation updates for BRM 7.5 Patch Set 1. <ul style="list-style-type: none">■ Made minor formatting and text changes.

Version	Date	Description
E16696-03	December 2012	Documentation updates for BRM 7.5 Patch Set 3. <ul style="list-style-type: none"> Made minor formatting and text changes. Added the "Updating the pin.conf File to Use Payment Terms" section.
E16696-04	March 2013	Documentation updates for BRM 7.5 Patch Set 4. <ul style="list-style-type: none"> Made minor formatting and text changes. Added the "Assigning Bill Items to Event Balance Impacts", "How Batch Rating Assigns Custom Bill Items to Events for Balance Impacts", "Creating a Batch Rating iScript for Balance Impacts", "Creating Custom Sponsored Bill Items", and "Splitting Sponsored Charges into Multiple Items" sections. Added the "Setting up Billing in Subordinate Hierarchies" section. Replaced multidatabase information with multischema information.
E16696-05	July 2013	Documentation updates for BRM 7.5 Patch Set 5. <ul style="list-style-type: none"> Made minor formatting and text changes.
E16696-06	February 2014	Documentation updates for BRM 7.5 Patch Set 7. <ul style="list-style-type: none"> Made minor formatting and text changes.
E16696-07	May 2014	Documentation updates for BRM 7.5 Patch Set 8. <ul style="list-style-type: none"> Made minor formatting and text changes.
E16696-08	August 2014	Documentation updates for BRM 7.5 Patch Set 9. <ul style="list-style-type: none"> Made minor formatting and text changes. Added the "Using Two Events to Prorate Charges for Products Whose Validity Ends in First Cycle" section. Multiple updates in "Managing Bill Units with Your Custom Application". In the "pin_bill_accts" utility section, added SEPA to the Payment Method ID table. Added two parameters for trial billing in the "pin_trial_bill_accts" utility section.
E16696-09	January 2015	Documentation updates for BRM 7.5 Patch Set 11. <ul style="list-style-type: none"> Made minor formatting and text changes. Updated "How Pipeline Manager Triggers Billing".
E16696-10	June 2015	Documentation updates for BRM 7.5 Patch Set 12. <ul style="list-style-type: none"> Made minor formatting and text changes. Updated the following sections: <ul style="list-style-type: none"> About Automatic Bill Suppression Assigning Item Tags Based on Event and Service Combinations Mapping Item Tags to Item Types load_config_item_tags load_config_item_types

Version	Date	Description
E16696-11	August 2015	Documentation updates for BRM 7.5 Maintenance Patch Set 1. <ul style="list-style-type: none"> Updated the "Purging Trial Invoices" section.
E16696-12	December 2015	Documentation updates for BRM 7.5 Patch Set 14. <ul style="list-style-type: none"> Added the "About Time-Stamp Rounding" section. Updated the following sections: <ul style="list-style-type: none"> Setting the Minimum Amount to Charge Enabling Product Priority While Applying Cycle Fees
E16696-13	August 2016	Documentation updates for BRM 7.5 Patch Set 16. <ul style="list-style-type: none"> Updated the following sections: <ul style="list-style-type: none"> Exceptions to Bill Suppression About Actions Performed at the End of a Billing Cycle About Accounting Cycles
E16696-14	August 2017	Documentation updates for BRM 7.5 Patch Set 19. <ul style="list-style-type: none"> Added the following section: <ul style="list-style-type: none"> Enabling BRM to Apply Proration Rules
E16696-15	December 2019	Documentation updates for BRM 7.5 Patch Set 23. <ul style="list-style-type: none"> Updated the following section: <ul style="list-style-type: none"> Mapping Item Tags to Item Types

Part I

About Billing

Part I describes the basic concepts of the Oracle Communications Billing and Revenue Management (BRM) billing process. It contains the following chapters:

- [About Billing](#)
- [About Corrective Billing](#)

About Billing

This chapter describes the basic concepts of Oracle Communications Billing and Revenue Management (BRM) billing.

See "[About Corrective Billing](#)" for information on generating new bills based on corrections performed against a past billed period.

About Billing Customers

BRM billing is based on monthly cycles. Each account's bill unit has a billing day of month (DOM), which is typically the day of month on which the account is created. For example, if an account is created on May 7, all of its bill units, by default, would have the seventh day of the month as their billing DOM. If the account is billed monthly, its bills are generated on June 7, July 7, August 7, and so on. Most customer accounts are billed monthly, but you can bill accounts at any monthly interval; for example, bimonthly, quarterly, semiannually, or annually.

Note: By default, all bill units in an account have the same billing DOM and billing frequency, but you can modify each bill unit to have a different billing DOM and billing frequency.

To bill customers, you run a set of billing applications. A bill is produced for every bill unit. One billing application, `pin_bill_accts`, finds every bill unit that has a billing date of the previous day (or earlier if you do not run the billing applications daily). After finding the bill units, BRM does the following to them:

1. **Performs monthly accounting.** BRM compiles the total amount of balance impacts that have occurred in the past month. This can include usage fees and subscription fees. This monthly accounting occurs at the end of each accounting cycle.
2. **Finalizes the bill.** To finalize a bill, BRM changes the status of all the bill items associated with the bill from pending to open so that they stop accumulating charges and so that payments can be applied to them. In addition, a payment due date is added to the bill. The time period during which charges accumulate in an account before a bill is finalized is called the billing cycle. Typically, a bill is finalized monthly, at the end of each accounting cycle. However, you can bill in any multiple of one month; for example, every two months, quarterly, or yearly. A finalized bill includes balance impacts from each accounting cycle in the billing cycle.

Note: The actions that billing performs can vary depending on your billing configuration; for example, if you use delayed billing or Bill Now. See "[About Delayed Billing](#)" and "[About Bill Now](#)".

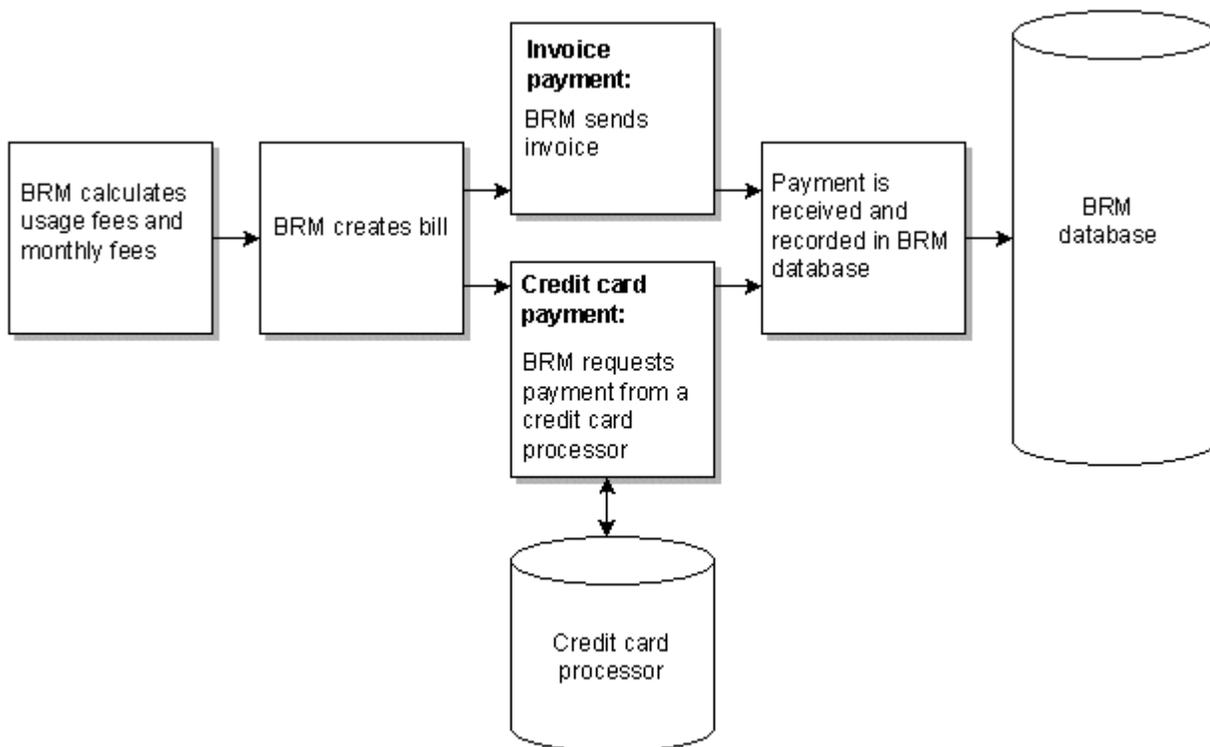
3. **Requests a payment.** BRM supports two types of payments:
- You process BRM-initiated payments by automatically requesting payments from a credit card or debit card processor.
 - You process externally initiated payments by sending invoices, receiving the payments, and processing the payments in batches.

An invoice lists the events that were charged for, and the customer's total balance for that bill.

When a payment is recorded in the BRM database, the customer's account balances are updated automatically.

Figure 1-1 shows how BRM compiles bills and requests payments from customers:

Figure 1-1 Regular Billing Process in BRM



For information about the billing opcodes, see "[How BRM Creates a Bill](#)".

About Accounting and Billing Cycles

Billing is performed in *cycles*. There are two types of cycles:

- The *accounting cycle* compiles all of a customer's balance impacts and stores them in bill items. The accounting cycle is always monthly.

- The *billing cycle* defines how often to request a payment for the balance impacts contained in the bill items. You can request payments every month, or in any number of complete months; for example, quarterly. Therefore, the accounting cycle and the billing cycle always start on the same date, but they can be different lengths.

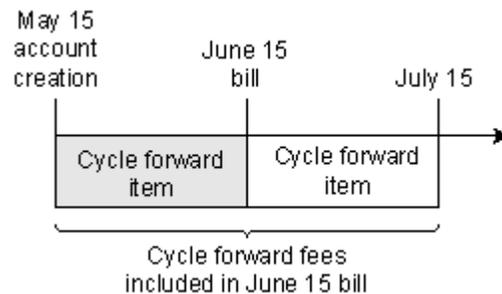
Accounting cycles and billing cycles are different in other important ways:

- **Customer impact.** The accounting cycle is an internal cycle; that is, it does not affect a customer in any way other than adjusting the account balance. The billing cycle is an external cycle. After you run billing, your customers receive an invoice or receive a credit card or debit card transaction.
- **When activity occurs.** For accounting cycles, activities, such as recording balance impacts into usage items, occur *during* the accounting cycle. For billing cycles, activity occurs only at the *end* of the billing cycle when BRM finalizes a bill and requests a payment from the customer.

About the First Cycle Forward Fee in a New Account

A cycle forward fee incurred at registration is included in the cycle forward item. Therefore, the first bill includes two cycle forward fees, as shown in [Figure 1–2](#).

Figure 1–2 Cycle Forward Fees for New Account



The first cycle forward fee affects other aspects of billing:

- How revenue is reported in general ledger reports. See "About Unbilled/Unearned Cycle Forward Fees" in *BRM Collecting General Ledger Data*.

About Accounting Cycles

By default, an accounting cycle always ends at midnight, specifically at 23:59:59, and the next accounting cycle always begins at 00:00:00. BRM performs various tasks at the end of one accounting cycle and the beginning of the next accounting cycle:

At the end of an accounting cycle, BRM performs these tasks:

- Applies balance impacts from deferred cycle forward events.
- Applies balance impacts from cycle discount events.
- Applies balance impacts for fold events. For example, if a product uses fold events to remove unused free hours, the fold events are rated and the balance impacts are applied at the end of the accounting cycle.
- Creates one or more cycle forward items, one for each service that the customer owns. The cycle forward items include any cycle forward balance impacts that apply to the following month. The cycle forward items have a status of pending.

- Applies balance impacts from cycle rollover events.
- Applies balance impacts from cycle arrears events to the current usage item.
- Applies balance impacts from cycle forward arrears events to the next cycle's cycle forward arrears item.

At the beginning of a new accounting cycle, BRM performs these tasks:

- Creates a usage item. Balance impacts from usage fees, cancel fees, and purchase fees are added as they occur. The usage item has a status of pending.
- Creates pre-created items for each service that is specified in the `/config/item_tags` and `/config/item_types` storable objects. These items have a status of pending.

If the account uses a multimonth billing cycle, new cycle forward and usage items are created every month, resulting in multiple cycle forward and usage items.

About Accounting Cycle Dates

By default, an accounting cycle begins on the day of the month that the customer registers. In BRM, this day is called the accounting day of month (DOM).

For example, if the customer registers on the 15th, the accounting cycle starts on the 15th day of the month. The end of the accounting cycle will depend on when the accounting cycle began. If the accounting cycle began on a date that is common to all months, it will end on the same date. If the customer registers on the 29th, 30th, or 31st of the month, for months that do not have the date, the accounting cycle will end on the first day of the month following the next month. For example, if the customer registers on January 31, the accounting cycle will begin on January 31. Because February does not have 31 days, the accounting cycle will end on March 1 by default.

Although an accounting cycle is always one month long, the length of the accounting cycle changes from month to month. For example, if an accounting cycle starts on the 15th day of the month, there are more days between January 15 and February 15 than there are between February 15 and March 15.

You can change the default accounting cycle date for all accounts to a single date, but that can result in an excessive load on the BRM system. See ["Setting the Default Accounting Day of Month \(DOM\)"](#).

To assign an account to an accounting DOM other than the default, see ["About Managing Billing Cycles"](#).

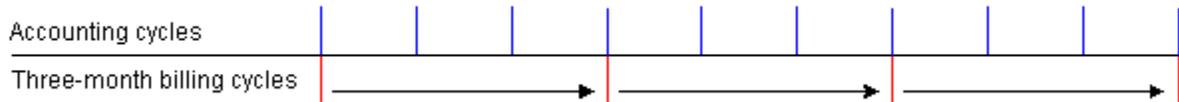
About Billing Cycles

Billing cycles consist of a billing DOM and a billing frequency:

- The billing DOM specifies the date on which BRM finalizes a bill and requests payment from the customer. The billing DOM is determined by the bill unit's billing segment, the DOM of the account's other bill units, the default setting in the Connection Manager (CM) configuration file, or the current date. For information, see ["How BRM Sets the Billing DOM"](#).
- The billing frequency specifies how often to finalize a bill and request payments from customers. The billing cycle length is any whole multiple of the accounting cycle. For example, a monthly billing cycle corresponds to one accounting cycle, and a quarterly billing cycle corresponds to three accounting cycles ([Figure 1-3](#)). To change the default billing cycle length, see ["Setting the Default Billing-Cycle Length"](#).

Accounting and Billing Cycles

Figure 1–3 One Month Accounting and Three Month Billing Cycles



The billing DOM and billing frequency are set at the bill unit (`/billinfo` object) level. Accounts with multiple bill units can have different billing cycle settings for each bill. For example, an account with two bill units can have:

- One bill unit with a billing DOM of 5 and a monthly billing frequency.
- One bill unit with a billing DOM of 15 and a quarterly billing frequency.

Note: Child bill units must have the same billing DOM and billing frequency as their parent bill unit.

How BRM Sets the Billing DOM

BRM sets a bill unit's billing DOM based on the following order of priority:

1. **The DOM assigned to the billing segment.** BRM assigns the DOM set for the billing segment in the `/config/billing_segment` object. For more information, see ["Assigning DOM Based on the Billing Segment"](#).

Note: To customize how BRM assigns the DOM according to the billing segment, see ["Customizing the DOM Assignment Process"](#).

2. **The DOM used by the first bill unit in the account.** If a DOM is not assigned to the billing segment, BRM sets the DOM to that of the first bill unit in the account.
3. **Default setting in the CM `pin.conf` file.** If a DOM is not assigned to the billing segment nor is available from another bill unit, the DOM is set to the value assigned in the `actg_dom` entry in the CM configuration file (`BRM_home/sys/cm/pin.conf`, where `BRM_Home` is the directory in which BRM components are installed). To set the default value, see ["Setting the Default Accounting Day of Month \(DOM\)"](#).
4. **The current date.** If a DOM is not available from the billing segment, other bill units, or the CM `pin.conf` file, BRM assigns the DOM to the current date.

For example, if an account was created with two bill units, **BU1** that has an assigned DOM and **BU2** that does not have a DOM assigned, BRM would assign a DOM to **BU2** as follows:

- If **BU2** is the second bill unit in the account, BRM sets the DOM to that of **BU1**.
- If **BU2** is the first bill unit in the account, BRM sets the DOM to the value in the CM `pin.conf` file. If a value is not set in the CM `pin.conf` file, the DOM is set to the current date.

About Actions Performed at the End of a Billing Cycle

Typically, at the end of a billing cycle, BRM performs the following:

- Calculates deferred taxes, if any, and applies them as balance impacts. See "Choosing When to Calculate Taxes" in *BRM Calculating Taxes*.
- Changes the status of the bill items associated with each bill unit to open. Therefore, no further balance impacts are added to those items, and BRM can request payments for those items.
- Finalizes a bill for each bill unit in the BRM database. A bill includes balance impacts from the bill items. If the billing cycle includes more than one accounting cycle, the bill includes balance impacts from multiple bill items. For example, a quarterly bill includes balance impacts from three usage items and at least three cycle forward items.
- Depending on the payment method, either requests a BRM-initiated payment (either credit card or direct debit) or creates an invoice for the bill. For some payment methods, such as Undefined, BRM makes no payment request. See "About Payment Methods" in *BRM Configuring and Collecting Payments*.

Collecting payments does not occur automatically at the end of a billing cycle. You must set up billing applications that automatically request payments at the end of an account's billing cycle. See "[About the Billing Utilities](#)".

About Auto-Triggered Billing

By default, auto-triggered billing is *always* enabled.

Note: If you use delayed billing, auto-triggered billing is always enabled for the delay period. And by default, it is also enabled for the last two days *only* of each bill unit's accounting cycle. For more information about auto-triggered billing during delayed billing, see "[About Delayed Billing](#)".

When auto-triggered billing is enabled, BRM automatically triggers billing when an event occurs *after* the billing date but *before* billing is run. For example:

1. An account's billing date is May 15, and billing will be run for the account on May 15 at 02:00:00.
2. On May 15 at 01:00:00: one hour after the end of the previous billing cycle but one hour before that billing is run: a usage event associated with the account occurs. This usage event belongs to the next billing cycle.
3. To ensure that the usage event is recorded in the correct billing period, BRM immediately performs the billing processes: for example, changes item status to open: and finalizes a bill for the account.
4. The event that triggered billing is included in the items for the next bill.

Note: Auto-triggered billing performs only the operations that the `pin_bill_accts` utility normally performs. It does not do any payment requests; those are done when you run the `pin_collect` utility.

Events that trigger billing include the following:

- Purchasing a product
- Changing account status

- Canceling a product
- Rating a usage event

Note: The event does not need to have a balance impact to trigger billing.

You can disable auto-triggered billing when, for example, you might want to run billing only by running the billing application (`pin_bill_accts`). See "[Disabling Auto-Triggered Billing](#)".

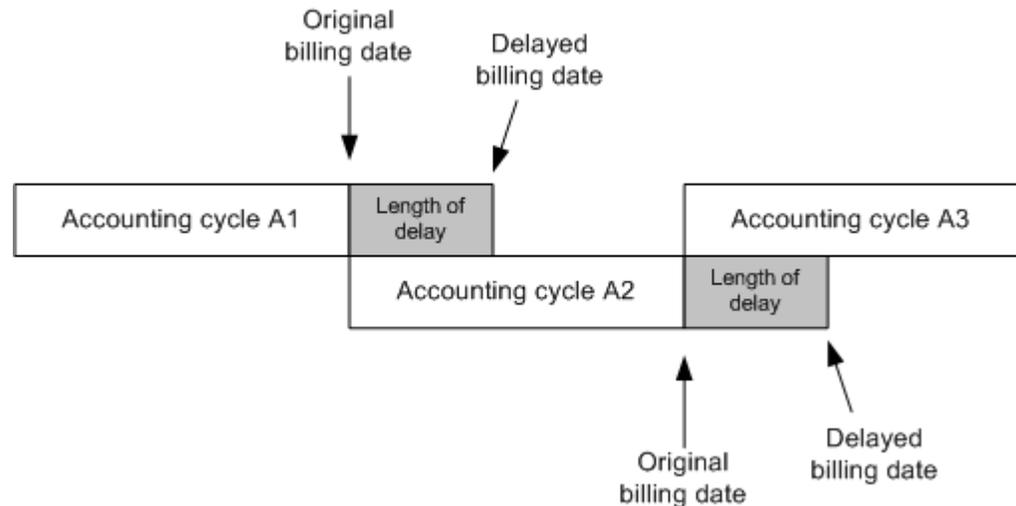
About Delayed Billing

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 you put events in batches to be recorded later but do not process those batches 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.

[Figure 1–4](#) shows that with delayed billing, the billing date occurs after the original billing date, during the next accounting cycle.

Figure 1–4 *Delayed Billing Timeline*



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

For information about setting up delayed billing, see "[Setting Up Delayed Billing](#)".

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 Customer Center Balance tab. The combined pending bill created during the delay period includes separate items for the previous accounting cycle and for the

current 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.

When delayed billing is used, the **/billinfo** object might be billed twice, once during the delay interval and again after the delay interval.

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 only 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:

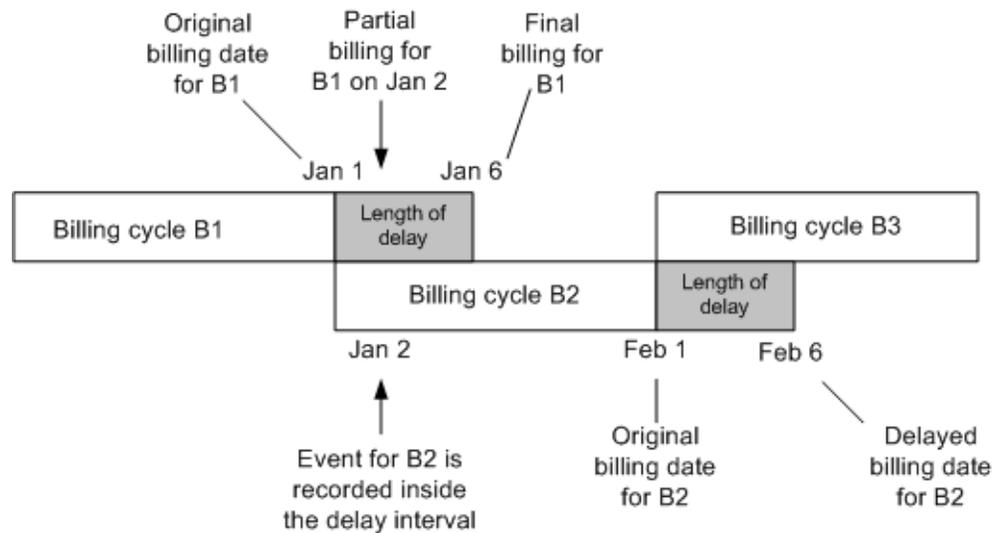
- Applies deferred cycle forward fees to the next billing cycle.
- Applies cycle arrears fees to the previous billing cycle.
- Applies cycle forward fees.
- Applies cycle rollovers.

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 1-5](#):

- 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 1–5 Partial and Final Billing Timeline

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 auto-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 **auto_triggering_limit** parameter to 0. See "[Specifying Auto-Triggered Billing for Delayed Billing](#)".
 - You can also change the number of days auto-triggered billing is *enabled* at the *end* of each accounting cycle. See "[Specifying Auto-Triggered Billing for Delayed Billing](#)".
-
-

For more information, see:

- [About Auto-Triggered Billing](#)

- [Billing Cycle Override](#)

About Delayed Billing and Changing the Billing DOM

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. See "[About Auto-Triggered Billing](#)".

About On-Demand Billing

Usually, you bill a customer at the end of the customer's billing cycle. However, you can bill a customer immediately for a purchase, even if the customer's billing cycle has not ended. To perform on-demand billing, you create a deal or plan and flag it for on-demand billing. A bill for purchase fees only is created and finalized as soon as the plan or deal is purchased.

If the purchase is for a parent **/billinfo**, a **/bill** object is created with just that parent's purchase total. If the purchase is for a subordinate **/billinfo**, the parent receives the bill, but it includes only the subordinate **/billinfo** total.

Important: An on-demand bill includes only purchase fees. To create a bill that includes all the customer's pending charges, use the Bill Now feature in Customer Center. See "[About Bill Now](#)" for more information.

See the following related topics:

- For information about the opcodes used for on-demand billing, see "[How On-Demand Billing Works](#)".
- For information about configuring on-demand billing in a price list, see "Using Deals to Bill Your Customers on Demand" and "Using Plans to Bill Your Customers on Demand" in *BRM Setting Up Pricing and Rating*.
- For information about using on-demand billing in Pricing Center, see the discussion about enabling on-demand billing in the Pricing Center online help.

About Bill Now

By default, the Bill Now feature in Customer Center generates a bill that includes all pending items, and any cycle arrears and cycle forward arrears fees for an account or a specified bill unit. Therefore, for both *open item accounting* and *balance forward accounting* types, Bill Now adds the previous total amount to the current total. For example, an account with open item accounting bill type has a previous total of \$20 and its current total is \$10. When Bill Now is run, the account is billed for \$30.

When an account has multiple bill units (**/billinfo**), only one parent bill unit can be processed at a time.

For example, consider the following bill hierarchy:

```
Parent Account
|
-- Bill 1 (nonsubordinate bill unit)
|
-- Child Account 1 (subordinate bill unit)
-- Bill 2 (nonsubordinate bill unit)
|
-- Child Account 2 (subordinate bill unit)
```

Bill 1 and Bill 2 are parent bill units. To generate the bills for both bill units, run Bill Now sequentially to process one bill unit at a time. You can select items that belong to either Bill 1 or Bill 2. If you select items that belong to different parent bill units, Bill Now reports an error.

Important: If you run Bill Now on a subordinate **/billinfo**, a bill is created for the parent **/billinfo** that includes only the subordinate **/billinfo** items. If you run Bill Now on a parent **/billinfo**, a bill is created that contains a total of the items from both the parent and any subordinate **/billinfo** objects.

You can also 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. Unbilled items will continue to be displayed under Bill in Progress until they are billed.

To customize which pending items are included during billing, edit the search parameters in the PCM_OP_BILL_POL_GET_PENDING_ITEMS policy opcode. Optionally, you can also include billing-time discounts and folds in the bill. For more information, see ["Customizing Bill Now"](#) and ["Applying Discounts and Folds with Bill Now"](#).

Important: If you run Bill Now on a subordinate **/billinfo** and select specific pending bill items, a bill is created for the parent **/billinfo** that includes only the selected pending items from the subordinate **/billinfo** items. If you run Bill Now on a parent **/billinfo** and select specific pending bill items, a bill is created that contains the selected pending items from both the parent and any subordinate **/billinfo** objects.

In your customer relationship management (CRM) tool, you can use Bill Now to bill for a specified service. See ["Running Bill Now for a Service"](#) and PCM_OP_BILL_CREATE_SPONSORED_ITEMS.

If you use delayed billing, you can use Bill Now to create two bills using your CRM tool during the delayed billing period. See ["Creating Two Bills during the Delayed Billing Period"](#).

For more information on Bill Now, see the following:

- [How Bill Now Works](#)
- [Configuring Bill Now](#)

- About Billing upon Subscription Service Cancellation in *BRM Managing Customers*

About Accounting Types

BRM bills include the charges incurred during the current billing cycle and, optionally, any unpaid charges from previous billing cycles. You control whether BRM bills include charges from previous billing cycles by setting the accounting type:

- With *open item accounting*, a customer is billed only for charges from the bill items in the current bill. If a customer does not pay a bill, the next bill does not include charges for the bill that the customer did not pay.

You typically use open item accounting for non-credit card accounts, where a customer receives an invoice. Each invoice includes the items that apply to a single billing cycle. If a customer does not pay a bill, the customer still has the invoice for the old bill when the customer receives the next invoice.

- With *balance forward accounting*, a customer's bill includes all the charges that a customer owes, including those from previous billing cycles. If a customer does not pay a bill, the next bill includes the charges from the previous bill.

Accounts for customers who pay by credit card should always use balance forward accounting. Balance forward accounting is the default.

Accounting types are set at the bill unit (**/billinfo** object) level rather than at the account level. This enables accounts with multiple bill units to have different accounting type settings for each bill. For example, an account with two bill units can have one bill unit with an open item accounting type and another bill unit with a balance forward accounting type.

How BRM Determines the Accounting Type

BRM determines a bill unit's accounting type by reading and using the following accounting type settings in the order shown below:

1. **Customer Center or opcode flist setting.** You can specify a bill unit's accounting type when you create or modify an account in Customer Center.

If you use a custom client application, you can specify a bill unit's accounting type by passing it in the PIN_FLD_ACTG_TYPE input flist field of the following opcodes:

- PCM_OP_CUST_COMMIT_CUSTOMER
- PCM_OP_CUST_MODIFY_CUSTOMER
- PCM_OP_CUST_UPDATE_CUSTOMER

2. **Default setting in the Connection Manager (CM) configuration file.** You can specify a default accounting type by using the **actg_type** entry in the CM configuration file (*BRM_Home/sys/cm/pin.conf*). BRM uses this setting during account creation for all accounts and bill units only if an accounting type is not passed in the input flist. To set the default value, see "[Setting the Default Accounting Type](#)".

3. **System wide accounting type.** If an accounting type is not specified using any of the above settings, BRM automatically sets the bill unit's accounting type to balance forward accounting.

About Changing a Bill Unit's Accounting Type

A bill unit's accounting type can be changed at any time. However, BRM does not validate the change nor take any actions other than changing the accounting type in the next billing cycle. You must ensure that the impact of any accounting type changes do not confuse your customers. For example:

- If the accounting type is changed from open item accounting to balance forward accounting, the customer's next bill would include all open and unpaid items. Your customer should be informed that the bill now includes any past due charges from previous billing cycles.
- If the accounting type is changed from balance forward accounting to open item accounting, the customer's next bill would not include any unpaid items. Your customer should be informed that charges from previous bills are still past due, even though they do not appear on the current bill.

About Bill Items

Storing information in bill items enables you to make adjustments to the customer's balance due.

BRM tracks accounts receivable in these types of bill items:

- **Cycle forward items** track the accounts receivable for cycle forward fees. See "About Cycle Forward Events" in *BRM Setting Up Pricing and Rating* for more information about the events that these items represent.
- **Cycle arrears items** track the accounts receivable for cycle arrears fees. See "About Cycle Arrears Events" in *BRM Setting Up Pricing and Rating* for more information about the events that these items represent.
- **Cycle forward arrears items** track the accounts receivable for cycle forward arrears fees. See "About Cycle Forward Arrears Events" in *BRM Setting Up Pricing and Rating* for more information about the events that these items represent.
- **Usage items** track the accounts receivable for usage fees, purchase fees, and cancel fees. These fees are stored in the **/item/misc** storable object.

An **/item/misc** object is created for each account and for every service that the account owns. This enables you to manage fees for each service independently; for example, you can display the usage fees for separate telephony services.

- **Custom items** track the accounts receivable for customized bill items you create. See "[About Custom Bill Items](#)".

For more information on bill items and how you use them to manage accounts receivables, see "About Accounts Receivable" in *BRM Managing Accounts Receivable*.

About Multiple Bills per Cycle

An account's billing information is stored in a bill unit (a **/billinfo** object in the BRM database). The bill unit associates each account balance group with a bill and a payment method. When you bill accounts, a bill is produced for every bill unit.

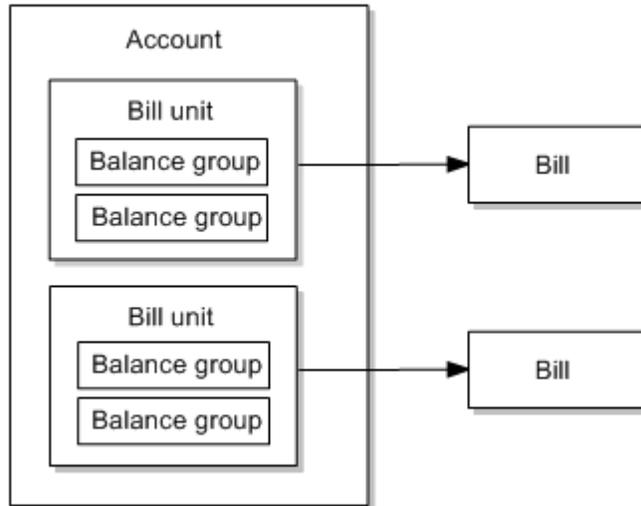
You can create multiple bill units for a single account. Each bill unit in an account has its own payment method, accounting type, billing DOM, and billing frequency.

By default, accounts are created with one bill unit. You create additional bill units by using Customer Center. See the Customer Center Help.

You can also create bill units by implementing BRM opcodes in your custom code. Specify the account to which the bill unit belongs and link the account balance groups to the new `/billinfo` object. See ["Managing Bill Units with Your Custom Application"](#).

When an account has multiple bill units, a bill is produced for each bill unit in the account, as shown in [Figure 1–6](#).

Figure 1–6 Multiple Bill Unit Account



You perform the following for each bill unit in an account:

- Associate a payment method, such as credit card, direct debit, or invoice. With multiple bill units, accounts can be billed for services separately, using a different payment method for each bill.
- Specify the billing DOM.
- Specify the billing frequency.
- Specify the accounting type: open item accounting or balance forward accounting.

About Billing Cycles for Hierarchical Accounts with Multiple Bill Units

A child account is not required to use the same billing cycle as its parent account. However, if the child account has a subordinate (nonpaying) bill unit, that bill unit must have the same billing DOM and billing frequency as the parent account. Paying bill units in the child account that are paid by the child account use the child account's billing DOM and cycle.

For more information, see "Creating Hierarchical Bill Units" in *BRM Managing Accounts Receivable*.

How BRM Creates a Bill

The `PCM_OP_BILL_MAKE_BILL` opcode creates a `/bill` object. `PCM_OP_BILL_MAKE_BILL` does the following:

- Applies cycle fees.
- Totals pending items in the `/bill` current total field.
- Totals open items in the `/bill` previous total field.

- If rollover correction is enabled, can trigger the creation of rerating requests at the end of the delayed period if call detail records (CDRs) from the previous cycle borrow rollover from the current cycle. See ["Enabling Rerating and Rollover Correction Due to Delayed Events"](#).

To trigger the creation of rerating requests, this opcode can create a notification event of type `/event/notification/rollover_correction/rerate` for the account being billed and possibly other accounts from which a line was transferred into the account. Depending on how automatic rerating is configured, the notification event triggers the creation of rerating requests.

- If rollover correction is enabled, triggers rollover correction if CDRs borrow from the previous cycle borrow rollover from the current cycle. See ["Enabling Rerating and Rollover Correction Due to Delayed Events"](#).
- If payment incentives are enabled, calls the `PCM_OP_PYMT_GRANT_INCENTIVE` opcode, which determines whether to grant the payment incentive and also calculates the incentive. `PCM_OP_BILL_MAKE_BILL` then retrieves the payment incentives and recalculates the affected bill, changing the balance impacts accordingly. See ["Configuring Payment Incentives"](#) in *BRM Configuring and Collecting Payments*.
- To determine the payment due date of a bill, calls the `PCM_OP_BILL_POL_CALC_PYMT_DUE_T` policy opcode. See ["How BRM Calculates Bill Due Dates"](#).
- At the end of the last accounting cycle in a bill unit's billing cycle, `PCM_OP_BILL_MAKE_BILL` calls the `PCM_OP_BILL_POL_CHECK_SUPPRESSION` policy opcode to find out whether a bill should be suppressed.

If the output flist of the policy opcode indicates that the bill *should* be suppressed, performs these tasks:

- Generates an `/event/billing/suppression` event.
- Extends the bill for another billing cycle instead of finalizing it.

If the output flist of the policy opcode indicates that the bill *should not* be suppressed, performs these tasks:

- If the output flist includes an exception to bill suppression, generates an `/event/billing/suppression/exception` object and then finalizes the bill. (See ["Exceptions to Bill Suppression"](#).)
- If the result does not include an exception, finalizes the bill.

Whether or not the output flist of the policy opcode indicates that the bill should be suppressed, `PCM_OP_BILL_MAKE_BILL` *always* subtracts 1 from the `/billinfo` counter that tracks the remaining cycles for which a bill has been manually suppressed (`PIN_FLD_SUPPRESSION_CYCLES_LEFT`) if the value in the counter is greater than 0.

See ["How Bill Suppression Works"](#).

- When you use delayed billing, `PCM_OP_BILL_MAKE_BILL` performs some functions on the billing DOM and performs the rest of the functions at the end of the delay interval. See ["About Delayed Billing"](#).
- Calls the `PCM_OP_SUBSCRIPTION_CALC_BEST_PRICING` opcode to calculate the best price after applying all the charges and discounts and before applying the billing-time tax.

If the best pricing calculation is successful, `PCM_OP_BILL_MAKE_BILL` finalizes the bill, taking into account the use of any alternate deal.

If the **/billinfo** object being billed is a parent, PCM_OP_BILL_MAKE_BILL creates a single **/bill** object for that parent, which includes all pending and open items from its subordinate **/billinfo** objects.

About Bill Numbering in BRM

By default, bill numbers for regular bills are **B1-1**, **B1-2**, **B1-3**, and so on. And for corrective bills, the default numbering is **CB1-1**, **CB1-2**, **CB1-3**, and so on.

The default numbering takes the format *Prefix-Seq_Number*, where:

- *Prefix* represents the prefix to be used for the bill.
For regular bills, BRM assigns **BDB_num** as the default prefix. *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**.
For corrective bills, BRM assigns **CBDB_num** as the default prefix. For example, when *DB_num* is 1, 3, or 7, the prefixes for corrective bills are **CB1**, **CB3**, or **CB7**.
- *Seq_Number* is the unique sequence number from the appropriate **/data/sequence** object. BRM supports two types of **/data/sequence** objects, (PIN_SEQUENCE_TYPE_BILL_NO and PIN_SEQUENCE_TYPE_CORR_BILL_NO).

Note: Of these two **/data/sequence** objects, PIN_SEQUENCE_TYPE_CORR_BILL_NO is used exclusively for corrective bills.

For regular bills, BRM uses the unique sequence number from the PIN_SEQUENCE_TYPE_BILL_NO **/data/sequence** object.

For corrective bills, you can select to use the unique sequence number from either of the two **/data/sequence** objects, (PIN_SEQUENCE_TYPE_BILL_NO and PIN_SEQUENCE_TYPE_CORR_BILL_NO).

A regular bill and all its corrective bills have the same POID (the bill object identifier in the BRM database). As a result, you can use the bill POID to retrieve the complete set of bills, that is, the corrected bill and its prior bills.

You can customize bill numbers in BRM as described in "[Setting Business Policies for Billing](#)".

How Bill Now Works

The PCM_OP_BILL_MAKE_BILL_NOW opcode creates a bill for a specified **/billinfo** object immediately from Customer Center. If a **/billinfo** object is not specified, this opcode creates one **/bill** for each **/billinfo** for the given account.

The PIN_FLD_NAME field in the **/bill** object contains the type of billing the **/bill** object is for:

- Regular billing
- Billing on demand
- Bill Now
- Bill Now for the current cycle
- Bill Now on the next cycle

The last two options enable creating two bills during the delayed period if your customer management system (CMS) supports that functionality. One bill is generated for the current cycle charges; the other is generated for the next cycle charges.

PCM_OP_BILL_MAKE_BILL_NOW performs the following tasks:

- Calls the subscription management opcodes to apply cycle forward, cycle arrears, and cycle forward arrears fees.

If called for a service of a sponsored account, PCM_OP_BILL_MAKE_BILL_NOW calls the PCM_OP_BILL_CREATE_SPONSORED_ITEMS opcode. The bill produced is for the pending items for the specified service of the sponsored account.

- If called on a subordinate **/billinfo**, creates one bill for the parent **/billinfo** that only includes the items from the subordinate **/billinfo**. In such cases, the PIN_FLD_GROUP_OBJ field in the **/event/billing/cycle/tax** object contains the POID of the subordinate **/billinfo**. If called on a parent **/billinfo**, creates one bill that contains a total of the items from both the parent and any subordinate **/billinfo** objects. This includes any subordinate cycle taxes. In such cases, the PIN_FLD_GROUP_OBJ field contains the POID of the parent **/billinfo**.
- Calls the PCM_OP_SUBSCRIPTION_CALC_BEST_PRICING opcode to calculate the best price after applying all charges and discounts and before applying the billing-time tax. Best pricing operation is performed for the period between the start of the billing cycle to the time billing is run.

If you choose to bill only a subset of the pending items by using the PCM_OP_BILL_POL_GET_PENDING_ITEMS policy opcode, the best pricing calculation is performed for all the items. However, the bill contains only the selected items with the balance based on the alternate deals.

If the best pricing calculation is successful, this opcode finalizes the bill, taking into account the use of any alternate deal.

- If configured, PCM_OP_BILL_MAKE_BILL_NOW calls the PCM_OP_BILL_CYCLE_TAX opcode to calculate taxes. See "Calculating Taxes during Billing" in *BRM Calculating Taxes*.

Note: Bill Now ignores the **cycle_tax_interval** value in the CM's configuration file (**pin.conf**) and always rolls activities for each subordinate **/billinfo** into the parent **/billinfo** and calculates taxes for the parent only.

- If configured, Bill Now prorates cycle arrears and cycle forward arrears fees. See "[Prorating Cycle Arrears and Cycle Forward Arrears for Bill Now](#)".

Bill Now does not generate invoices. You must separately run the **pin_inv_accts** utility or the **pin_bill_day** script. For more information about invoices, see "How Invoices Are Generated" in *BRM Designing and Generating Invoices*.

Several options control how Bill Now works. See "[Configuring Bill Now](#)".

How On-Demand Billing Works

The PCM_OP_BILL_MAKE_BILL_ON_DEMAND opcode creates a **/bill** object immediately when a deal or plan that is flagged for on-demand billing is purchased.

To create a bill on demand for a deal or plan, the PCM_FLD_ON_DEMAND_INFO field must be set in /plan or /deal objects. Set this field using Pricing Center by selecting **Bill on Demand** on the plan or deal Attributes tab. For information about using on-demand billing see the discussion about enabling on-demand billing in the Pricing Center online help.

Billing for Sponsorship

If your BRM system is setup for sponsorship and contains sponsor or resource sharing groups, you must reconfigure your billing setup to ensure that member accounts are billed before owner accounts. See "[Setting Up Billing for Sponsorship](#)".

Caution: If you do not reconfigure billing for sponsorship, members' sponsored charges might not be included in their owner's bill for the current billing cycle. Instead, they are added to the owner's bill for the next billing cycle.

Related Documents

For more information about billing, see the following:

- [Setting Business Policies for Billing](#)
- [Running Billing Utilities](#)
- [About the Billing Utilities](#)
- "About BRM-Initiated Payment Processing" in *BRM Configuring and Collecting Payments*
- "Designing and Generating Invoices" in *BRM Designing and Generating Invoices*
- [Creating Custom Bill Items](#)
- [About Proration](#)

About Corrective Billing

This chapter describes the support for corrective billing in Oracle Communications Billing and Revenue Management (BRM).

Before reading this chapter, read "[About Billing](#)".

About Bill Corrections

Corrections to a bill become necessary when a bill or invoice sent to a customer requires an update. Corrections to a bill may or may not affect the charges on a bill.

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. The corrective bill includes all accounts receivable (A/R) actions for the charges applied to the bill items. BRM does not apply any additional discounts or charges.

When a corrective bill is generated, it is assigned a new due date based on the current payment setup for the account.

A corrective 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 corrective invoices, see *BRM Designing and Generating Invoices*.

About Simple Corrections to Bills

Simple corrections to a bill result 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 do not result in a change to the amount owed by the customer, irrespective of the method of payment.

About Corrections to Charges in Bills

In BRM, corrections to the charges on a bill result from manual or automatic adjustments (debits or credits), for example, settled disputes that require some item adjustment, catalog corrections such as retroactive tariff updates, other automatic adjustments, and so on. All the A/R actions included in the corrective bill impact the totals or balances for that bill.

You can create a series of corrective bills for a bill. For example, you generate a corrective bill CB-57 for a regular bill B1-124. Subsequently, you can generate a corrective bill CB-89 for corrections to CB-57, CB-104 for corrections to CB-89, and so on.

About Billing Customers

To bill customers, you use 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. Alternately, you can generate corrective bills by using an external application that BRM supports, such as Seibel's Customer Relationship Management (CRM) application. To do so, you must customize the external application. See "[Generating Corrective Bills](#)".

Note: The corrective billing process in BRM does not support the use of billing scripts associated with regular billing, namely, **pin_bill_day**, **pin_bill_week**, and **pin_bill_month**.

About Corrective Bills

BRM can generate corrective bills for:

- Original bills
- Bills created using the Bill Now selection in Customer Center
- On-demand billing
- Parent and subordinate bills in a bill hierarchy
- A prior corrective bill. That is, you can correct a subsequent corrective bill to address the errors on a prior corrective bill.

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. For a description of a finalized bill, see *BRM Glossary*.

About Corrective Billing in Subordinate Hierarchies

When there are correction to charges for a bill in a hierarchy, BRM does the following:

- It accepts requests to generate a corrective bill for the parent bill in a hierarchy even when it is a subordinate bill that needs the correction. This is because any adjustment applied to a subordinate bill immediately impacts the bill and bill items for the parent bill.

Note: If you submit a request to generate a corrective bill for a subordinate bill in a hierarchy, BRM rejects the request.

- When you request a corrective bill for the parent bill in a hierarchy, BRM creates corrective bills for the parent bill and all the subordinate bills in the hierarchy. The corrective bill includes all A/R actions applied up to that point in time for the parent bill and all subordinate bills.
- When BRM generates a corrective bill for the parent bill in a hierarchy,
 - It assigns a new bill number to the parent bill and to all the subordinate bills in a hierarchy.
 - It assigns a new bill due date to the billable items in the parent bill and to the billable items in all the subordinate bills in the hierarchy.
 - It maintains bill history of all the subordinate bills in the hierarchy.

About Generating and Viewing Corrective Bills in BRM

Corrective billing is an optional feature in BRM. When you initially install BRM, it enables you to generate regular bills and invoices only. By default, BRM does not generate corrective bills and invoices.

To generate corrective bills and corrective invoices, you must enable the **EnableCorrectiveInvoices** business parameter in the **billing** instance of the `/config/business_params` object. See ["Enabling Corrective Billing in BRM"](#) for more information.

Regulating Permissions for CSR Roles

By default, BRM provides all CSRs with permission to create corrective bills. When the CSR role has permission to generate corrective bills, Customer Center enables the **Produce Corrective Bill** entry as a selection item in the drop-down menu for **Actions** in the **Balance** tab. See *BRM Customer Center Online Help* for more information.

Validating Bills for the Corrective Billing Process

The corrective billing process uses the `PCM_OP_BILL_MAKE_CORRECTIVE_BILL` opcode to provide standard validations. This opcode calls the `PCM_OP_BILL_POL_VALID_CORRECTIVE_BILL` policy opcode to complete the policy validations on a selected bill.

For more information on the `PCM_OP_BILL_MAKE_CORRECTIVE_BILL` opcode and the `PCM_OP_BILL_POL_VALID_CORRECTIVE_BILL` policy opcode, see *BRM Developer's Reference*.

Standard Validations Performed by BRM

`PCM_OP_BILL_MAKE_CORRECTIVE_BILL` checks to see whether you enabled corrective billing and whether the corrective bill is for a finalized bill. If the bill is part of a hierarchy, it checks whether you are requesting a corrective bill for the parent bill. The opcode returns an error if you attempt to generate a corrective bill for the subordinate bill in a hierarchy.

Policy Validations Performed by BRM

By default, the `PCM_OP_BILL_POL_VALID_CORRECTIVE_BILL` policy opcode checks the bill and returns its findings to `PCM_OP_BILL_MAKE_CORRECTIVE_BILL`. Then, `PCM_OP_BILL_MAKE_CORRECTIVE_BILL` continues or terminates its process based on the output from the `PCM_OP_BILL_POL_VALID_CORRECTIVE_BILL` policy opcode. For more information, see `PCM_OP_BILL_POL_VALID_CORRECTIVE_BILL` in *BRM Developer's Reference*.

The checks performed by the `PCM_OP_BILL_POL_VALID_CORRECTIVE_BILL` policy opcode are as follows:

- The policy opcode verifies that an invoice exists for the (original or corrective) bill for which you are attempting to generate a corrective bill. If there is no invoice associated with that bill, the policy opcode does not permit the generation of a corrective bill. It enters an error message in the `PIN_FLD_ERROR_DESCR` field and stops any further validation on the bill.
- If there are no A/R charges for the prior bill, the policy opcode verifies that it is creating only a replacement invoice for that bill. If a replacement invoice is not being generated for the corrective bill, the policy opcode enters an error message in the `PIN_FLD_ERROR_DESCR` field and stops any further validation on the bill.

- When balance forward accounting is associated with a bill, the policy opcode verifies that the corrective bill is being generated for the last bill in the billing cycle. If this validation fails, the policy opcode enters a PIN_BILL_VALIDATION_BILL_NOT_LAST flag in the PIN_FLD_FLAGS field. (For open item accounting, BRM permits the generation of corrective bills for any previous cycle.)
- The policy opcode verifies whether you enabled a corrective bill to be generated when there is a payment to be processed against that bill (see "[Generating Corrective Bills for Partially or Fully Paid Bills](#)"). If this validation fails, the policy opcode enters a PIN_BILL_VALIDATION_NOT_PAID flag in the PIN_FLD_FLAGS field.
- The sum of the corrections must equal or exceed the specified amount threshold necessary to create the corrective bill. If this validation fails, the policy opcode enters a PIN_BILL_VALIDATION_AR_TOO_LOW flag in the PIN_FLD_FLAGS field. See "[Specifying the Threshold Amount for Corrective Bills](#)" for more information.

You can customize the policy validations for corrective bills. See "[Setting Up Custom Validations for Corrective Bills](#)" for more information.

About Numbering Corrective Bills in BRM

BRM assigns a new bill number to each corrective bill that it generates. See "[About Bill Numbering in BRM](#)" for information on bill numbering. If the corrective bill belongs to a hierarchy, BRM assigns the new bill number to all the bills in the hierarchy.

By default, corrective bill numbering is enabled in BRM. That is, when a corrective bill is generated for a prior bill with B1-189 as its bill number, BRM assigns CB1 as the prefix and uses PIN_SEQUENCE_TYPE_CORR_BILL_NO to obtain the sequence number currently available for corrective bills in BRM, (for example, 15). The bill number for the corrective bill is set to CB1-15. For subsequent corrections, BRM retains the prefix from the previous corrective bill and assigns the next available sequence number associated with PIN_SEQUENCE_TYPE_CORR_BILL_NO.

If corrective bills are to retain the prefix and sequence number available for *regular* bills, you change the default value by setting the **GenerateCorrectiveBillNo** business parameter to **disabled** in the `/config/business_params` object.

When you change the default setting for the **GenerateCorrectiveBillNo** business parameter, BRM retains the original bill's prefix and uses PIN_SEQUENCE_TYPE_BILL_NO to obtain the sequence number currently available for regular bills in BRM. For example, if PIN_SEQUENCE_TYPE_BILL_NO yields 2007 as the next available sequence number in BRM, the bill number for the corrective bill for a prior bill with B1-189 as its bill number is set to B1-2007.

For subsequent corrections, BRM retains the prefix from the previous corrective bill and assigns the next available sequence number associated with PIN_SEQUENCE_TYPE_BILL_NO.

You can also set up custom numbering for corrective bills.

See "[Setting Up Bill Numbers for Corrective Bills](#)" for more information.

About Maintaining a Bill's History

BRM preserves the historical information for every bill that it corrects by maintaining a `/history_bills` object for each corrected bill.

Before BRM generates a corrective bill for a regular bill, **Bill Now**, or **Bill On Demand**, it saves a copy of the regular or corrective bill as a **/history_bills** object in the BRM database. If the bill being corrected is the parent bill in a hierarchy, BRM stores copies of the parent bill and all subordinate bills in that hierarchy as **/history_bills** objects.

[Table 2-1](#) shows an example of a bill and its corrective bill objects in the BRM database. The original bill is numbered B1-1234, and its corrections are CB1-207 and CB1-226.

Table 2-1 Objects Stored by BRM for Corrective Billing

Action Performed	Bill Object Stored	Bill History Object Stored
Finalize a bill	B1-1234	None
Create First Corrective Bill	CB1-207	The following entries are stored: <ol style="list-style-type: none"> 1. B1-1234 2. CB1-207
Create Second Corrective Bill for Bill	CB1-226	The following entries are stored: <ol style="list-style-type: none"> 1. B1-1234 2. CB1-207 3. CB1-226

An account's original bill and all of its corrective bills have the same POID. For more information on the **/history_bills** object, see *BRM Developer's Reference*.

Corrective Invoice Document Types in BRM

A corrective invoice document in BRM can be either of the following:

- **Replacement invoice:** Shows all the items and events from the original bill with the corrected amounts.
- **Invoice correction letter:** Lists only the corrections.

BRM can create detailed or summary versions of those documents. For more information, see *BRM Designing and Generating Invoices*.

By default, BRM stores a detail replacement invoice as the default type of corrective invoice document for corrective bills.

BRM stores the type of corrective invoice document in an **event** object when it generates the corrective bill. If you provide a different value for the type of corrective invoice document when you run the **pin_inv_accts** utility, this setting is ignored.

For more information, see the discussion on corrective invoicing in *BRM Designing and Generating Invoices*.

About Threshold Amounts for Corrective Billing

You can specify a minimum amount that the 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 hierarchy, BRM checks that threshold against the sum of A/R actions for all accounts which had charges contributing to that bill.

The **CorrectiveBillThreshold** business parameter in the `/config/business_params` object acts as your default threshold for generating corrective bills. The default value for **CorrectiveBillThreshold** is 0.

You specify the threshold amount for generating a corrective bill as follows:

- By setting the value for **CorrectiveBillThreshold** in the `/config/business_params` object. This acts as the default threshold for generating corrective bills. See ["Specifying the Threshold Amount for Corrective Bills"](#).
- By setting the value for the `-threshold_amount` parameter when you generate the corrective bill using the `pin_make_corrective_bills` utility. This value overrides the value set in the **CorrectiveBillThreshold** parameter. See ["Providing Threshold Amounts Based on Customers"](#).

Correction Reasons for Corrective Billing

BRM provides the following default reasons for correcting a bill:

- Update to the invoice address
- Manual adjustment
- Price correction

BRM stores these reasons in `reasons.en_US`, in the `BRM_home/sys/msg/reasoncodes` directory, where `BRM_Home` is the directory where you installed BRM components. It 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. See ["Providing Custom Reason Codes for Bill Corrections"](#) for more information.

For more information on string manipulations functions and reason codes, see *BRM Developer's Reference*.

About Payment Due Dates for Corrective Bills

BRM determines due dates for regular and corrective bills by using the `PCM_OP_BILL_POL_CALC_PYMT_DUE_T` policy opcode.

By default, the `PCM_OP_BILL_POL_CALC_PYMT_DUE_T` policy opcode recalculates the due dates for corrective bills starting with the current time when the opcode generates the corrective bill and taking into account the payment terms. For example, if the payment terms set the due date for a bill to be 30 days from the billing date, and the corrective bill was generated on April 12, the due date for the corrective bill would be May 12.

For information, see `PCM_OP_BILL_POL_CALC_PYMT_DUE_T` facility module opcode in *BRM Developer's Reference*.

Handling Payments for Bills with Corrections

Corrections to bill charges and bill 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.

Processing Payments for Previous Bills

By default, 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.

You can require payments to be specific to the bill and reject payments for prior bills. To do so, set the **RejectPaymentsForPreviousBill** business parameter in the **billing** instance of the **/config/business_params** object to **enabled**. BRM will reject the payments to the prior bills and send the rejected payments to Payment Suspense Manager. See "[Rejecting Payments for Prior Bills](#)" for more information.

Processing Payments Received before Creating a Corrective Bill

By default, if BRM receives a partial or full payment for a bill, it does not enable a corrective bill to be generated for that bill. Instead, BRM assigns all corrections 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. See "[Generating Corrective Bills for Partially or Fully Paid Bills](#)" for more information.

Processing Payments Received before Issuing a Corrective Invoice

When BRM processes payments for a bill *before* it issues the corrective invoice for the bill, the corrective bill includes all the payment details for payments already processed against that bill.

It is recommended that, when you generate a corrective bill, you should also generate the corrective invoice for that corrective bill. A/R actions that occur between the time a corrective bill and its corrective invoice are generated may lead to incorrect invoices.

Processing Payments after Issuing a Corrective Invoice

After BRM issues the corrective invoice for a corrective bill, it cancels the original invoice for the bill and processes subsequent payments against the corrective bills for that billed period. Payments become due on the new due dates associated with the corrective invoice for the corrected bill.

If a payment is allocated to specific bill items on the original invoice, BRM processes that payment against the latest corrected balance for the bill items. Any payment that is in excess is automatically credited to the balance on the account.

About Open Item Accounting and Rerating

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 corrective invoices for all the previous bills that were affected by the rerating. When you generate corrective invoices for such corrected bills, the rerated events are displayed in the corrective 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.

About Handling Updates to Discount Sharing and Charge Sharing

You must rerate accounts following updates to the discount sharing 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.

About 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.

Preventing Corrective Bill Generation to Handle 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.

For information on disputes, settlements, and write-offs, see *BRM Managing Accounts Receivable*.

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 dispute, the corrective bill contains the dispute details, the disputed bill item, and the updated bill balance.

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 for a settlement, the corrective bill contains the settlement details, the settled bill item, and the updated bill balance.

A dispute may be settled for the full amount of the dispute in the customer's favor. If you generate a corrective bill during such a dispute, the corrective bill generated at settlement time may become a duplicate because there may not be any 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, the corrective bill contains the write-off details, the written-off bill item, and the updated bill balance.

About Automatic Allocation from Rerating

By default, BRM creates unallocated items for any adjustment items that are created as a result of rerating. For information about the rerating process in BRM, see *BRM Setting Up Pricing and Rating*.

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. When the **AllocateReratingAdjustments** business parameter is enabled, adjustment details by original item by original bill are reported on the next bill for regular billing also.

To ensure that the corrective billing process includes or excludes these adjustments, check the setting for the **AllocateReratingAdjustments** business parameter *before* you run the rerating process.

If BRM has rerated a **/bill** object without allocating automatic adjustments (from the rerating) to the original bills, the corrective bill for that **/bill** object will not include the item adjustments generated by that rerating.

Note: For such a **/bill** object, if you set the **AllocateReratingAdjustments** business parameter to **enabled** and rerun the rerating process for the bill, BRM does not allocate the adjustments.

You must manually allocate the rerated items *before* you generate the corrective bill for such a **/bill** object.

If you enable BRM to automatically allocate the item adjustments generated by the rerating process to the original bill, BRM allocates adjustments to the bill items being corrected in the following manner:

- For open item accounting, the adjustment items are allocated to each bill that was corrected. Allocation is made to each of the original bills that included events or items that were corrected by these adjustments.
- For balance forward accounting, corrections are posted to the last bill only. Therefore, the rerating allocation is made to the final bill only. This final bill carries over the balances from all prior bill periods.

See ["Enabling Automatic Allocation of Adjustments from Rerating"](#) for more information.

When BRM Generates Corrective Bills

You can have multiple corrections for a single bill. BRM generates corrective bills differently based on whether the corrections occur in the same period or in multiple periods.

How BRM Handles Multiple Bill Corrections in the Same Period

When there are multiple corrections for a bill period, BRM corrects the most recent corrective bill for the period. 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.

[Table 2–2](#) shows multiple corrections for a bill. From the **Actions** menu on the **Bill Details** page in Customer Center, if you select:

- **Corrections Only** to view the corrective bill for Dec. 27, only the numbers for "Usage Y Adjustment" are displayed.
- **Show All** to view the corrective bill for Dec. 27, the numbers for "Usage X Adjustment" and "Usage Y Adjustment" are displayed.

Table 2–2 How BRM References Multiple Bills in One Period

Date	Event	Type of Bill Created	Bill Number Assigned to This Bill	Bill Number of the Bill Corrected
Nov. 30	Create Bill	Original	B-168	None

Table 2–2 (Cont.) How BRM References Multiple Bills in One Period

Date	Event	Type of Bill Created	Bill Number Assigned to This Bill	Bill Number of the Bill Corrected
Dec. 10	Adjust for Cash Forward event; Correct the bill	Corrective	CB1-7	B-168
Dec. 15	Usage X Adjustment	Corrective	CB1-13	CB1-7
Dec. 27	Usage Y Adjustment	Corrective	CB1-19	CB1-13

How BRM Handles Bill Corrections Spanning Multiple Periods

When corrections span multiple billed periods, BRM uses the accounting method specified in the account to determine the period for which to generate the corrective bill. For more information on BRM accounting types, see "[About Accounting Types](#)".

Balance Forward Accounting Type For accounts using the balance forward accounting method, BRM generates a single corrective bill for the last billed period.

Open Item Accounting Type For accounts using the open item accounting method, BRM creates corrective bills for each billed period 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 period to the next open bill, which in this case is the bill for December. The corrections for the November period are included in the November bill because that bill is still pending payment.

Setting Up the Corrective Billing Process in BRM

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 in BRM](#)".
- Restrict CSR permissions, as necessary. See "[Restricting CSR's Permission to Corrective Billing](#)".
- Set the type of corrective invoice for the corrective bill. See "[Specifying the Corrective Invoice Type for a Corrective Bill](#)".
- Create custom validation for corrective bills. See "[Setting Up Custom Validations for Corrective Bills](#)".
- Set up a numbering scheme for corrective bill numbers. See "[Setting Up Bill Numbers for Corrective Bills](#)".
- Customize due dates for the corrective bills. See "[Customizing Due Dates for Corrective Bills](#)".
- Create custom reasons for the correction. BRM provides default reasons, but you can add to this list. See "[Specifying Correction Reasons](#)".
- Specify the threshold amount for corrective bills. See "[Specifying the Threshold Amount for Corrective Bills](#)".
- Enable automatic adjustment from rerating. See "[Enabling Automatic Allocation of Adjustments from Rerating](#)".

- 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 ["Generating Corrective Bills for Partially or Fully Paid Bills"](#).

Enabling Corrective Billing in BRM

You can enable corrective billing by modifying the **billing** instance of the `/config/business_params` object.

To enable corrective billing:

1. Go to the `BRM_Home/sys/data/config` directory, where `BRM_Home` is the directory where you installed BRM components.
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 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. Search for the following line.

```
<EnableCorrectiveInvoices>disabled</EnableCorrectiveInvoices>
```

5. Change **disabled** to **enabled**.

Caution: BRM uses the XML in this file to overwrite the existing **billing** 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 subscription configurations.

6. Save this file as **bus_params_billing.xml**.
7. Use the following command to load this change into the appropriate `/config/business_params` object. (Execute this command from the `BRM_Home/sys/data/config` directory.)

```
pin_bus_params PathToWorkingDirectory/bus_params_billing.xml
```

Where *PathToWorkingDirectory* is the directory in which **bus_params_billing.xml** resides.

Note: To execute it from a different directory, see the description for **pin_bus_params** in *BRM Developer's Guide*.

8. Read the object with the **testnap** utility or Object Browser to verify that all fields are correct.

See "Using testnap" in *BRM Developer's Guide* for general instruction on using the **testnap** utility. See "Reading Objects by Using Object Browser" in *BRM Developer's Guide* for information on how to use Object Browser.

9. Stop and restart the Connection Manager (CM). For more information, see "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.
10. (Multischema systems only) Run the `pin_multidb` script with the `-R CONFIG` parameter. For more information, see "pin_multidb" in *BRM System Administrator's Guide*.

Restricting CSR's Permission to Corrective Billing

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.

Specifying the Corrective Invoice Type for a Corrective Bill

You can provide the corrective invoice type for a corrective bill during the billing process.

Specifying the Default Invoice Type at the Account Level

Set up a default type of corrective invoice in the `PIN_FLD_INV_TYPE` field of the `/payinfo` object before you start generating corrective bills.

Use the `PCM_OP_CUST_POL_PREP_PAYINFO` policy opcode to override the default value (0, which yields a detail replacement invoice). For more information, see `PCM_OP_CUST_POL_PREP_PAYINFO` in *BRM Developer's Reference*. For information on how BRM arrives at the type of corrective invoice, see "pin_inv_accts" in *BRM Designing and Generating Invoices*.

Specifying the Invoice Type When Creating the Corrective Bill

You associate a corrective invoice type with a corrective bill when you generate the corrective bill in one of the following ways:

- In Customer Center, select **Replacement Invoice** or **Invoice Correction Letter** from the menu for **Invoice Correction Type** and **Summary** or **Detail** from the menu for **Invoice Correction Format**. Then, select **Produce Customer Bill** to generate the corrective bill. See the description for the **Balance** tab in BRM Customer Center Help.
- Run the `pin_make_corrective_bill` utility with the `-corrective_inv_type` parameter set to one entry from each of the following:
 - **R** or **L**. **R** indicates **Replacement Invoice** and **L** indicates **Invoice Correction Letter** as the selected type of corrective invoice.
 - **S** or **D**. **S** indicates **Summary** and **D** indicates **Detail** version of the selected type of corrective invoice.

Enter the two values separated by a blank. For example, to generate a detail replacement invoice,

```
pin_make_corrective_bills -corrective_inv_type R D
```

If you omit one of the entries or omit the blank between the two entries, the `pin_make_corrective_bill` utility will fail to generate a corrective bill for that bill.

See "[pin_make_corrective_bills](#)" for more information.

Setting Up Custom Validations for Corrective Bills

To set up 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 PCM_OP_BILL_POL_VALID_CORRECTIVE_BILL in *BRM Developer's Reference*.

Setting Up Bill Numbers for Corrective Bills

The default numbering format for both regular and collective bills has been described under "[About Bill Numbering in BRM](#)".

Change the default bill numbering for corrective bills for the following reasons:

- To retain the default prefix and use the sequence numbers for *regular* bills. See "[Retaining the Numbering Format Used for Regular Bills](#)".
- To set up a custom numbering format for corrective bills. See "[Customizing the Bill Numbering Format](#)".

Retaining the Numbering Format Used for Regular Bills

To retain the numbering format you used for regular bills:

1. 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 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.

2. Open the **bus_params_billing.xml.out** file.
3. Search for the following line.

```
<GenerateCorrectiveBillNo>enabled<GenerateCorrectiveBillNo>
```

4. Change **enabled** to **disabled**.

Caution: BRM uses the XML in this file to overwrite the existing **billing** 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 subscription configurations.

5. Save this file as **bus_params_billing.xml**.
6. Go to the *BRM_Home/sys/data/config* directory which includes support files used by the **pin_bus_params** utility.
7. Use the following command to load this change into the appropriate **/config/business_params** object.

```
pin_bus_paramsPathToWorkingDirectory/bus_params_billing.xml
```

where *PathToWorkingDirectory* is the directory in which **bus_params_billing.xml** resides.

Note: To execute it from a different directory, see the description for **pin_bus_params** in *BRM Developer's Guide*.

8. Read the object with the **testnap** utility or Object Browser to verify that all fields are correct.

See "Using testnap" in *BRM Developer's Guide* for general instruction on using the **testnap** utility. See "Reading Objects by Using Object Browser" in *BRM Developer's Guide* for information on how to use Object Browser.
9. Stop and restart the Connection Manager (CM). For more information, see "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.
10. (Multischema systems only) Run the **pin_multidb** script with the **-R CONFIG** parameter. For more information, see "pin_multidb" in *BRM System Administrator's Guide*.

Customizing the Bill Numbering Format

You can customize the bill number prefix and change the starting sequence number for the bill objects. See "[Customizing Bill and Invoice Numbers](#)" for more information.

Customizing Due Dates for Corrective Bills

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. See "[Customizing Bill Due Date Calculations for Payment Terms](#)" for more information.

Specifying Correction Reasons

You can specify correction reasons:

- When you create a corrective bill from Customer Center. You can select a reason for the correction.
- When you run the **pin_make_corrective_bills** utility. You provide the correction reason.

Providing Custom Reason Codes for Bill Corrections

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.

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 Threshold Amount for Corrective Bills

You specify the minimum threshold amount for corrective bills by setting the `CorrectiveBillThreshold` business parameter in the `/config/business_params` object. The default value is 0.

To set the `CorrectiveBillThreshold` business parameter:

1. 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 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.

2. Open the `bus_params_billing.xml.out` file.
3. Search for the following line:

```
<CorrectiveBillThreshold>0</CorrectiveBillThreshold>
```

4. Change 0 to the required value. For example,

```
<CorrectiveBillThreshold>15</CorrectiveBillThreshold>
```

Caution: BRM uses the XML in this file to overwrite the existing `billing` 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 subscription configurations.

5. Save this file as `bus_params_billing.xml`.
6. Go to the `BRM_Home/sys/data/config` directory which includes support files used by the `pin_bus_params` utility.
7. Use the following command to load this change into the appropriate `/config/business_params` object.

```
pin_bus_paramsPathToWorkingDirectory/bus_params_billing.xml
```

where `PathToWorkingDirectory` is the directory in which `bus_params_billing.xml` resides.

Note: To execute it from a different directory, see the description for `pin_bus_params` in *BRM Developer's Guide*.

8. Read the object with the `testnap` utility or Object Browser to verify that all fields are correct.

See "Using testnap" in *BRM Developer's Guide* for general instruction on using the `testnap` utility. See "Reading Objects by Using Object Browser" in *BRM Developer's Guide* for information on how to use Object Browser.

9. Stop and restart the CM. See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.
10. (Multischema systems only) Run the `pin_multidb` script with the `-R CONFIG` parameter. For more information, see "pin_multidb" in *BRM System Administrator's Guide*.

Providing Threshold Amounts Based on Customers

You can provide specific threshold amount based on a customer or set of customers when you run the `pin_make_corrective_bills` utility.

To use different thresholds for a customer or customers in a specific segment:

1. Run the `pin_make_corrective_bills` utility separately for the customers in that specific segment.
2. Enter the appropriate threshold as the value for the `-threshold_amount` input parameter to the `pin_make_corrective_bills` utility.

See "[Billing Utilities](#)" for more information on the `pin_make_corrective_bills` utility.

Enabling Automatic Allocation of Adjustments from Rerating

You enable automatic allocation of rerating adjustments by setting the `AllocateReratingAdjustments` business parameter in the `/config/business_params` object by using the `pin_bus_params` utility.

To set the `AllocateReratingAdjustments` business parameter:

1. Use the following command to create an editable XML file from the `rerate` instance of the `/config/business_params` object:

```
pin_bus_params -r BusParamsRerate bus_params_rerate.xml
```

This command creates the XML file named `bus_params_rerate.xml.out` in your working directory. To place this file in a different directory, specify the path as part of the file name.

2. Open the `bus_params_rerate.xml.out` file.
3. Search for the following line.

```
<AllocateReratingAdjustments>disabled</AllocateReratingAdjustments>
```

4. Change `disabled` to `enabled`.

Caution: BRM uses the XML in this file to overwrite the existing `rerate` 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 subscription configurations.

5. Save this file as `bus_params_rerate.xml`.
6. Go to the `BRM_Home/sys/data/config` directory, which includes support files used by the `pin_bus_params` utility.
7. Use the following command to load this change into the appropriate `/config/business_params` object.

```
pin_bus_paramsPathToWorkingDirectory/bus_params_rerate.xml
```

where *PathToWorkingDirectory* is the directory in which **bus_params_rerate.xml** resides.

Note: To execute it from a different directory, see the description for **pin_bus_params** in *BRM Developer's Guide*.

8. Read the object with the **testnap** utility or Object Browser to verify that all fields are correct.
See "Using testnap" in *BRM Developer's Guide* for general instruction on using the **testnap** utility. See "Reading Objects by Using Object Browser" in *BRM Developer's Guide* for information on how to use Object Browser.
9. Stop and restart the CM. For more information, see "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.
10. (Multischema systems only) Run the **pin_multidb** script with the **-R CONFIG** parameter. For more information, see "pin_multidb" in *BRM System Administrator's Guide*.

Rejecting Payments for Prior Bills

You can reject payments for prior bills (and thereby require them to be specific to the bill) by setting the **RejectPaymentsForPreviousBill** business parameter in the **/config/business_params** object. When you do so, BRM sends the rejected payments to Payment Suspense Manager.

To reject payments for prior bills and require them to be specific to the bill:

1. 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 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.

2. Open the **bus_params_billing.xml.out** file.
3. Search for the following line:

```
<RejectPaymentsForPreviousBill>disabled</RejectPaymentsForPreviousBill>
```

4. Change **disabled** to **enabled**.

Caution: BRM uses the XML in this file to overwrite the existing **billing** 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 subscription configurations.

5. Save this file as **bus_params_billing.xml**.
6. Go to the **BRM_Home/sys/data/config** directory, which includes support files used by the **pin_bus_params** utility.
7. Use the following command to load this change into the appropriate **/config/business_params** object.

```
pin_bus_paramsPathToWorkingDirectory/bus_params_billing.xml
```

where *PathToWorkingDirectory* is the directory in which **bus_params_billing.xml** resides.

Note: To execute it from a different directory, see the description for **pin_bus_params** in *BRM Developer's Guide*.

8. Read the object with the **testnap** utility or Object Browser to verify that all fields are correct.

See "Using testnap" in *BRM Developer's Guide* for general instruction on using the **testnap** utility. See "Reading Objects by Using Object Browser" in *BRM Developer's Guide* for information on how to use Object Browser.
9. Stop and restart the CM. For more information, see "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.
10. (Multischema systems only) Run the **pin_multidb** script with the **-R CONFIG** parameter. For more information, see "pin_multidb" in *BRM System Administrator's Guide*.

Generating Corrective Bills for Partially or Fully Paid Bills

You can generate corrective bills for fully paid or partially paid bills by setting the **AllowCorrectivePaidBills** business parameter in the **/config/business_params** object by using the **pin_bus_params** utility.

To generate corrective bills for fully paid or partially paid bills:

1. 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 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.

2. Open the **bus_params_billing.xml.out** file.
3. Search for the following line:

```
<AllowCorrectivePaidBills>disabled</AllowCorrectivePaidBills>
```

4. Change **disabled** to **enabled**.

Caution: BRM uses the XML in this file to overwrite the existing **billing** 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 subscription configurations.

5. Save this file as **bus_params_billing.xml**.
6. Go to the **BRM_Home/sys/data/config** directory, which includes support files used by the **pin_bus_params** utility.
7. Use the following command to load this change into the appropriate **/config/business_params** object.

```
pin_bus_paramsPathToWorkingDirectory/bus_params_billing.xml
```

where *PathToWorkingDirectory* is the directory in which **bus_params_billing.xml** resides.

Note: To execute it from a different directory, see the description for **pin_bus_params** in *BRM Developer's Guide*.

8. Read the object with the **testnap** utility or Object Browser to verify that all fields are correct.

See "Using testnap" in *BRM Developer's Guide* for general instruction on using the **testnap** utility. See "Reading Objects by Using Object Browser" in *BRM Developer's Guide* for information on how to use Object Browser.

9. Stop and restart the CM. For more information, see "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.
10. (Multischema systems only) Run the **pin_multidb** script with the **-R CONFIG** parameter. For more information, see "pin_multidb" in *BRM System Administrator's Guide*.

Generating Corrective Bills

After you have configured corrective billing, you are ready to generate corrective bills.

You can generate corrective bills in one of the following ways:

- Allocate adjustments to a selected bill in the Customer Center and submit the bill for corrective billing. For more information, see Customer Center Help.
- Run the **pin_make_corrective_bills** utility which generates the corrective bill.

Whether you generate corrective bills using Customer Center or the **pin_make_corrective_bills** utility, BRM runs the **PCM_OP_BILL_MAKE_CORRECTIVE_BILL** opcode to generate the required corrective bills. For more information on **PCM_OP_BILL_MAKE_CORRECTIVE_BILL**, see *BRM Developer's Reference*.

After you begin the process of generating a corrective bill, you cannot cancel the process.

Generating Corrective Bills Using Customer Center

To generate a corrective bill through Customer Center, you select the bill to be corrected, select the necessary adjustments to apply to the bill, select the type of corrective invoice you require, and submit the bill and its adjustments to BRM for corrective billing.

When BRM creates the corrective bill, it displays the bill number for the corrective bill. The **Balances** tab displays the contents of the newly-generated corrective bill. If there was an error, BRM provides the details for the error in the log file for the utility.

For more information, see Customer Center Help.

Generating Corrective Bills for Bills without Charge Corrections

Charge corrections are corrections to charges on a bill that result from adjustments, dispute, settlements, payments, writeoff, or writeoff reversal, and so on.

By default, the **pin_make corrective_bills** utility generates corrective bills for bills with charge corrections only. To run the **pin_make corrective_bills** utility and generate corrective bills when there are only simple changes such as invoice address changes, call this utility with **-create_if_no_corrections** parameter set to **Y** and **-corrective_inv_type** parameter set to **R**.

Note: When the opcode called by **pin_make corrective_bills** creates the **/event/billing/corrective_bill** object for a bill associated with no charge corrections, the opcode sets the corrective invoice type in that object to be replacement invoice only.

For a detailed replacement invoice, run **pin_make corrective_bills** with the following entries:

```
pin_make_corrective_bills -create_if_no_corrections Y -corrective_inv_type R D
```

For a summary replacement invoice, run **pin_make corrective_bills** with the following entries:

```
pin_make_corrective_bills -create_if_no_corrections Y -corrective_inv_type R S
```

See the description for **pin_make corrective_bill** in "[pin_make_corrective_bills](#)".

How PCM_OP_BILL_MAKE_CORRECTIVE BILL Works

When you generate corrective bills by using Customer Center or the **pin_make corrective_bills** utility, BRM calls the **PCM_OP_BILL_MAKE_CORRECTIVE_BILL** opcode.

The **PCM_OP_BILL_MAKE_CORRECTIVE_BILL** opcode does the following:

1. Locks the bill's **/billinfo** object. After BRM locks a **/billinfo** object, it does not accept any corrections until the **/billinfo** object is unlocked.

Caution: If User A and User B both access a bill and User A submits the bill for corrective billing, BRM locks the **/billinfo** object. BRM applies any adjustments made by User B to the corrected bill object only *after* BRM unlocks the **/billinfo** object.

Also, if User B had applied the adjustment *before* User A submitted the bill for correction, the corrected bill will contain *both* adjustments.

2. Verifies that a corrective bill can be generated for the corrections by completing the following:
 - Standard validations for the request. See "[Standard Validations Performed by BRM](#)".
 - Policy validations. See "[Policy Validations Performed by BRM](#)". The opcode also performs any custom validations that you have added and that are necessary. See "[Setting Up Custom Validations for Corrective Bills](#)".

If Customer Center calls this opcode and the bill fails to validate, the opcode returns the information to Customer Center, which then displays an appropriate message requiring you to take necessary actions.

If the opcode validates that a corrective bill can be generated and such a bill is to be generated, the opcode continues its process. It creates the `/history_bills` object for the `/billinfo` object. See "[About Maintaining a Bill's History](#)".

3. Checks to see whether the `pin_make_corrective_bills` utility called it with the `-validate_only` parameter. If so, this opcode reports whether BRM can generate a corrective bill for the selected bill and does not proceed further.
4. Assigns a new bill number for the corrective bill. See "[Setting Up Bill Numbers for Corrective Bills](#)".
5. Sets the due date for the corrective bill. See "[Customizing Due Dates for Corrective Bills](#)".
6. Creates the `/event/billing/corrective_bill` object for the prior bill in the following way:
 - The opcode includes all the A/R actions applied or allocated to the prior bill. These actions impact the totals or balances for the respective bill items and the bill. For the bill items that do not have any A/R action, BRM stores the value from the original bill in the current bill.

The following entries for the `PIN_FLD_FLAGS` field in a bill object indicate that there was an allocation of settlement or refund for the bill. The `BRM_Home/include/pin_flds.h` file contains the values that BRM uses in the `PIN_FLD_FLAGS` field for a corrective bill.

```
#define PIN_BILL_FLG_SETTLEMENT_ALLOC 0x200
#define PIN_BILL_FLG_REFUND_ALLOC 0x400
```

- The opcode stores a correction reason you provide. See "[Specifying Correction Reasons](#)".
- The opcode stores the name for the corrective bill. The `BRM_Home/include/pin_bill.h` file contains the values that BRM uses in the `PIN_FLD_NAME` field for a corrective bill:

```
#define PIN_OBJ_NAME_CORRECTIVE_BILL "PIN Corrective Bill"
#define PIN_OBJ_NAME_CORRECTIVE_BILL_NOW "PIN Corrective Bill Now"
#define PIN_OBJ_NAME_CORRECTIVE_BILL_ON_DEMAND "PIN Corrective Bill On Demand"
```

If the opcode encounters an error in generating the corrective bill, it logs an error against the prior bill in the `default.pinlog` utility log file and does not generate the corrective bill.

When this opcode is run by Customer Center, BRM displays a message to indicate the success or error of the process. Review the `default.pinlog` file for errors and take necessary actions. This file is located either in the directory from which you started the utility or in a directory specified in the configuration file.

Generating Corrective Bills with the `pin_make_corrective_bills` Utility

The `pin_make_corrective_bills` utility generates corrective bills for specific bill numbers or account numbers.

To generate a corrective bill using `pin_make_corrective_bills`:

1. If you plan to input the bill numbers or account numbers using a billing run configuration file, set up the necessary file. The default file is `pin_bill_run_control.xml` in the `BRM_home/apps/pin_bill` directory. See "[About Bill Run Management](#)" for a sample billing run configuration file.

2. Go to the *BRM_Home/apps/pin_bill*d (or appropriate) directory where this file resides.
3. Verify that the selected accounts or bills are eligible for corrective billing by running the following command:

```
cd BRM_Home/apps/pin_bill  
pin_make_corrective_bills -validate_only
```

The utility writes the information for each bill to the log file for the utility.

4. Check the log file for error messages and act as necessary.
5. Run the **pin_make_corrective_bills** utility, providing all the necessary inputs. Omit the **-validate_only** parameter.

For more information on the **pin_make_corrective_bills** utility, see "[pin_make_corrective_bills](#)".
6. Check the **default.pinlog** log file for error messages and act as necessary.
7. Verify the contents of the newly-generated corrective bills using Customer Center.

Post-Processing Actions for Corrective Bills

After you generate corrective bills, you can do the following:

- View corrective bills. See "[Viewing Corrective Bills](#)".
- 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*.
- Manage **/history_bills** objects in the BRM database by running the **pin_purge** utility. For more information, see "pin_purge" in *BRM System Administrator's Guide*.

Viewing Corrective Bills

You can view all bills through Customer Center.

- Original Bill: Customer Center displays the original bill with the adjustments and item amounts that were included with the original bill
- Corrective Bill: Customer Center, BRM displays the A/R actions and corresponding corrected item as the details for the corrective bill. Use the **Actions** menu to select the level of detail:
 - **Corrections Only**. BRM displays unbilled A/R Actions and Items with A/R Actions (Unbilled) amount not equal to 0.

- **Show All.** This is the default. BRM displays both unbilled and billed A/R Actions.

If you use an external application such as Seibel CRM, you can view only the most recent bill for a bill period, that is, the last corrective bill if it was created.

Part II

Configuring Billing

Part II describes how to configure Oracle Billing and Revenue Management (BRM) billing. It contains the following chapters:

- [Setting Business Policies for Billing](#)
- [About Proration](#)
- [Managing Bill Units with Your Custom Application](#)
- [Offering the Best Price to Your Customers](#)
- [Setting Up Pipeline-Triggered Billing](#)
- [About Bill Cycle Management](#)
- [About Bill Run Management](#)
- [About Bill Suppression](#)
- [Creating Custom Bill Items](#)
- [Remitting Funds to Third Parties](#)

Setting Business Policies for Billing

This chapter describes how to configure Oracle Communications Billing and Revenue Management (BRM) billing and set up system-wide billing defaults (for example, the default billing-cycle length).

Before configuring billing, read "[About Billing](#)".

This chapter does not cover information on increasing billing performance, such as running multiple billing processes. For information on increasing billing performance, see "Tuning Billing Performance" in *BRM System Administrator's Guide*.

Note: Many billing defaults are set by editing configuration files.

For information on setting defaults for invoicing, see "Setting Invoicing Defaults" in *BRM Designing and Generating Invoices*.

Setting Default Billing Properties for Account Creation

You can change the defaults for the following billing properties for new accounts:

- [Setting the Default Accounting Day of Month \(DOM\)](#)
- [Setting the Default Billing-Cycle Length](#)
- [Setting the Default Accounting Type](#)
- [Setting the Billing DOM According to the Payment Method](#)
- [Setting the First Billing Cycle to the Day after Account Creation](#)

For information about setting default invoicing properties, see "Setting Invoicing Defaults" in *BRM Designing and Generating Invoices*.

Setting the Default Accounting Day of Month (DOM)

Tip: 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.

For information about the accounting DOM, see "[About Accounting Cycle Dates](#)".

To set the default accounting DOM:

1. Open the Connection Manager (CM) configuration file (*BRM_Home/sys/cm/pin.conf*) in a text editor. *BRM_Home* is the directory where you installed BRM components.

2. Uncomment the following line and enter a value from 1 to 28.

```
- fm_cust_pol actg_dom 28
```

Note: To use the day that the account was created as the default, comment out the line by using the pound (#) symbol.

3. Save and close the file.

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

Setting the Default Billing-Cycle Length

For information about billing-cycle length, see "[About Billing Cycles](#)".

Note: If you create a consumer account through Customer Center, the value of the PIN_FLD_BILL_WHEN field is always set to 1. If you create a business account through Customer Center, you can configure the value of the PIN_FLD_BILL_WHEN field. If the input list of the PCM_OP_CUST_COMMIT_CUSTOMER opcode does not have any value defined for PIN_FLD_BILL_WHEN field, the value specified in the CM *pin.conf* file's - *fm_cust_pol bill_when* entry is considered.

To set the default billing-cycle length:

1. Open the CM configuration file (*BRM_Home/sys/cm/pin.conf*) in a text editor.
2. Add the following line and enter the number of months in one billing cycle.

Note: The default is 1, which is monthly billing.

```
- fm_cust_pol bill_when 2
```

3. Save and close the file.

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

Setting the Default Accounting Type

You set the default accounting type for all bill units by using the CM *pin.conf* file.

For information about accounting types, see "[About Accounting Types](#)".

To set the default accounting type:

1. Open the CM configuration file (*BRM_Home/sys/cm/pin.conf*) in a text editor.
2. Add the following line:

```
- fm_cust_pol actg_type n
```

- To set the default accounting type to open item accounting, enter **1**.
 - To set the default accounting type to balance forward accounting, enter **2**. This is the default.
3. Save and close the file.

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

Setting the Billing DOM According to the Payment Method

You can set the billing DOM for new customers according to the payment method. For example, you can set up all accounts that pay for bills using the invoice payment method to be billed for those bills on the same day. To do this, customize the PCM_OP_CUST_POL_PREP_BILLINFO policy opcode. Also, use event notification to implement your customization when existing customers change payment methods.

For more information, see "[Preparing /billinfo Data](#)".

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

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 (`/billinfo` object), 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.

Important: When billing fails for a subordinate account, the status of the subordinate **/billinfo** object and of the parent account's **/billinfo** object are both set to **PIN_BILL_ERROR**. This status ensures that when billing fails for a subordinate account, the parent account is also not billed. Otherwise, the parent bill may not include charges from the subordinate account, 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:

```
- pin_bill_accts unset_error_status 1
```

 - **0** sets the billing status in the **/billinfo** object. This is the default.
 - **1** does not set the billing status in the **/billinfo** object.
3. Save and close the file.
4. Run the billing utility.

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.

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

By default, the minimum amount is 2. When the amount due is less than 2, charges accrue in the account balances associated with the bill unit until they reach the minimum amount, and then the amount due is collected.

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 and close the file.

Setting the Minimum Amount to Charge

You set the default minimum amount to charge a customer in the CM **pin.conf** file **minimum_payment** entry. To check a batch of charges and refunds for any amounts below the minimum before charging and refunding customers, use the **PCM_OP_PYMT_POL_PRE_COLLECT** policy opcode. The **PIN_FLD_SESSION_OBJ** field in the input flist references the type of session in which the event occurred: either **/event/billing/batch/refund** or **/event/billing/batch/payment**, depending on the batch type.

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

Before performing the charges and refunds, the PCM_OP_PYMT_COLLECT opcode allocates the PIN_FLD_CHARGE array elements to open items and then calls the PCM_OP_PYMT_POL_PRE_COLLECT policy opcode.

The PCM_OP_PYMT_POL_PRE_COLLECT policy opcode then checks each element of the input PIN_FLD_CHARGES array to ensure:

- The result of selecting open items for allocating charges is set to PIN_SELECT_RESULT_PASS.
- The amount charged is greater than or equal to the minimum payment amount.
- The amount refunded is greater than or equal to the minimum and the account has a negative balance.
- The value of the input PIN_FLD_COMMAND field is valid.

By default, the results can be the following:

- If the amount charged is less than the minimum amount, the PCM_OP_PYMT_POL_PRE_COLLECT policy opcode sets the PIN_FLD_DESCR field to "Below minimum" and the result to PIN_CHARGE_RES_FAIL_NO_MIN.
- If the amount refunded is less than the minimum amount, the PCM_OP_PYMT_POL_PRE_COLLECT policy opcode sets the PIN_FLD_DESCR field to "Below minimum" and the result to PIN_CHARGE_RES_FAIL_NO_MIN.
- If PIN_FLD_COMMAND is set to PIN_CHARGE_CMD_REFUND and the account balance is zero or higher, the PCM_OP_PYMT_POL_PRE_COLLECT policy opcode sets the PIN_FLD_DESCR field to "No credit available" and the result to PIN_CHARGE_RES_NO_CREDIT_BALANCE.

You can change the minimum credit card charge amount by modifying the default minimum payment amount in the PCM_OP_PYMT_POL_PRE_COLLECT policy opcode.

You can also set minimums in configuration files:

- For information on setting the minimum charge amount, see ["Specifying the Minimum Payment to Collect"](#).
- For information on setting the minimum refund amount, see ["Specifying the Minimum Amount to Refund"](#).

You can also customize the PCM_OP_PYMT_POL_PRE_COLLECT policy opcode to retrieve soft descriptor information that enables you to display the name under which you do business (your DBA name), product name, and customer service number on your customer's checking account or credit card statement. See the discussion on customizing the policy source file for soft descriptors in *BRM Designing and Generating Invoices*.

Setting the Minimum Amount for Invoices

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

Setting the Minimum Amount for Finalizing Bills

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. For more information, see ["About](#)

[Automatic Bill Suppression"](#).

Specifying the Minimum Amount to Refund

You can specify the minimum amount to give as a refund. The **pin_refund** billing utility processes refund items with an amount greater than the minimum you specify.

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

By default, the minimum amount is 2. If the minimum amount is not reached, you can use Customer Center to transfer the amount to another item. For information, see the discussion *managing A/R* in the Customer Center Help.

1. Open the CM configuration file (*BRM_Home/sys/cm/pin.conf*) in a text editor.
2. Change the value of the **minimum_refund** entry. For example, to process refund items only for amounts greater than 3:

```
- fm_pynt_pol minimum_refund 3
```

3. Save and close the file.

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

Defining Non-refundable Items

By default, only refund items are nonrefundable.

You can make other credit items nonrefundable by modifying the **ar** parameter instance in the **/config/business_params** object. The **PIN_FLD_PARAM_VALUE** field contains a comma-delimited list of items to be excluded.

Important: Do not remove **/item/refund** from the **PIN_FLD_PARAM_VALUE** string.

You modify the **/config/business_params** object by using the **pin_bus_params** utility. See "pin_bus_params" in *BRM System Administrator's Guide*.

To set an item as nonrefundable:

1. Use the following command to create an editable XML file from the **ar** parameter instance in the **/config/business_params** object:

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

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

2. Search the XML file for following line:

```
<NonrefundableCreditItems>/item/refund</NonrefundableCreditItems>
```

3. Add the nonrefundable **/item** objects after the **/item/refund** entry, separated by commas.

Important: Do not remove **/item/refund** from the **PIN_FLD_PARAM_VALUE** string.

Caution: BRM uses the XML in this file to overwrite the existing **ar** 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 BRM's billing configuration.

4. Use the following command to load the change into the **/config/business_params** object:

```
pin_bus_params bus_params_AR.xml
```

You should execute this command from the *BRM_Home/sys/data/config* directory, which includes support files used by the utility. To execute it from a different directory, see "pin_bus_params" in *BRM System Administrator's Guide*.

5. Read the object with the **testnap** utility or Object Browser to verify that all fields are correct.

For general instructions on using **testnap**, see "Using testnap" in *BRM Developer's Guide*. For information on how to use Object Browser, see "Reading Objects by Using Object Browser" in *BRM Developer's Guide*.

6. Stop and restart the CM. For more information, see "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.
7. (Multischema systems only) Run the **pin_multidb** script with the **-R CONFIG** parameter. For more information, see "pin_multidb" in *BRM System Administrator's Guide*.

Managing Cycles

You can customize how to handle billing cycles and cycle fees. See the following topics:

- [About Time-Stamp Rounding](#)
- [Specifying How to Handle Partial Accounting Cycles](#)
- [Aligning Account and Cycle Start and End Times](#)
- [Specifying Which Billing Cycle to Assign to Deferred Purchase Fees](#)
- [About Billing Cycle Forward Fees in Advance](#)
- [About Applying Cycle Forward Fees in Parallel](#)
- [About Flexible Cycles](#)
- [Calculating Product Cycle Fees for Backdating](#)
- [About Bill Cycle Management](#)

About Time-Stamp Rounding

By default, BRM rounds time stamps to midnight. You can configure BRM to use exact time stamps by changing the **timestamp_rounding** entry in the CM configuration file (*BRM_Home/sys/cm/pin.conf*) from 1 (enabled) to 0 (disabled). You might need to do that to support a custom application.

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

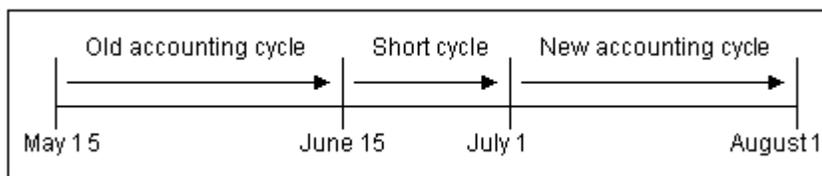
- **Billing cutoff time.** To use a billing cutoff time other than the default, time-stamp rounding must be enabled. Changing the cutoff time also changes the time to which time stamps are rounded throughout BRM. See "[Configuring the Billing Cutoff Time](#)" and "[How TimeStamp Fields Are Affected by 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 CM time-stamp rounding entry to specify whether this time stamp is rounded. See "Configuring Time-Stamp Rounding for Cycle Grants" in *BRM Setting Up Pricing and Rating*.
- **Validity period start time of resources granted by purchase events.** Use the CM time-stamp rounding entry and a business parameter to specify whether this time stamp is rounded. See "Configuring Time-Stamp Rounding for Purchase Grants" in *BRM Setting Up Pricing and Rating*.

Specifying How to Handle Partial Accounting Cycles

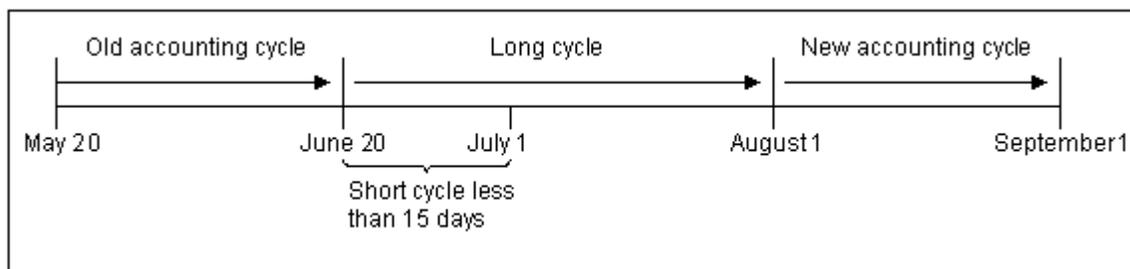
When you change the accounting cycle date in the middle of an accounting cycle, the new date does not take effect until after the current accounting cycle is over. This results in a gap of time between the end of the old accounting cycle and the start of the new accounting cycle.

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 3-1](#).

Figure 3-1 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 3-2](#).

Figure 3–2 Long Accounting Cycle

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

Short and Long Cycles with New Accounts

A short or long cycle can also occur when a customer registers and the billing DOM is different from the day of month when they register. For example, your company might require that all customers be billed on the first day of the month. If a customer registers 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.

How BRM Calculates Long Billing Cycles

By default, BRM uses the following formula to calculate long billing cycles:

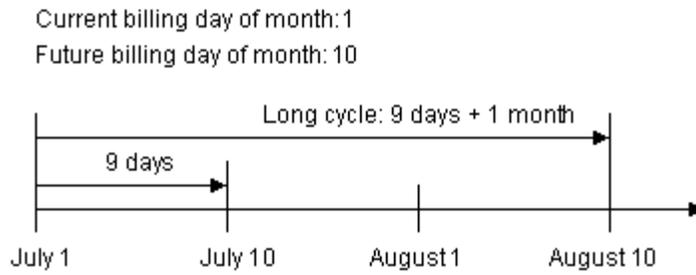
Use a short cycle unless one of the following is true:

- Future billing day of month > current billing day of month
AND
(Future billing day of month - current billing day of month) < 15
- Future billing day of month < current billing day of month
AND
(Current billing day of month - future billing day of month) > 15

Examples:

- The following example is shown in [Figure 3–3](#). If the current billing DOM is 1 and the future billing DOM is 10:
 $10 > 1$
 $10 - 1 = 9$
 Use a long cycle.

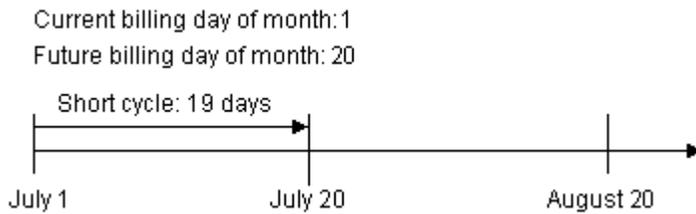
Figure 3–3 Long Cycle Example 1



- The following example is shown in [Figure 3–4](#). If the current billing DOM is 1 and the future billing DOM is 20:

$20 > 1$
 $20 - 1 = 19$
 Use a short cycle.

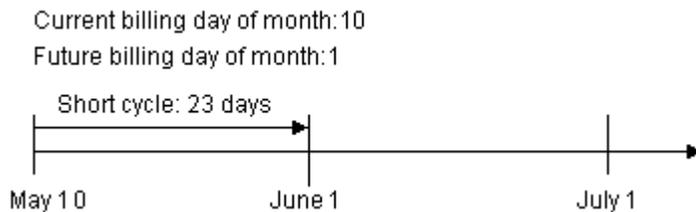
Figure 3–4 Short Cycle Example 1



- The following example is shown in [Figure 3–5](#). If the current billing DOM is 10 and the future billing DOM is 1:

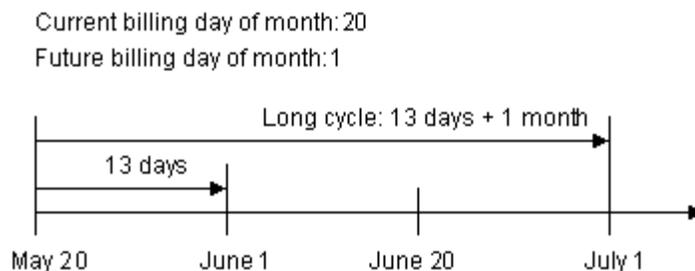
$1 < 10$
 $10 - 1 = 9$
 Use a short cycle.

Figure 3–5 Short Cycle Example 2



- The following example is shown in [Figure 3–6](#). If the current billing DOM is 20 and the future billing DOM is 1:

$1 < 20$
 $20 - 1 = 19$
 Use a long cycle.

Figure 3-6 Long Cycle Example 2

Rounding Up Long Billing Cycles

You can 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.

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 and close the file.
5. Stop and restart the CM. For more information, see "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

Changing How to Handle Partial Billing Cycles

To change how BRM handles short and long cycles, customize the `PCM_OP_CUST_POL_PREP_BILLINFO` policy opcode source code.

Aligning Account and Cycle Start and End Times

You can align the product 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 price list in Pricing Center or when you create an account in Customer Center.

For information, see "Managing Products" in *BRM Managing Customers*.
- The delayed start and end time is a whole number, not a fraction.
- The delay is measured in cycles.
- The product purchase, cycle, or usage start and end times are *not* modified when a deal is purchased.

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

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

Note: If the entry is set to 0 or not present, the start and end times are not aligned.

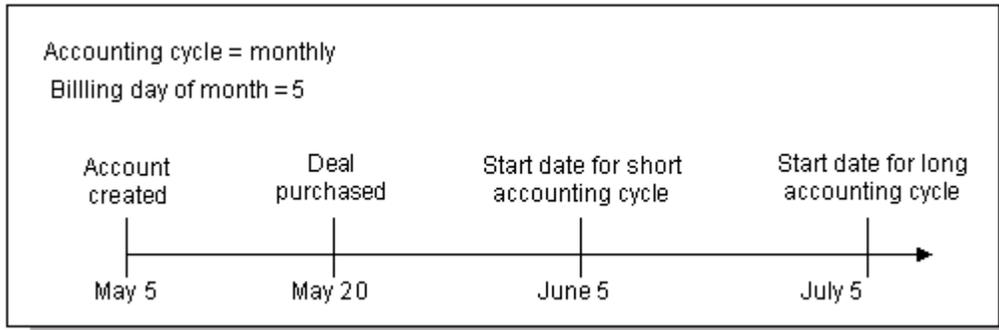
3. Save and close the file.
4. Stop and restart the CM. For more information, see "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

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 deal 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 3–7 shows the cycle start time for the above example:

Figure 3–7 Aligning Account and Cycle Start and End Times



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 current subordinate account(s) balance; 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 subordinate account(s) balance.

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 and close the file.
 4. Stop and restart the CM. See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

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 and close the file.
 4. Stop and restart the CM. For more information, see "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

About Billing Cycle Forward Fees in Advance

Usually, cycle forward fees for the following cycle are billed on the same day as the start of a new billing cycle. However, you can set up your price list to bill cycle forward fees in advance. See "Charging Cycle Forward Fees in Advance" in *BRM Setting Up Pricing and Rating*.

About Applying Cycle Forward Fees in Parallel

For accounts with multiple services (for example, a wholesale market customer account), you can configure BRM to apply cycle forward fees in parallel for multiple services instead of applying cycle forward fees sequentially for each service, thereby reducing the time to complete billing.

When you configure your BRM system for applying cycle forward fees in parallel, you can also configure BRM to:

- Enforce cycle fee processing prior to billing. By doing so, BRM eliminates the process of applying cycle forward fees during billing and improves the performance of the billing process.
- Use a single item at the account level to accumulate the cycle charges for all the services. By doing so, BRM reduces the number of items to process and improves overall system performance that is important when you are doing billing for wholesale customer accounts. When BRM applies cycle forward fees in parallel with service-level charges aggregated to a single account-level item, the account can have only a single bill unit (*/billinfo* object). Even though the account-level item aggregates the service charges, the respective service balance groups are still updated with the service charges.

Before configuring your BRM system for applying cycle forward fees in parallel, your system must meet the following requirements:

- The number of services attached to a single balance group must be less than 10 in order to get the performance benefit of applying cycle forward fees in parallel.
- There should be no dependency on the order of applying cycle forward fees for hierarchies (subordinate, charge-sharing, or discount-sharing). This is because the cycle forward fees are applied by the **pin_cycle_fees** utility instead of by the billing application that gives more control on the order of processing accounts in hierarchies.

Applying parallel cycle forward fees involves the following processes:

1. Running the **pin_cycle_fees** utility in parallel at the services level, which processes cycle forward fees aligned to the accounting cycle.

Note: When the parallel fee processing feature is enabled, cycle forward fees are applied by the **pin_cycle_fees** utility, which is run before the **pin_deferred_act** utility. However, when the parallel cycle forward fees processing fees feature is not enabled, cycle forward fees are applied by the billing application after running **pin_deferred_act**.

2. Running the **pin_update_items_journals** utility to post-process cycle forward fees.
3. Running the **pin_bill_accts** utility for regular billing.
4. Running the **pin_cycle_fees** utility to process flexible cycle forward fees.
5. Running the **pin_update_items_journals** utility to post-process flexible cycle forward fees.

Configuring BRM to Apply Cycle Forward Fees in Parallel

You use the **StagedBillingFeeProcessing** business parameter to specify how BRM applies cycle forward fees.

To configure BRM to apply cycle forward fees in parallel:

1. Go to the *BRM_Home/sys/data/config* directory.
2. Run the following command, which creates 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 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 in a text editor.
4. Search for the following line:

```
<StagedBillingFeeProcessing>0</StagedBillingFeeProcessing>
```

The default is **0**. (BRM applies the cycle forward fees as part of the billing process.)

Note: The cycle fee processing at the time of billing is only with reference to the cycle forward fees aligned with the accounting cycle boundary.

5. Do one of the following:
 - To apply cycle forward fees in parallel by service, specify 1.
 - To enforce cycle fee processing prior to billing and apply cycle forward fees in parallel by service, specify 2. See ["About Enforcing Cycle Forward Fee Processing Prior to Billing"](#) for more information.
 - To apply cycle forward fees in parallel by service with service-level charges aggregated to a single account-level item, specify 3. See ["About Aggregating Service Charges to Account Level Items"](#) for more information.
 - To enforce cycle fee processing prior to billing and apply cycle forward fees in parallel by service with service-level charges aggregated to a single account level item, specify 4. See ["About Enforcing Cycle Forward Fee Processing Prior to Billing"](#) and ["About Aggregating Service Charges to Account Level Items"](#) for more information.

Caution: BRM uses the XML in this file to overwrite the existing billing 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 billing and subscription configurations.

6. Save this file as `bus_params_billing.xml`.
7. Go to the `BRM_Home/sys/data/config` directory.
8. Load this change into the appropriate `/config/business_params` object by running the following command:

```
pin_bus_params PathToWorkingDirectory/bus_params_billing.xml
```

where *PathToWorkingDirectory* is the directory in which `bus_params_billing.xml` resides.

Note: To run this command from a different directory, see the description for "pin_bus_params" in *BRM Developer's Guide*.

9. Read the object with the `testnap` utility or Object Browser to verify that all fields are correct.

See the description of using `testnap` in *BRM Developer's Guide* for instructions on using the `testnap` utility. See the description of "Reading Objects by Using Object Browser" in *BRM Developer's Guide* for information on how to use Object Browser.
10. Stop and restart the Connection Manager (CM). For more information, see the description of "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

About Enforcing Cycle Forward Fee Processing Prior to Billing

When BRM enforces cycle fee processing prior to billing, the following processes are impacted:

- The `pin_cycle_fees` utility performs additional error processing to set the error status (as needed) on the corresponding `/billinfo` object.

Important: If you customize `pin_cycle_fees` and use the application global structure `PIN_FLD_EXTENDED_INFO` provided by the multithreaded application framework to hold custom information at run time, you must consider that `pin_cycle_fees` stores the error processing information in a single array element `PIN_FLD_ERROR_INFO` under `PIN_FLD_EXTENDED_INFO`.

- The billing process aborts if any of the following conditions is true:
 - The `BILLING_STATUS_FLAGS` field of the `/billinfo` object indicates that there was an error processing one of the cycle forward fees.
 - There is at least one service for which cycle fee processing (regular, deferred, deferred purchase, or deferred cancellation) has not been completed for the accounting cycle being billed.
- In rare cases, if billing is due for a bill unit for more than one accounting cycles, special handling is required. See "[Handling Skipped Billing](#)" for more information.

About Aggregating Service Charges to Account Level Items

When applying cycle forward fees in parallel by service with service-level charges aggregated to a single account-level item, multiple threads of `pin_cycle_fees` updates a single item. To avoid updating the same item by multiple threads, `pin_cycle_fees` logs the item and journal updates to temporary tables as follows:

- Logs item updates to the `/tmp_events_to_process` object in the `TMP_EVENTS_TO_PROCESS_T` table.
- Logs journal updates to the `/tmp_journals_to_process` object in the `TMP_JOURNALS_TO_PROCESS_T` table.

The `pin_update_items_journals` utility processes the temporary item and journal data and updates the main item and journal tables.

To ensure efficient access of these temporary tables, Oracle recommends the following:

- **Resetting high water mark.** Records are frequently inserted into and deleted from the temporary tables. This can result in fragmentation of the temporary tables. You must reset the high water mark for the temporary tables as the BRM schema user.

Run the following commands every time before calling the `pin_bill_accts` inside the `pin_bill_day` script.

```
ALTER TABLE TMP_JOURNALS_TO_PROCESS_T ENABLE ROW MOVEMENT;
ALTER TABLE TMP_JOURNALS_TO_PROCESS_T SHRINK SPACE;
ALTER TABLE tmp_events_to_process_t ENABLE ROW MOVEMENT;
ALTER TABLE tmp_events_to_process_t SHRINK SPACE;
```

For more information about the high water mark, see the Oracle Database documentation.

- **Presetting statistics.** Preset the statistics of the temporary tables that are created during BRM installation by running the following commands as a one-time activity. This enables BRM to avoid a full scan of these tables.

```
Exec dbms_stats.set_table_stats('SCHEMA_NAME', 'TMP_EVENTS_TO_PROCESS_T',
'', '', '', 200000000, 40000000, 1250) ;
```

```
Exec dbms_stats.set_index_stats('SCHEMA_NAME', 'I_TMP_EVENTS_ID',
'', '', numrows=>200000000, numlblks=>1000000, numdist=>200000000, avglblk=>1, avgdb
```

```
lk=>1,clstfct=>200000000);

Exec dbms_stats.set_column_stats('SCHEMA_NAME','TMP_EVENTS_TO_PROCESS_T','POID_ID0','','',distcnt=>200000000,density=>1/200000000,nullcnt=>0,srec=>srec_eve,avgclen=>11);

Exec dbms_stats.set_table_stats('SCHEMA_NAME','TMP_JOURNALS_TO_PROCESS_T','','',200000000,40000000,1250);

Exec dbms_stats.set_index_stats('SCHEMA_NAME','I_TMP_JOURNALS_ID','','',numrows=>200000000,numlblks=>1000000,numdist=>200000000,avglblk=>1,avgdblklk=>1,clstfct=>200000000);

Exec dbms_stats.set_column_stats('SCHEMA_NAME','TMP_JOURNALS_TO_PROCESS_T','POID_ID0','','',distcnt=>200000000,density=>1/200000000,nullcnt=>0,srec=>srec_eve,avgclen=>11);

Exec dbms_stats.lock_table_stats('SCHEMA_NAME','TMP_EVENTS_TO_PROCESS_T');
Exec dbms_stats.lock_table_stats('SCHEMA_NAME','TMP_JOURNALS_TO_PROCESS_T');
```

where *SCHEMA_NAME* is the BRM database user; for example, **pin1**.

Handling Skipped Billing

In rare cases, if billing is due for a bill unit for more than one accounting cycles, special handling is required. This multiple cycle overdue billing is referred to as skipped billing.

For example, consider that the current date is December 1 and BRM did not perform billing for the cycles ending November 1 and October 1. In this case, when you run **pin_bill_day** on the current date, three bills are due to be created.

When BRM tries to calculate the cycle forward fees, the following happens:

- **pin_cycle_fees** applies cycle forward fees due only as of October 1 because October 1 billing has not been processed yet.
- **pin_bill_accts** performs billing only on October 1 and aborts with an error when performing billing on November 1 because cycle forward fees due as of November 1 have not been processed yet.

To handle the case of skipped billing used in this example:

1. Run **pin_cycle_fees**, **pin_update_items_journals**, and **pin_deferred_act** in the following sequence:

```
pin_cycle_fees -defer_purchase
pin_cycle_fees -defer_cycle_fees
pin_cycle_fees -defer_cancel
pin_cycle_fees -regular_cycle_fees
pin_update_items_journals
pin_deferred_act
```

2. Run **pin_bill_accts**.
3. Repeat step 1 and step 2 two more times, which performs billing for November 1 and December 1.

Using the **pin_bill_day** Script to Apply Parallel Cycle Forward Fees

To support applying cycle forward fees in parallel, the **pin_bill_day** script includes the following commented out sections:

- **Pre-Billing Parallel Cycle Fee Processing:** Includes the following entries for `pin_cycle_fees` and `pin_update_items_journals`:

```
##### pin_cycle_fees -defer_purchase
##### pin_cycle_fees -defer_cycle_fees
##### pin_cycle_fees -defer_cancel
##### pin_cycle_fees -regular_cycle_fees
##### pin_update_items_journals
```

- **Post-Billing Parallel Cycle Fee Processing:** Includes the following entry for `pin_update_items_journals`:

```
##### pin_update_items_journals
```

To apply cycle forward fees in parallel by using the `pin_bill_day` script:

1. Make sure that the `StagedBillingFeeProcessing` parameter is not set to 0.
2. Open the `BRM_Home/bin/pin_bill_day` script in a text editor.
3. Uncomment the following entries in the **Pre-Billing Parallel Cycle Fee Processing** section:

```
##### pin_cycle_fees -defer_purchase
##### pin_cycle_fees -defer_cycle_fees
##### pin_cycle_fees -defer_cancel
##### pin_cycle_fees -regular_cycle_fees
##### pin_update_items_journals
```

4. Uncomment the following entry in the **Post-Billing Parallel Cycle Fee Processing** section:

```
##### pin_update_items_journals
```

5. Save and close the file.
6. Run `pin_bill_day`.

About Limitations and Impacts of Applying Cycle Forward Fees in Parallel

This section describes the limitations and impacts of configuring BRM to apply cycle forward fees in parallel:

- In rare cases, when the `pin_cycle_fees` utility successfully creates temporary item and journal data and the subsequent run of the `pin_update_items_journals` utility fails to update the item and journal tables, you must investigate and correct the problem in processing the temporary item and journal data before performing any accounts receivable action or generating ledger reports.
- If the parallel fee processing feature is configured to enforce cycle fee processing prior to billing, any balance impact event that occurs prior to running the `pin_cycle_fees` utility aborts with an error.
- If the parallel fee processing feature is *not* configured to enforce cycle fee processing prior to billing, any balance impact event that occurs prior to running the `pin_cycle_fees` utility results in triggered billing that can be slow due to serial application of cycle forward fees.
- There is no performance improvement to the following operations because parallel cycle fee processing does not apply to these operations:
 - Trial billing
 - Bill Now

- Billing on demand
- Account creation
- Purchase of deals
- Billing time discount
- Cycle fold
- Rollover
- Account activation, account inactivation, or account cancellation
- Best pricing
- Rerating

About Flexible Cycles

In BRM, cycle forward events are triggered to charge cycle forward fees. Typically, cycle forward fees are charged monthly at the beginning of the accounting cycle to charge for services provided during that cycle. By default, BRM supports monthly, bimonthly, quarterly, semi-annual, and annual cycle forward events. You can also configure BRM to support *flexible cycles*. Flexible cycles can be daily, weekly, monthly, or multimonth cycles that are not restricted to the billing or accounting cycles.

You can use flexible cycles to set up cycle forward fees to grant free resources, provide discounts, or charge fees at any time during the accounting cycle. For example, you can set up a cycle forward fee to grant free minutes every week or every day rather than once a month. Or you can set up a monthly cycle forward fee to grant free minutes on the 15th of every month, which is different from the monthly accounting cycle that begins the 1st of every month.

Configuring Flexible Cycles and Cycle Forward Fees

You set up flexible cycles by configuring flexible cycle forward events.

To set up flexible cycles:

1. Define a custom cycle forward event subclass by using Storable Class Editor in Developer Center.

For example, to define a cycle forward event that occurs every 10 days, create `/event/billing/product/fee/cycle/cycle_forward_10days`.

Tip: Use the `/event/billing/product/fee/cycle/cycle_forward_quarterly` object specification as a model.

2. Map the new event to a valid purchase level:
 - a. In the event map configuration file (`BRM_Home/sys/data/pricing/example/pin_event_map`), add an entry for the new cycle forward event. The entry must use the following format:

```
purchase_level:event_type:event_description:count: unit
```

where:

`count` specifies the frequency of the cycle. It must be a positive number.

`unit` must be **day**, **week**, **month**, or **year**.

For example, to map a biannual (24-month duration) cycle forward event to an account-level purchase type, the `pin_event_map` entry is:

```
/account:/event/billing/product/fee/cycle/cycle_forward_biannual:Biannual
Cycle Forward Event:24:month
```

or

```
/account:/event/billing/product/fee/cycle/cycle_forward_biannual:Biannual
Cycle Forward Event:2:year
```

See "Creating Service and Event Storable Classes" in *BRM Developer's Guide*.

- b. Run the `load_event_map` utility. For information on `load_event_map`, see *BRM Setting Up Pricing and Rating*.
3. Map the new event type to a valid ratable usage metric (RUM):
 - a. In the usage map configuration file, add an entry for the new cycle forward event. For example:

```
/event/billing/product/fee/cycle/cycle_forward_biannual:Occurrence: 0: 0:
0: 0: 0: 0: 0: cycle_forward_biannual
```

See "Mapping Event Types to RUMs" in *BRM Setting Up Pricing and Rating*.

- b. Run the `load_usage_map` utility. For information on `load_usage_map`, see *BRM Setting Up Pricing and Rating*.
4. Stop and restart the CM. For more information, see "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.
5. Use Pricing Center to create products and deals that include your new cycle forward fee.

Charging Cycle Forward Fees Associated with Flexible Cycles

Cycle forward fees are charged when you run monthly billing or by running the `pin_cycle_forward` billing utility.

When a cycle forward event is generated, balance impacts are applied using resource-level validity dates.

If a cycle forward event balance impact for a non-currency resource is set up with a relative cycle start date, balance impacts are applied either to the current cycle or to a future cycle.

For example, if the relative cycle is set to **1** and the cycle is from 1/1/04 to 3/1/04, a sub-balance is created with a validity period from 1/1/04 to 3/1/04. If the relative cycle is set to **2**, a sub-balance is created with a validity period from 3/1/04 to 5/1/04.

Prorating Cycle Forward Fees Associated with Flexible Cycles

Because flexible cycles are not aligned with accounting cycles, cycle forward fees are prorated based on cycle start and end dates. For more information, see ["Calculating Prorated Cycle Fees"](#).

Calculating Product 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 product's purchase date has been backdated, you can use the CM configuration file `calc_cycle_from_cycle_start_t` entry to calculate product fees based on the product's purchase date. This feature is useful when activating an inactive product.

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 product. However, the cycle start time is aligned only after short and long billing cycle differences are considered. For more information, see "[Specifying How to Handle Partial Accounting Cycles](#)".

To set the product 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 product cycle start time to consider the date specified in the PIN_FLD_CYCLE_START_T field for calculating the cycle fees.

3. Save and close the file.

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

Customizing Accounting Cycles

To customize accounting cycles, use the PCM_OP_BILL_POL_SPEC_FUTURE_CYCLE policy opcode.

This policy opcode is called from the PCM_OP_BILL_MAKE_BILL or the PCM_OP_CUST_SET_BILLINFO opcode whenever BRM calculates PIN_FLD_ACTG_NEXT_T and PIN_FLD_ACTG_FUTURE_T.

By default, PIN_FLD_ACTG_NEXT_T is calculated if you do not specify it in the input flist, but PIN_FLD_ACTG_FUTURE_T will always be calculated based on PIN_FLD_ACTG_NEXT_T.

The PCM_OP_BILL_POL_SPEC_FUTURE_CYCLE policy opcode can be modified to calculate the next and future accounting cycles appropriate for your business policy.

Note: To customize the time interval for applying cycle forward and cycle arrears fees for a specified product, use the PCM_OP_SUBSCRIPTION_POL_SPEC_CYCLE_FEE_INTERVAL policy opcode. See "Customizing the Cycle Interval for Products" in *BRM Setting Up Pricing and Rating*.

Customizing How to Bill Events That Occur Between Billing Cycles

Use the PCM_OP_ACT_POL_CONFIG_BILLING_CYCLE policy opcode to specify in which billing cycle to apply an event when an event occurs between the end of a billing cycle and when billing applications are run.

By default, this policy opcode selects the current month's bill, but you can customize this policy opcode to select the previous month's bill.

You specify how long after the billing cycle ends that new events are considered for the previous month's bill by using the **config_billing_cycle** entry in the CM configuration (**pin.conf**) file:

```
config_billing_cycle value
```

Set the **config_billing_cycle** entry to specify how long after the end of the billing cycle the new events are considered for the previous month's bill.

The **PCM_OP_ACT_POL_CONFIG_BILLING_CYCLE** policy opcode is called by the **PCM_OP_ACT_USAGE** opcode when the value of the **config_billing_cycle** entry is greater than 0 and less than or equal to the value of **config_billing_delay**.

If the **config_billing_cycle** value is greater than the **config_billing_delay** value, the CM returns an error.

You can customize the **PCM_OP_ACT_POL_CONFIG_BILLING_CYCLE** policy opcode to point qualifying events to either the previous month's bill or the current month's bill.

Enabling Product Priority While Applying Cycle Fees

When there are multiple products in a deal with cycle fees, you can configure BRM to apply cycle fees in the order of product priority by using the **UsePrioritySubscriptionFees** subscription business parameter.

You can apply cycle fees based on product priority during:

- Purchase or cancellation of a deal, for all the products per deal.
- Billing, for all the products in a deal per bill unit (**/billinfo**).

Note: This parameter does not prioritize products for cycle fees applied using **pin_cycle_fees -defer_cancel** and does not prioritize products for any customized products.

To enable product priority while applying cycle fee:

1. Go to the **BRM_Home/sys/data/config** directory, where **BRM_Home** is the directory in which you installed BRM.
2. Run the following command, which creates 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 the XML file named **bus_params_subscription.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_subscription.xml.out** file in a text editor.
4. Search the file for the following line:

```
<UsePrioritySubscriptionFees>disabled</UsePrioritySubscriptionFees>
```

By default, the **UsePrioritySubscriptionFees** parameter is disabled.

5. Change **disabled** to **enabled**.

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

6. Save this file as **bus_params_subscription.xml**.

7. Go to the *BRM_Home/sys/data/config* directory.

8. Load this change into the appropriate **/config/business_params** object by running the following command:

```
pin_bus_params PathToWorkingDirectory/bus_params_subscription.xml
```

where *PathToWorkingDirectory* is the directory in which **bus_params_subscription.xml** resides.

9. Read the object with the **testnap** utility or Object Browser to verify that all fields are correct.

For more information, see the descriptions about using **testnap** and about reading objects by using Object Browser in *BRM Developer's Guide*.

10. Stop and restart the Connection Manager (CM). For more information, see the description of starting and stopping the BRM System in *BRM System Administrator's Guide*.

11. (Multischema systems only) Run the **pin_multidb** script with the **-R CONFIG** parameter. For more information, see "pin_multidb" in *BRM System Administrator's Guide*.

About Calculating Charges When You Change the Rate

When you change the rate for cycle events in the middle of a cycle and add a new rate tier for the product associated with the event, you can ensure that the two different rates are applied and prorated correctly for the periods they are valid. You can change the rate for cycle events:

- In the current cycle. See ["About Rate Changes in the Current Cycle"](#).
- In a future cycle. See ["About Rate Changes in a Future Cycle"](#).

About Rate Changes in the Current Cycle

If you change the rate for cycle forward or cycle forward arrears events: that is, after the cycle fees have been charged: you use the **pin_rate_change** utility to notify BRM about the change. When you run the **pin_rerate** utility, BRM recalculates the cycle fees by applying the old rate to the part of the cycle before the rate change and the new rate to the part of the cycle after the rate change. BRM adjusts the balance impact accordingly.

For example, suppose the cycle is from April 15 to May 15 and the cycle fee is \$10. A cycle fee of \$10 is charged to the account. Suppose you change the cycle fee from \$10 to \$20 on April 30. A new rate tier at \$20, which is valid from April 30, is added to the product. When you run **pin_rate_change** and then **pin_rerate**, BRM recalculates the cycle fees for the accounts affected by the rate change as follows:

- Refunds \$10 of the old cycle fee.
- Recalculates the cycle fees using the rate of \$10 for the first 14 days and the rate of \$20 for the next 16 days in the cycle:
$$(\$10 \times 14/30) + (\$20 \times 16/30) = \$15.33$$

For more information on **pin_rate_change**, see **pin_rate_change** in *BRM Setting Up Pricing and Rating*.

For more information on rerating, see "About Comprehensive Rerating Using pin_rerate" and **pin_rerate** in *BRM Setting Up Pricing and Rating*.

About Rate Changes in a Future Cycle

If you change the rate for a cycle arrears event for which the cycle fees are charged at the end of the cycle or if you schedule a rate change for a future cycle, the cycle forward and cycle arrears functions use the two different rates and calculate the charges correctly.

When you configure BRM to use multiple rates in a cycle, BRM correctly calculates and prorates other real-time events such as discounts, product purchases, product cancellations, and line transfers by using the appropriate rate for the period when the event occurs.

Calculating Charges When You Change the Rate in a Cycle

To calculate charges correctly when the rate changes in the middle of a cycle, perform these tasks:

1. [Configuring BRM to Apply Multiple Rates in a Cycle](#).
2. [Configuring Event Notification for Rate Changes](#).
3. [Creating Rerating Requests When You Change the Rate](#).
4. [Recalculating the Cycle Fees When the Rate Changes](#). Perform this task only if the change occurs in the current cycle for cycle forward and cycle forward arrears events.

Note: Charges for cycle arrears events and for events in future cycles are calculated automatically by the cycle arrears and cycle forward functions.

Configuring BRM to Apply Multiple Rates in a Cycle

To enable BRM to apply multiple rates in a cycle and to generate rate change events:

1. Open the CM) configuration file (*BRM_Home/sys/cm/pin.conf*) in a text editor.
2. Ensure that the following entries are set to 1:
 - fm_subscription rate_change 1
 - fm_price log_price_change event 1

Important: To apply only one rate and not multiple rates in future cycles, disable these entries by setting them to 0. Using a single rate is more performance efficient than using multiple rates.

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*.

Configuring Event Notification for Rate Changes

When a rate changes in the middle of a cycle, BRM uses event notification to call the PCM_OP_SUBSCRIPTION_PREP_RATE_CHANGE opcode, which creates a `/rate_change` object in the database.

To enable BRM to do this, you must configure the event notification feature as follows:

1. If your system has multiple configuration files for event notification, merge them. See "Merging Event Notification Lists" in *BRM Developer's Guide*.
2. Ensure that the merged file includes the following information from the `BRM_Home/sys/data/config/pin_notify` file:

```
# Rate change related event notification
3768 0 /event/audit/price/product_update
3768 0 /event/audit/price/product_complete
```

3. (Optional) If necessary to accommodate your business needs, add, modify, or delete entries in your final event notification list. See "Editing the Event Notification List" in *BRM Developer's Guide*.
4. (Optional) If necessary to accommodate your business needs, create custom code for event notification to trigger. See "Triggering Custom Operations" in *BRM Developer's Guide*.
5. Load your final event notification list into the BRM database. See "Loading the Event Notification List" in *BRM Developer's Guide*.

For more information, see "Using Event Notification" in *BRM Developer's Guide*.

Configuring Event Notification to Create Rerating Requests for Rate Changes

When you change the rates in the middle of the current cycle for cycle forward and cycle forward arrears events, you must configure event notification to trigger the creation of rerating requests when you run `pin_rate_change`.

When you run `pin_rate_change`, it calls the PCM_OP_SUBSCRIPTION_RATE_CHANGE opcode; this opcode returns a notification event of type `/event/notification/rate_change` for each account picked up by `pin_rate_change`. Depending on how automatic rerating is configured, the notification event triggers the creation of rerating requests.

To enable BRM to do this, you must configure the event notification feature as follows:

1. If your system has multiple configuration files for event notification, merge them. See "Merging Event Notification Lists" in *BRM Developer's Guide*.
2. Ensure that the merged file includes the following information from the `BRM_Home/sys/data/config/pin_notify` file:

```
# Rerating related event notification
3787 0 /event/notification/rate_change
```

3. (Optional) If necessary to accommodate your business needs, add, modify, or delete entries in your final event notification list. See "Editing the Event Notification List" in *BRM Developer's Guide*.

4. (Optional) If necessary to accommodate your business needs, create custom code for event notification to trigger. See "Triggering Custom Operations" in *BRM Developer's Guide*.
5. Load your final event notification list into the BRM database. See "Loading the Event Notification List" in *BRM Developer's Guide*.

For more information, see "Using Event Notification" in *BRM Developer's Guide*.

Creating Rerating Requests When You Change the Rate

When you change the rates in the middle of the current cycle for cycle forward and cycle forward-arrears events, you must run **pin_rate_change** to create rerating requests.

Note: For rate changes in a *future* cycle or cycle arrears events, you do not need to run this utility.

Important: To use only one rate for a cycle and not multiple rates, do not run **pin_rate_change**.

Enter this command to run the utility:

```
pin_rate_change -v -d
```

For more information, see **pin_rate_change** in *BRM Setting Up Pricing and Rating* and "[Configuring Event Notification to Create Rerating Requests for Rate Changes](#)".

Recalculating the Cycle Fees When the Rate Changes

To recalculate the cycle fees and adjust the balance impacts for the accounts affected by the rate change, run **pin_rerate**. See **pin_rerate** in *BRM Setting Up Pricing and Rating*.

Prorating Different Resources When the Rate Changes

When you configure BRM to use multiple rates in a cycle, if the rate changes in the middle of the cycle, BRM automatically prorates the non-currency resources, such as free minutes, for cycle events according to the rate applicable for the period in the cycle. You can configure BRM to use multiple rates for one type of resource, for example currency, and a single rate for another type of resource, for example free minutes.

To rate resources differently:

1. Create two products. For example, one for currency balance impacts and another for non-currency balance impacts.
2. Set one to prorate.
3. Set the other to be charged in full.

Using 31-Day Billing

By default, you can set the billing DOM to any day between 1 and 28. If your customer signs up on the 29th, 30th, or 31st, the billing DOM gets set to the 1st. This is done because all months do not have these days. This can result in a large number of customers being billed on the 1st of the month.

You can change this default 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.

About Setting the Alternate Billing Day of Month

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 to the 1st of the next month.

For example, if a customer registers on March 31:

- The set to back option sets the following billing dates:
 - March 31
 - April 30
 - May 31

In this example, because April does not have 31 days, the billing DOM is on the last day of April.

- The set to forward option sets the following billing dates:
 - March 31
 - May 1
 - May 31

In this example, because April does not have 31 days, the billing DOM is on the first day of the following month, May.

Tip: To set the billing DOM to always be the last day of the month, set it to 31 and use the set to back option.

Note:

- Using these special days means that the billing DOM varies from month to month in a calendar year.
- The general ledger (G/L) earned and unearned report accounts for the variation in the number of days in different accounting cycles.
- The cycle fees are charged in full regardless of how many days there are in a month. Cycle fees will be prorated only in special cases; for example, if you cancel a service in the middle of a month or if you register in the middle of a month and your billing DOM is different from the date of account creation, the cycle fee may be prorated for such months. See ["Calculating Prorated Cycle Fees"](#) for details on proration.
- If 31-day billing feature is not enabled, by default, billing DOM is set to the 1st of the month. For example, if your customer signs up on October 29th, the billing DOM is set to December 1st instead of November 29th. Consequently, the period for which cycle fees is calculated is greater than one unit interval, and the cycle fee charged is greater than the cycle fee amount.

Setting the 31-Day Billing Feature

By default, billing does **not** use the special days 29th, 30th, and 31st. To use the special days, you must either modify the `init_objects.source` file before loading it into the database or modify the `/config/fld_validate` object using `testnap`.

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>
```

Important: 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, use the **testnap** utility to set the Forward or Back billing option in the **/config/business_params** object.

For instructions on how to find this object and change the value, see "Reading an Object and Fields" and "Modifying Objects" in *BRM Developer's Guide*.

In this example, PIN_FLD_MAXIMUM is set to **31**, indicating that BRM will use 31-day billing:

```
0 PIN_FLD_POID                POID [0] 0.0.0.1 /config/fld_validate 606
0 PIN_FLD_VALIDATION          SUBSTRUCT [0]
1 PIN_FLD_MAXIMUM             DECIMAL [0] 31
```

Tip: To verify that you changed the field, read the object by using the **testnap** utility or by displaying the **/config/business_params** object in Object Browser. See "Reading an Object and Fields" in *BRM Developer's Guide*.

Important: Stop and restart the CM after editing the object. See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

Setting the Forward and Back Billing Options

By default, the billing DOM is set to the 1st of the next month. If you use 31-day billing, you can choose to run billing on the last day for the same month (set to back option) or the first day of the next month (set to forward option). See "[About Setting the Alternate Billing Day of Month](#)".

You configure the billing DOM by modifying a field in the **billing** instance of the **/config/business_params** object.

Important: You cannot set this parameter differently for different brands.

You modify the **/config/business_params** object by using the **pin_bus_params** utility. For information on this utility, see **pin_bus_params** in *BRM Developer's Guide*.

To configure the billing DOM to be last day of the month or 1st of the next month:

1. 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 working directory. If you do not want this file in your working directory, specify the full path as part of the file name.

2. Search the XML file for the following line:

```
<MoveDayForward>firstDay</MoveDayForward>
```

3. Change **firstDay** to **lastDay**.

Caution: BRM uses the XML in this file to overwrite the existing **billing** 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 BRM's billing configuration.

4. Save the file as **bus_params_billing.xml** and close the file.
5. Go to the `BRM_Home/sys/data/config` directory which includes support files used by the **pin_bus_params** utility.
6. Use the following command to load this change into the appropriate `/config/business_params` object.

```
pin_bus_params PathToWorkingDirectory/bus_params_billing.xml
```

where *PathToWorkingDirectory* is the directory in which **bus_params_billing.xml** resides.

Note: To execute it from a different directory, see the description of **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 testnap" in *BRM Developer's Guide*. For information on how to use Object Browser, see "Reading Objects by Using Object Browser" 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*.
9. (Multischema systems only) Run the **pin_multidb** script with the **-R CONFIG** parameter. For more information, see "pin_multidb" in *BRM System Administrator's Guide*.

Customizing Bill and Invoice Numbers

For information about customizing the way BRM generates bill and invoice numbers, see the following:

- [About Bill Numbering in BRM](#)
- [About Numbering Corrective Bills in BRM](#)

Customizing the Format of Bill and Invoice Numbers

You can customize the prefix and the numbering sequence for bills.

To use a different bill number format, use `PCM_OP_WRITE_FLDS` to modify the `PIN_FLD_HEADER_STR` field in the `/data/sequence` object. For example, to use a bill number format with numbers only and no letters, such as 100, 101, 102, and so on, set `PIN_FLD_HEADER_STR` to two colons (`::`). For information about modifying fields in an object, see "Writing Fields in Objects" in *BRM Developer's Guide*.

Use the `PCM_OP_BILL_POL_SPEC_BILLNO` policy opcode to customize bill numbers. The `PCM_OP_BILL_POL_SPEC_BILLNO` policy opcode assigns a default number to the `/account` storable object in the database. This policy opcode is called by

the `pin_fld_billno` utility and uses the default implementation information to create a unique billing number. The billing number is then returned to the storable object in the database. For information on `PCM_OP_BILL_POL_SPEC_BILLNO`, see *BRM Developer's 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. See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

Configuring Bill Now

For information about Bill Now, see the following:

- [About Bill Now](#)
- [How Bill Now Works](#)

Selecting the Input for Bill Now

By default, Bill Now generates a bill that includes all pending items. You can customize Bill Now to include only specified pending items. To change the default behavior, edit the search criteria in the `PCM_OP_BILL_POL_GET_PENDING_ITEMS` policy opcode.

Important: If you run Bill Now on a subordinate `/billinfo`, a bill is created for the parent `/billinfo` that only includes the items from the subordinate `/billinfo`. If you run Bill Now on a parent `/billinfo`, a bill is created that contains a total of the items from both the parent and any subordinate `/billinfo` objects.

Changing the Bill Now Due Date

The default due date for a bill created with Bill Now is calculated as the billing cycle length minus one day after Bill Now is run:

$$\text{date_of_bill} + \text{billing_cycle_length} - \text{one_day}$$

For example, if you run Bill Now on June 2, and the billing cycle is one month, the bill is due July 1.

To change the Bill Now due date to, for example, `date_of_bill + billing_cycle_length - 7` days, you customize the `PCM_OP_BILL_POL_CALC_PYMT_DUE_T` policy opcode.

Providing Discounts to Closed Accounts

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 "Specifying to Delete Canceled Discounts" in *BRM Managing Customers*.

1. Open the CM configuration file (`BRM_Home/sys/cm/pin.conf`) in a text editor.
2. Set the `keep_cancelled_products_or_discounts` entry to 1:

```
- fm_subscription_pol keep_cancelled_products_or_discounts 1
```

If this entry is not present, add it.

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*.

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 modifying a field in the `billing` instance of the `/config/business_params` object.

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.

You modify the `/config/business_params` object by using the `pin_bus_params` utility. For information on this utility, see `pin_bus_params` in *BRM Developer's Guide*.

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

1. 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 working directory. If you do not want this file in your working directory, specify the full path as part of the file name.

2. Search the XML file for the following line:

```
<ApplyCycleFeeForBillNow>disabled</ApplyCycleFeeForBillNow>
```

3. Change `disabled` to `enabled`.

Caution: BRM uses the XML in this file to overwrite the existing `billing` 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 BRM's billing configuration.

4. Save the file as **bus_params_billing.xml** and close the file.
5. Go to the *BRM_Home/sys/data/config* directory which includes support files used by the **pin_bus_params** utility.
6. Use the following command to load this change into the appropriate */config/business_params* object.

```
pin_bus_params PathToWorkingDirectory/bus_params_billing.xml
```

where *PathToWorkingDirectory* is the directory in which **bus_params_billing.xml** resides.

Note: To execute it from a different directory, see the description for **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 testnap" in *BRM Developer's Guide*. For information on how to use Object Browser, see "Reading Objects by Using Object Browser" 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*.
9. (Multischema systems only) Run the **pin_multidb** script with the **-R CONFIG** parameter. For more information, see "pin_multidb" in *BRM System Administrator's Guide*.

Creating Two Bills during the Delayed Billing Period

By default, generating two bills with Bill Now during the delayed billing period is disabled in BRM. You enable this feature by modifying a field in the **billing** instance of the */config/business_params* object.

You modify the */config/business_params* object by using the **pin_bus_params** utility. For more information, see the description of **pin_bus_params** in *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. 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 working directory. If you do not want this file in your working directory, specify the path as part of the file name.

2. Search the XML file for the following line:

```
<CreateTwoBillNowBillsInDelay>disabled</CreateTwoBillNowBillsInDelay>
```

3. Change **disabled** to **enabled**.

Caution: BRM uses the XML in this file to overwrite the existing **billing** 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 BRM's billing configuration.

4. Save the file as **bus_params_billing.xml** and close the file.
5. Go to the *BRM_Home/sys/data/config* directory which includes support files used by the **pin_bus_params** utility.
6. Use the following command to load this change into the appropriate **/config/business_params** object.

```
pin_bus_params PathToWorkingDirectory/bus_params_billing.xml
```

where *PathToWorkingDirectory* is the directory in which **bus_params_billing.xml** resides.

Note: To execute it from a different directory, see the description of **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 testnap" in *BRM Developer's Guide*. For information on how to use Object Browser, see "Reading Objects by Using Object Browser" 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*.
9. (Multischema systems only) Run the **pin_multibd** script with the **-R CONFIG** parameter. For more information, see "pin_multibd" in *BRM System Administrator's Guide*.

Calculating Deferred Taxes with Bill Now

When you run Bill Now on a parent **/billinfo** that includes subordinate **/billinfo** objects, Bill Now ignores the **cycle_tax_interval** entry in the CM configuration file. Normally, the entry specifies whether deferred taxes are calculated separately for the parent and each subordinate **/billinfo** or are consolidated into a single item for the parent. See "About Tax Calculation for Account Groups" in *BRM Calculating Taxes*.

With Bill Now, no matter which option you select, it always rolls activities for each subordinate 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 subordinate bill units.

Customizing Bill Now

You can use the **PCM_OP_BILL_POL_GET_PENDING_ITEMS** policy opcode to select only those pending items you want to be used by **PCM_OP_BILL_MAKE_BILL_NOW** opcode.

By default, the **PCM_OP_BILL_POL_GET_PENDING_ITEMS** opcode selects all pending items and passes them back to **PCM_OP_BILL_MAKE_BILL_NOW**.

If the bill is produced for the parent **/billinfo** object, this bill, by default, includes pending items from the parent and all subordinate **/billinfo** objects. To include items for just one of the subordinate **/billinfo** objects, add functionality to the PCM_OP_BILL_POL_GET_PENDING_ITEMS policy opcode to filter out the rest of the items for the **/billinfo** objects associated with the parent **/billinfo**.

Applying Discounts and Folds with Bill Now

To apply discounts or folds, your customer management application needs one of the values listed in [Table 3–1](#) in the PIN_FLD_FLAGS field in the PCM_OP_BILL_MAKE_BILL_NOW input list:

Table 3–1 Values to Apply Folds, Discounts, or Both

To include:	Set the Value To:
Folds	16
Discounts	32
Folds <i>and</i> discounts	48 (You add the values to get both actions.)

Important:

- A billing-time discount is not allowed when Bill Now includes only selected items or a specific service.
- When the billing-time discount flag is specified, you must ensure that a fold is configured for the specific resource used by the billing-time discount. In addition to the billing-time discount flag, you must also specify the fold flag to fold the resources used by the discount. If a fold is not configured for the resource used by the discount or the fold flag is not specified, the billing-time discount is applied again during regular billing.
- When the fold flag is specified, all resources are folded, even those not used by the billing-time discount. For example, when closing an account, you can specify the fold flag to fold all the resources. In other cases, you may not want to fold all the resources. In such cases, do not specify the billing-time discount or fold flag if there are other resources that are not used by the billing-time discount.

Running Bill Now for a Service

You can extend your customer management application to generate a Bill Now type of bill for a specific service. When the selected service belongs to a sponsored account, a bill can be generated for the sponsoring account of the charge sharing group.

For information on the relevant opcodes, see the following:

- PCM_OP_BILL_MAKE_BILL_NOW
- PCM_OP_BILL_CREATE_SPONSORED_ITEMS
- PCM_OP_BILL_POL_GET_PENDING_ITEMS

Disabling Auto-Triggered Billing

To disable auto-triggered billing, you must specify the following:

1. Billing delay interval. See ["Disabling Auto-Triggered Billing by Specifying Billing Delay"](#).
2. Value of **AutoTriggeringLimit** entry. See ["Disabling Auto-Triggered Billing by Setting AutoTriggeringLimit"](#).

For more information, see ["About Auto-Triggered Billing"](#).

Disabling Auto-Triggered Billing by Specifying Billing Delay

To disable auto-triggered billing, you must specify billing delay even if you do not use delayed billing.

Note: If you do not use delayed billing, you can set the billing delay interval to 0.

When billing delay is specified, the system maintains an internal list of bill items for both the previous billing cycle and the next billing cycle so that new events impact bill items of the next billing cycle and old events impact bill items of the previous billing cycle.

Set or specify the billing delay as follows:

1. Open the CM configuration file (*BRM_Home/sys/cm/pin.conf*) in a text editor.
2. Uncomment the **config_billing_delay** entry.

Caution: You can change the value of the **config_billing_delay** entry (for example, change the billing delay interval from 5 days to 3 days) at any time. However, after you begin rating events in a production database, do not comment or uncomment the **config_billing_delay** entry. Doing so might cause database errors.

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

3. Save and close the file.
4. Stop and start the CM. See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.
5. Open the billing application configuration file (*BRM_Home/apps/pin_billd/pin.conf*).
6. Set the **config_billing_delay** entry to the same value as in the CM **pin.conf** file.
7. Save and close the file.

Disabling Auto-Triggered Billing by Setting AutoTriggeringLimit

To disable auto-triggered billing, you must also set the **AutoTriggeringLimit** entry to be greater than 0. When auto-triggered billing is disabled, the **AutoTriggeringLimit** value is used as a precaution to trigger billing when the previous billing run is still pending and next billing is imminent.

By default, **AutoTriggeringLimit** is set to 2. For example, if billing for the previous cycle has not occurred and the billing for the next cycle is due in the next two days, then billing for the previous cycle is auto-triggered within these two days.

To change the **AutoTriggeringLimit** value, do the following:

1. Use the following command to create an editable XML file from **billing** instance of 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. If you do not want this file in your working directory, specify the full path as part of the file name.

2. Search the XML file for the following line:

```
<AutoTriggeringLimit>N</AutoTriggeringLimit>
```

To disable auto-triggered billing, set *N* 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.

Caution: BRM uses the XML in this file to overwrite the existing **billing** 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 BRM's billing configuration.

3. Save the file as **bus_params_billing.xml** and close the file.
4. Go to the *BRM_Home/sys/data/config* directory which includes support files used by the **pin_bus_params** utility.
5. Use the following command to load this change into the appropriate **/config/business_params** object.

```
pin_bus_params PathToWorkingDirectory/bus_params_billing.xml
```

where *PathToWorkingDirectory* is the directory in which **bus_params_billing.xml** resides.

Note: To execute it from a different directory, see the description of **pin_bus_params** in *BRM Developer's Guide*.

6. Read the object with the **testnap** utility or Object Browser to verify that all fields are correct.

For general instructions on using **testnap**, see "Using testnap" in *BRM Developer's Guide*. For information on how to use Object Browser, see "Reading Objects by Using Object Browser" in *BRM Developer's Guide*.

7. Stop and restart the CM. For more information, see "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.
8. (Multischema systems only) Run the **pin_multidb** script with the **-R CONFIG** parameter. For more information, see "pin_multidb" in *BRM System Administrator's Guide*.

Auto-triggered billing is now disabled for all but the delay interval and the last N days of each bill unit's accounting cycle. If the delay interval is set to 0, auto-triggered billing is disabled for all but the last N days of the bill unit's accounting cycle.

For more information, see ["About Auto-Triggered Billing"](#).

Setting Up Delayed Billing

Important: If you set up 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. For more information, see ["Enabling Rerating and Rollover Correction Due to Delayed Events"](#).

Configuring Delayed Billing

To set up delayed billing, you must specify the following:

- Billing delay interval. See ["Specifying the Billing Delay Interval"](#).
- Value of **AutoTriggeringLimit** entry. See ["Specifying Auto-Triggered Billing for Delayed Billing"](#).

For information about delayed billing, see ["About Delayed Billing"](#).

Specifying the Billing Delay Interval

Set the billing delay interval as follows:

1. Open the CM configuration file (*BRM_Home/sys/cm/pin.conf*) in a text editor.
2. If necessary, uncomment the **config_billing_delay** entry.
The entry is enabled (uncommented) by default.
3. Change the value of the **config_billing_delay** entry to specify the delay interval:

```
config_billing_delay D[:H]
```

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

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

For example:

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

Caution: You can change the value of the `config_billing_delay` entry (for example, change the billing delay interval from 5 days to 3 days) at any time. However, after you begin rating events in a production database, do not comment or uncomment the `config_billing_delay` entry. Doing so might cause database errors.

4. Save and close the file.
5. Stop and restart the CM. See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.
6. Open the billing application configuration file (`BRM_Home/apps/pin_billd/pin.conf`).
7. Set the value of the `config_billing_delay` entry to the same value as in the CM `pin.conf` file.
8. Save and close the file.

Specifying 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 change the default to enable auto-triggered billing to be enabled for more than two days at the end of each accounting cycle or you can change it to be *always enabled* when delayed billing is used.

To change the default, use the `pin_bus_params` utility to modify the `AutoTriggeringLimit` parameter in the `billing` instance of the `/config/business_params` object. See *BRM System Administrator's Guide* for information on `pin_bus_params`.

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. 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 the XML file named `bus_params_billing.xml.out` in your working directory. If you do not want this file in your working directory, specify the full path as part of the file name.

2. Search the XML file for following line:

```
<AutoTriggeringLimit>2</AutoTriggeringLimit>
```

3. To change the number of days for which auto-triggered billing is enabled at the *end* of each accounting cycle, change `2` to a number greater than `0` and less than one 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.

Caution: BRM uses the XML in this file to overwrite the existing **billing** 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 BRM's billing configuration.

4. 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.
5. Save the file as **bus_params_billing.xml** and close the file.
6. Go to the `BRM_Home/sys/data/config` directory which includes support files used by the `pin_bus_params` utility.
7. Use the following command to load this change into the appropriate `/config/business_params` object.

```
pin_bus_params PathToWorkingDirectory/bus_params_billing.xml
```

where *PathToWorkingDirectory* is the directory in which **bus_params_billing.xml** resides.

Note: To execute it from a different directory, see the description for **pin_bus_params** in *BRM Developer's Guide*.

8. Read the object with the **testnap** utility or Object Browser to verify that all fields are correct.

For general instructions on using **testnap**, see "Using testnap" in *BRM Developer's Guide*. For information on how to use Object Browser, see "Reading Objects by Using Object Browser" 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. (Multischema systems only) Run the **pin_multidb** script with the **-R CONFIG** parameter. For more information, see "pin_multidb" in *BRM System Administrator's Guide*.

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 roll over unused resources 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. For more information, see "[About Delayed Billing](#)".

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

Important: New events that occur inside the billing delay interval are rated and recorded for the next billing cycle. If cycle forward fees and rollover resources 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 and close 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. For more information about delayed billing, see ["About Delayed Billing"](#).

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 **config_billing_delay** entry is specified and set to a number greater than zero.

3. Save and close 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 deal 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 deals 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 and close the file.
4. Stop and restart the CM. For more information, see "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

Billing Cycle Override

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. Open the CM configuration file (*BRM_Home/sys/cm/pin.conf*) in a text editor.
2. Set the `config_billing_cycle` entry to the length of the configurable billing cycle interval.
3. For example, the following sets the interval to 5 days:

```
- config_billing_cycle 5
```

Important: The `config_billing_cycle` value must be greater than zero and less than or equal to the `config_billing_delay` (delayed billing interval value). Otherwise, BRM reports an error and terminates the CM.

Note: 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.

4. 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 PCM_OP_ACT_POL_CONFIG_BILLING_CYCLE.

Using Pipeline Manager to Trigger Billing

When you use pipeline batch rating, if an account is currently in the process of being billed, incoming call records are suspended (not rated) for that account until its billing is complete. The number of accounts being billed affects the time it takes to complete the billing process. When you need accounts to be billed quickly so that their new usage can be rated, you can set up Pipeline Manager to trigger billing. This will reduce the number of call records that might need suspending. When Pipeline Manager triggers billing for an account, it is billed in a separate billing process.

To set up pipeline-triggered billing, see ["Setting Up Pipeline-Triggered Billing"](#).

Enabling a Billing Delay for Rated Event Loader

If you use Rated Event (RE) Loader to load batch-rated events, RE Loader cannot load a CDR for the next billing cycle unless delayed billing is enabled. For more information, see the discussion on enabling a billing delay for CDRs to be loaded when billing is incomplete in *BRM Configuring and Running Billing*.

Enabling Rerating and Rollover Correction Due to Delayed Events

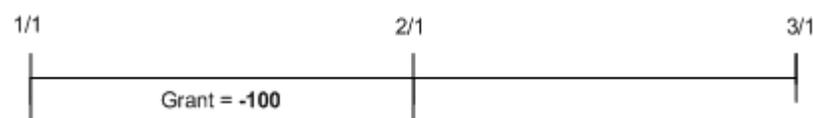
If a delayed event arrives after the end of the accounting cycle and during the delayed billing period, the event can borrow against the rollover of the current cycle even when the current rollover has been consumed by events of the current cycle.

If rerating and rollover correction is enabled and delayed events borrow from the rollover of the current cycle, the current cycle events are rerated and the rollover is reallocated so that it comes from the appropriate cycles.

For example, suppose the billing delay period is configured for 10 days.

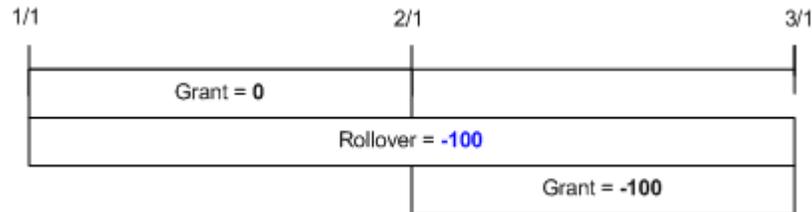
On January 1, a monthly cycle event grants 100 free minutes that are valid from January 1 to February 1 as shown in [Figure 3–8](#).

Figure 3–8 January Cycle Event Grant of 100 Free Minutes



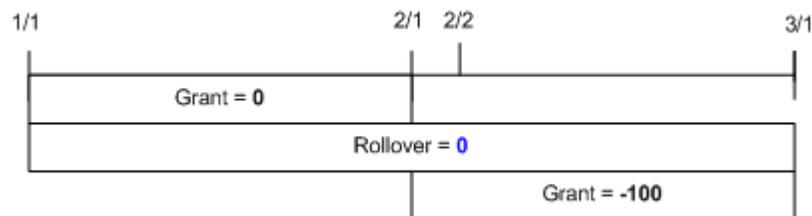
On February 1, the 100 free minutes from January are rolled over to the next cycle and are valid from January 1 to March 1. The cycle event creates a new sub-balance with 100 free minutes valid from February 1 to March 1 as shown in [Figure 3-9](#).

Figure 3-9 January Rollover and February Cycle Event Grant of 100 Free Minutes



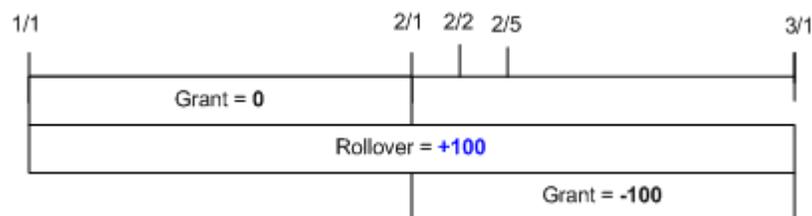
On February 2, events from the current cycle consume the entire 100 free minutes in the rollover bucket as shown in [Figure 3-10](#).

Figure 3-10 100 Free Minutes Consumed from Rollover Bucket

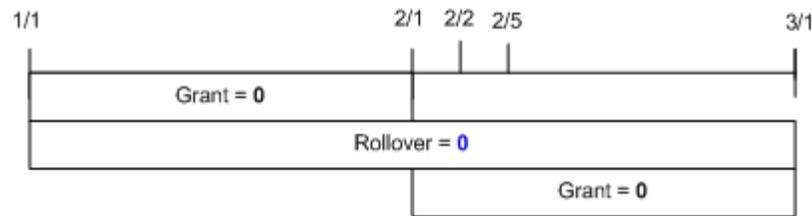


On February 5, a delayed event is rated for 100 minutes. These 100 minutes are consumed from the rollover bucket leaving a balance of +100 minutes as shown in [Figure 3-11](#).

Figure 3-11 Additional 100 Minutes Consumed from Rollover Bucket



When rollover correction is enabled, the current cycle events are rerated and the original balance impacts are backed out. Therefore, 100 minutes are reallocated into the rollover bucket. Because the delayed event caused a balance impact of +100 and rerating caused a balance impact of -100, the rollover bucket balance becomes 0. The current cycle events are rated again and consume from the free resources granted on February 1.

Figure 3–12 Post Rerating Balance Impacts

Important: If rollover correction is not enabled, rerating is not triggered to rerate the current cycle events. Therefore, current cycle events remain rated as free even if their rollover has been consumed by delayed events. By default, rerating and rollover correction for delayed events is disabled.

Note: When delayed call detail records (CDRs) borrow from allocated rollover credit and there is a credit monitoring threshold, the monitoring results may be inaccurate until rerating is run. That is because some current CDRs appear to be free when they are not.

To enable rerating and rollover correction, you must do the following:

- Modify configurable business parameters.
See "[Modifying Business Parameters to Enable Rerating and Rollover Correction](#)".
- Configure event notification.
See "[Configuring Event Notification for Rerating and Rollover Correction](#)".

Modifying Business Parameters to Enable Rerating and Rollover Correction

To enable rerating and rollover correction, you must modify a field in the **billing** instance of the **/config/business_params** object by using the **pin_bus_params** utility as follows (for information on this utility, see **pin_bus_params** in *BRM System Administrator's Guide*):

1. 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 working directory. If you do not want this file in your working directory, specify the full path as part of the file name.

2. Search the XML file for the following line:

```
<RerateDuringBilling>disabled</RerateDuringBilling>
```

3. Change **disabled** to **enabled**.

Caution: BRM uses the XML in this file to overwrite the existing **billing** 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 BRM's billing configuration.

4. Search the XML file for following line:

```
<RolloverCorrectionDuringBilling>disabled</RolloverCorrectionDuringBilling>
```

5. Change **disabled** to **enabled**.
6. Save the file as **bus_params_billing.xml** and close the file.
7. Go to the *BRM_Home/sys/data/config* directory which includes support files used by the **pin_bus_params** utility.
8. Use the following command to load this change into the appropriate **/config/business_params** object.

```
pin_bus_params PathToWorkingDirectory/bus_params_billing.xml
```

where *PathToWorkingDirectory* is the directory in which **bus_params_billing.xml** resides.

Note: To execute it from a different directory, see the description for **pin_bus_params** in *BRM Developer's Guide*.

9. 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 testnap" in *BRM Developer's Guide*. For information on how to use Object Browser, see "Reading Objects by Using Object Browser" in *BRM Developer's Guide*.

10. Stop and restart the CM. For more information, see "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.
11. (Multischema systems only) Run the **pin_multidb** script with the **-R CONFIG** parameter. For more information, see "pin_multidb" in *BRM System Administrator's Guide*.

For information about setting the length of the delayed billing period, see "[Setting Up Delayed Billing](#)".

Configuring Event Notification for Rerating and Rollover Correction

If rerating and rollover correction is enabled and delayed events borrow from the rollover of the current cycle, BRM rerating uses event notification to trigger automatic rerating of the event.

BRM rerating generates the non-persistent **/event/notification/rollover_correction/erate** event specifically to use for event notification in this case.

By default, when this event occurs, BRM creates a **erate** job.

To enable event notification for rerating and rollover correction, you must configure the event notification feature as follows:

1. If your system has multiple configuration files for event notification, merge them. See "Merging Event Notification Lists" in *BRM Developer's Guide*.
2. Ensure that the merged file includes the following information from the *BRM_Home/sys/data/config/pin_notify* file:

```
# Rerating related event notification
3787 0 /event/notification/rollover_correction/rerate
```
3. (Optional) If necessary to accommodate your business needs, add, modify, or delete entries in your final event notification list. See "Editing the Event Notification List" in *BRM Developer's Guide*.
4. (Optional) If necessary to accommodate your business needs, create custom code for event notification to trigger. See "Triggering Custom Operations" in *BRM Developer's Guide*.
5. Load your final event notification list into the BRM database. See "Loading the Event Notification List" in *BRM Developer's Guide*.

For more information, see "Using Event Notification" in *BRM Developer's Guide*.

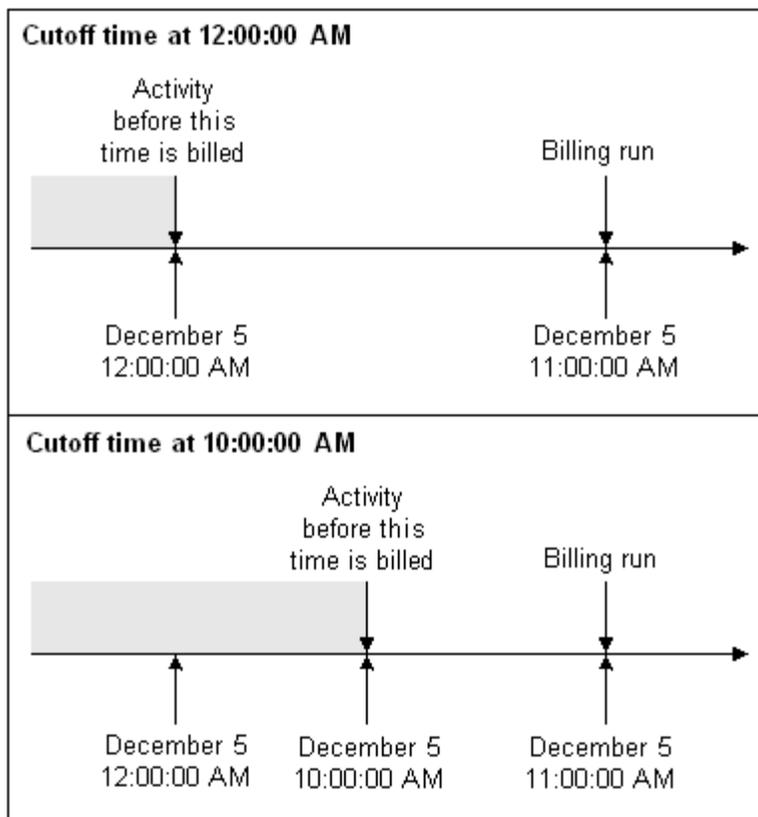
Configuring the BRM Cutoff Time

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 3–13](#) shows how billing works for different cutoff times:

Figure 3–13 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 across all brands.

Important: Ensure that you have a very good reason for changing the cutoff time. Changing the cutoff time can add complexity to your system. For example, if your cutoff time is 10 a.m. and you search for events between February 1 and February 26, the search finds events from 10:00:00 a.m. on February 1 to 9:59:59 a.m. on February 26. Events that occurred before 10:00:00 a.m. on February 1 are not displayed. All events that occurred until 9:59:59 a.m. on February 27 are displayed, even though you searched for events that occurred through February 26.

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 registers for 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 price list element: 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 rates 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 Searches Are Affected by Changing the Cutoff Time

- Event Browser uses the cutoff time for searching. For example, if your cutoff time is 10:00:00 a.m. and you search for events that occurred on December 5, Event Browser displays events that occurred from 10:00:00 a.m. on December 5 to 9:59:59 a.m. on December 6.
- Payment Tool uses the cutoff time to display bill information. For example, if your cutoff time is 10:00:00 a.m. and you search for bills created on December 5, Payment Tool displays bills created from 10:00:00 a.m. on December 5 to 9:59:59 a.m. on December 6.

How TimeStamp Fields Are Affected by Changing the Cutoff Time

Many BRM features use time stamps to determine how to perform activities. Time stamps are usually rounded to midnight. If you change the cutoff time, the time stamp is rounded to the cutoff time instead.

Note: The cutoff time is also considered while setting the time-stamp values for the start and end dates through Customer Center.

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

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

To configure the billing cutoff time, perform the following steps:

1. Open the CM configuration file (*BRM_Home/sys/cm/pin.conf*) in a text editor.
2. Set the **timestamp_rounding** entry to **1**.
3. Save and close the file.

4. Use the following command to create an editable XML file for the **BusParamsBilling** parameter class:

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

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

5. Search the XML file for following line:

```
<BillingCycleOffset>0</BillingCycleOffset>
```

6. Change **0** to the desired cutoff time. For example, to set the cutoff time to 10:00 a.m., change **0** to **10**. The default for this field is **0**, which is equivalent to 12:00 a.m.

BRM uses the XML in this file to overwrite the existing **billing** 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 BRM's billing configuration.

7. Save the file as **bus_params_billing.xml** and close the file.
8. Go to the *BRM_Home/sys/data/config* directory which includes support files used by the **pin_bus_params** utility.
9. Use the following command to load this change into the appropriate **/config/business_params** object.

```
pin_bus_params PathToWorkingDirectory/bus_params_billing.xml
```

where *PathToWorkingDirectory* is the directory in which **bus_params_billing.xml** resides.

Note: To execute it from a different directory, see the description for **pin_bus_params** in *BRM Developer's Guide*.

10. 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 testnap" in *BRM Developer's Guide*. For information on how to use Object Browser, see "Reading Objects by Using Object Browser" in *BRM Developer's Guide*.

11. Stop and restart the CM. For more information, see "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.
12. (Multischema systems only) Run the **pin_multidb** script with the **-R CONFIG** parameter. For more information, see "pin_multidb" in *BRM System Administrator's Guide*.

Setting Up Billing for Sponsorship

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 sponsor groups or resource 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 sponsored and 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 sponsor or resource sharing group owner account.

Caution: If you do not reconfigure your system, sponsor 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 set up billing for sponsor groups, you modify a field in the **billing-flow** instance of the **/config/business_params** object.

You modify the **/config/business_params** object by using the **pin_bus_params** utility. For information on this utility, see **pin_bus_params** in *BRM Developer's Guide*.

Note: The following setups ensures that two-level sponsor relationships are correctly billed. For sponsor relationships that exceed two levels, the billing sequence may be incorrect, thus resulting in incorrect billing.

For example, suppose account A sponsors a group to which account B belongs and account B sponsors a group to which account C belongs. Owner account A and member account B will be 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 will be 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 sponsor groups:

1. Use the following command to create an editable XML file from the **billing-flow** instance of the **/config/business_params** object:

```
pin_bus_params -r BusParamsBillingFlow bus_params_billing_flow.xml
```

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

2. Search the XML file for following line:

```
<BillingFlowSponsorship>undefined</BillingFlowSponsorship>
```

3. Change **undefined** to one of the following:

- **sponsorsFirst** if you want sponsor group accounts to be billed before the member accounts.
- **sponseesFirst** if you want the member accounts to be billed before the sponsor group accounts.

If the billing order of the sponsor group account and member accounts does not matter, keep the original setting of **undefined**.

Caution: BRM uses the XML in this file to overwrite the existing **billing-flow** 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 BRM's billing configuration.

4. Save the file as **bus_params_billing.xml** and close the file.
5. Go to the **BRM_Home/sys/data/config** directory which includes support files used by the **pin_bus_params** utility.
6. Use the following command to load this change into the appropriate **/config/business_params** object.

```
pin_bus_params PathToWorkingDirectory/bus_params_billing.xml
```

where *PathToWorkingDirectory* is the directory in which **bus_params_billing.xml** resides.

Note: To execute it from a different directory, see the description for **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 testnap" in *BRM Developer's Guide*. For information on how to use Object Browser, see "Reading Objects by Using Object Browser" 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*.
9. (Multischema systems only) Run the **pin_multidb** script with the **-R CONFIG** parameter. For more information, see "pin_multidb" in *BRM System Administrator's Guide*.

To set up billing for discount sponsor groups:

1. Use the following command to create an editable XML file from the **billing-flow** instance of the **/config/business_params** object:

```
pin_bus_params -r BusParamsBillingFlow bus_params_billing_flow.xml
```

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

2. Search the XML file for following line:

```
<BillingFlowDiscount>undefined</BillingFlowDiscount>
```

3. Change **undefined** to one of the following:

- **discountParentsFirst** if you want discount group owner accounts to be billed before the member accounts.
- **memberDiscountFirst** if you want the member accounts to be billed before discount group owner accounts.

If the billing order of the discount group owner and member accounts does not matter, keep the original setting of **undefined**.

Caution: BRM uses the XML in this file to overwrite the existing **billing-flow** 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 BRM's billing configuration.

4. Save and close the file.
5. Use the following command to load the change into the **/config/business_params** object:

```
pin_bus_params bus_params_billing_flow.xml
```

You should execute this command from the **BRM_Home/sys/data/config** directory, which includes support files used by the utility. To execute it from a different directory, see **pin_bus_params** in *BRM System Administrator's Guide*.

6. 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 testnap" in *BRM Developer's Guide*. For information on how to use Object Browser, see "Reading Objects by Using Object Browser" in *BRM Developer's Guide*.
7. Stop and restart the CM. See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.
8. Stop and restart the CM. For more information, see "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

For more information about sponsor groups, see "About Account Groups" in *BRM System Administrator's Guide*.

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. See "[Manually Running the pin_bill_day Script](#)".

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_bill/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 and close 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_bill*.

For example, *BRM_Home/apps/pin_bill/db1* for database 1, *BRM_Home/apps/pin_bill/db2* for database2, and so on.

2. Copy the *BRM_Home/apps/pin_bill/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 and close the file.
4. Run the billing utilities from the new subdirectories.

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 account bill units in your system that you do not want to bill.

You can resume billing on suspended accounts.

For information about suspending bill units in your custom application, see "[Suspending and Resuming Billing of Closed Accounts](#)".

Tip: 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.

Note:

- If you suspend a parent account, all nonpaying child accounts 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 "[Reducing Billing Run Loads](#)".
-

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: For hierarchical accounts, the total balance due includes the totals from the subordinate bill unit of their child accounts.

- The account has had no billable activity since the previous bill.

To customize this default behavior, modify the `PCM_OP_BILL_POL_POST_BILLING` policy opcode.

To configure billing of closed accounts:

1. Open the CM configuration file (*BRM_Home/sys/cm/pin.conf*) in a text editor.
2. Do one of the following:
 - To disable billing of closed accounts, set the value of **stop_bill_closed_accounts** entry to **1**:

```
- stop_bill_closed_accounts 1
```
 - To enable billing of closed accounts, set the value of **stop_bill_closed_accounts** to **0**.

By default, billing of closed accounts is enabled.

3. Save and close the file.
4. Stop and restart the CM. For more information, see "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

For information about configuration files, see "Using Configuration Files to Connect and Configure Components" 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. See "[Suspending and Resuming Billing of Closed Accounts](#)".

Setting up Billing in Subordinate Hierarchies

If you use the subordinate hierarchy, BRM validates that all subordinate accounts have been billed successfully before billing the parent account. When billing fails for a subordinate account, the parent account is not billed. However, in rare instances when the subordinate account billing is continuously failing and you want to proceed with parent account billing, you can skip the validation of the subordinate account billing by enabling the **SkipCheckForSubordinatesBilled** parameter in the **billing** instance of the `/config/business_params` object.

You modify the `/config/business_params` object by using the `pin_bus_params` utility. For information on this utility, see "pin_bus_params" in *BRM Developer's Guide*.

To skip validation of the subordinate account billing:

1. Go to the *BRM_home/sys/data/config* directory.
2. Run 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 working directory. If you do not want this file in your working directory, specify the full path as part of the file name.

3. Open the **bus_params_billing.xml.out** file.
4. Search the XML file for the following line:

```
<SkipCheckForSubordinatesBilled>disable</SkipCheckForSubordinatesBilled>
```

5. Change **disable** to **enable**.
 - **disable:** BRM validates that the subordinate accounts have been billed successfully before billing the parent account. This is the default.
 - **enable:** BRM skips the validation of the subordinate account billing and bills the parent account.

Note: When **SkipCheckForSubordinatesBilled** is enabled and subordinate account billing has errors, the parent bill will not include charges from the subordinate account.

Caution: BRM uses the XML in this file to overwrite the existing **billing** 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 BRM's billing configuration.

6. Save the file as **bus_params_billing.xml**.
7. Go to the *BRM_home/sys/data/config* directory.
8. Run the command:

```
pin_bus_params PathToWorkingDirectory/bus_params_billing.xml
```

where *PathToWorkingDirectory* is the directory in which **bus_params_billing.xml** resides.

Note: To run this command from a different directory, see the description for "pin_bus_params" in *BRM Developer's Guide*.

9. 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 testnap" in *BRM Developer's Guide*. For information on how to use Object Browser, see "Reading Objects by Using Object Browser" in *BRM Developer's Guide*.

10. Stop and restart the CM. See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

11. (Multischema systems only) Run the **pin_multidb** script with the **-R CONFIG** parameter. For more information, see "pin_multidb" in *BRM System Administrator's Guide*.

About Proration

This chapter describes basic proration concepts.

Calculating Prorated Cycle Fees

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

If a base product is valid during part of the cycle and one or more customized products is valid during another part of the cycle, BRM calculates the cycle fees for each product based on the proportion of time in the cycle that the products are valid. The total fee charged for the cycle is the sum of the individual charges calculated for each product that is valid during the cycle. See "[Calculating Prorated Cycle Fees and Refunds for Customized Products](#)".

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 product 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 `timestamp_rounding` entry in the CM `pin.conf` file is set to `0`, the unit interval is calculated in seconds because the timestamp will not be rounded to midnight and the proration will begin from the time that is indicated by the timestamp. If `timestamp_rounding` is set to `1`, the unit interval will be calculated in days because the timestamp will be rounded to midnight. For more information, see ["About Time-Stamp Rounding."](#)

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: 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: 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. See ["Setting the 31-Day Billing Feature"](#).) 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. See ["Examples Using the 29th, 30th, and 31st for Billing Day of Month"](#).

Examples of Proration

Following are some examples of proration. In each of these examples, the unit interval is calculated differently. For additional examples, see ["Calculating Prorated Cycle Fees and Refunds for Customized Products"](#).

Example 1: Use_number_of_days_in_month Is Not Set or Set to 0

In this example, illustrated in Figure 4-1, 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

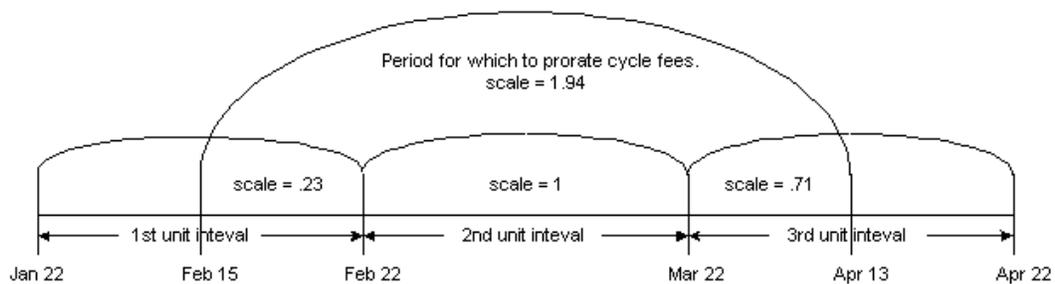
Starting from the billing time (April 22), BRM calculates the last unit interval by moving to the left one month at a time (on the time axis). Therefore, the last unit interval will be Mar 22–Apr 22. Similarly, it will continue to calculate unit intervals until the start time of the first unit interval (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 4-1 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.

$$\text{Scale} = 7/31 = 0.23$$

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.

$$\text{Scale} = 22/31 = 0.71$$

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: Use_number_of_days_in_month Is Set to 1

In this example, illustrated in Figure 4–2, 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:

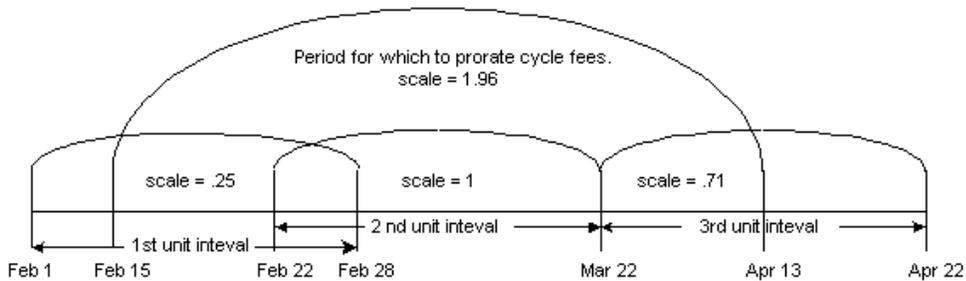
Mar 22–Apr 22 (last unit interval)

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 4–2 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.

$$\text{Scale} = 7/28 = 0.25$$

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.

$$\text{Scale} = 22/31 = 0.71$$

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 4-3](#), 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)

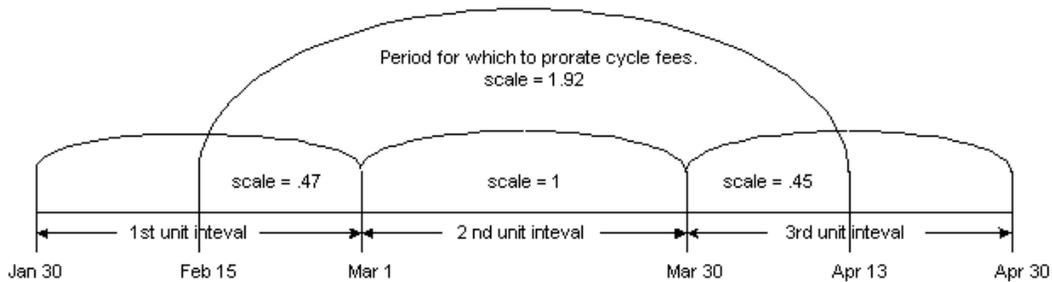
Mar 1–Mar 30 (second unit interval)

Note: Because February does not have 30 days, and because the billing option is set to forward, BRM calculates this unit interval as beginning on the first day of the next month, which is March 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 4–3 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:

$$\text{Scale} = 14/30 = 0.47$$

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:

$$\text{Scale} = 14/31 = .45$$

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:

$$\text{Scale for the period Feb 15–Apr 13} = .47 + 1.0 + .45 = 1.92$$

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 4–4](#), 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)

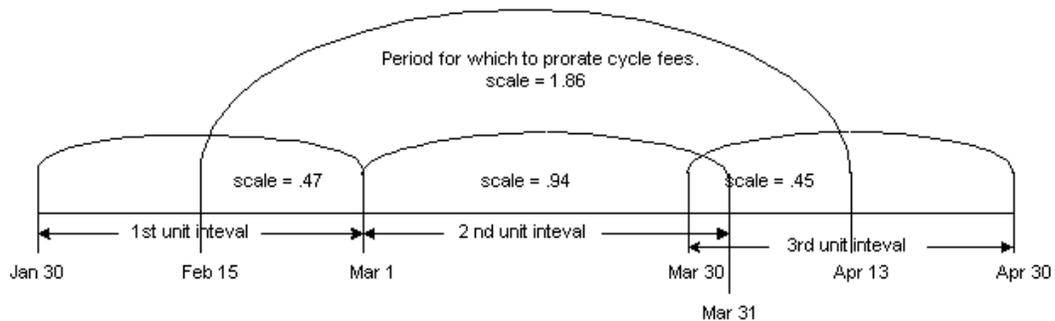
Mar 1–Mar 31 (second unit interval)

Note: Because February does not have 30 days, and because the billing option is set to forward, BRM calculates this unit interval as beginning on the first of the next month, which is March 1. 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 4-4 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:

$$\text{Scale} = 14/30 = 0.47$$

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.

$$\text{Scale} = 29/31 = .94$$

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:

$$\text{Scale} = 14/31 = .45$$

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:

$$\text{Scale for the period Feb 15–Apr 13} = .47 + .94 + .45 = 1.86$$

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 4-5](#), 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 will be 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)

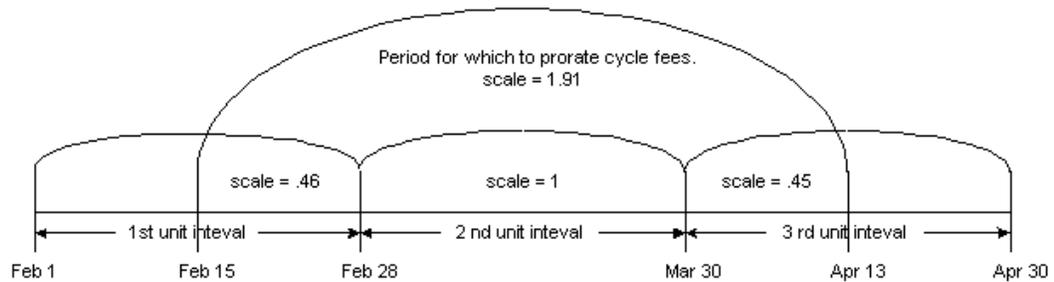
Feb 28–Mar 30 (second unit interval)

Note: Because February does not have 30 days, and because 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 4-5 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:

$$\text{Scale} = 13/28 = 0.46$$

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:

$$\text{Scale} = 14/31 = .45$$

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:

$$\text{Scale for the period Feb 15–Apr 13} = .46 + 1.0 + .45 = 1.91$$

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 4-6, 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 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)

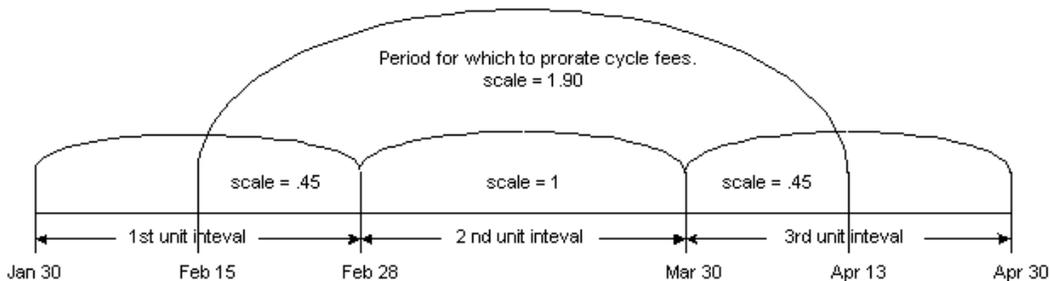
Feb 28–Mar 30 (second unit interval)

Note: Because February does not have 30 days, and because 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 4–6 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:

$$\text{Scale} = 13/29 = 0.45$$

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.

$$\text{Scale} = 14/31 = .45$$

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

Enabling BRM to Apply Proration Rules

By default, BRM does not apply proration rules to product purchases or cancellations that align with the start of the product cycle. This allows you to apply full cycle fees or refunds for such purchases or cancellations irrespective of the proration rules set. For example, if start date of the product cycle is July 14 and the product is purchased on the same day, proration rules are not applied and full cycle fees are applied for that product. However, if the product is purchased on July 20, proration rules are applied and the cycle fee is prorated.

However, you can configure BRM to apply proration rules for all product purchases or cancellations irrespective of whether they align with the start of the product cycle or not. For example, you can prevent a refund for a fee even if the full cycle is being refunded by applying proration rules. 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 the *BRM_home/sys/data/config* directory, where *BRM_home* is the directory in which you installed the BRM software.
2. Run the following command, which creates 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 the XML file named `bus_params_subscription.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_subscription.xml.out` file in a text editor.
4. Search the file for the following line:

```
<ApplyProrationRules>disabled</ApplyProrationRules>
```

By default, the **ApplyProrationRules** parameter is disabled.

5. Change **disabled** to **enabled**.
6. Save this file as `bus_params_subscription.xml`.
7. Go to the *BRM_home/sys/data/config* directory, which includes support files used by the `pin_bus_params` utility.
8. Run the following command, which loads this change into the **/config/business_params** object:

```
pin_bus_params PathToWorkingDirectory/bus_params_subscription.xml
```

where *PathToWorkingDirectory* is the directory in which **bus_params_subscription.xml** resides.

9. Read the object with the **testnap** utility or Object Browser to verify that all fields are correct.

For more information on reading objects by using Object Browser, see *BRM Managing Customers*. For instructions on using the **testnap** utility, see *BRM Developer's Guide*.

Proration for Special Cases

BRM includes proration settings that address customers with cycle arrears fees who purchase, inactivate, and reactivate products 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
```

Important: You must restart the CM after you change this parameter. See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

When you set up proration in a product's rate plan, you choose a proration setting for when the product is purchased and for when the product is canceled:

- If you want the proration setting for purchase time used to rate periods in which a customer both purchases and inactivates a product, set the **cycle_arrear_proration** parameter to **0**.
- If you want the cancellation proration setting used to rate periods in which a customer cancels a product, set the **cycle_arrear_proration** parameter to **1**.

Special Cases

Some special proration cases may result in unexpected billing results:

- A customer purchases a product with a cycle arrears fee, inactivates the product, and then reactivates the product all in the same accounting cycle. The product is set to prorate on purchase and 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 customer's product was active during the first cycle, the full amount for the cycle is charged.

- A customer purchases a product with a cycle arrears fee, inactivates the product, and then reactivates the product all in the same accounting cycle. The product is set to no charge on purchase and 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 product was active, except for the time of the last active period in the cycle.

For example, if the customer purchases a product, inactivates it after 5 days, reactivates it later for another period of 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), or 10 days of use.

- A customer purchases a product with a cycle arrears fee, inactivates the product, and then reactivates the product all in the same accounting cycle. The product is set to charge full on purchase and 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 product was active.

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

In this case, the customer is charged a prorated amount equal to the time the product was 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 a product, inactivates the product, and reactivates the product in the same period. The product 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 when the product was 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 long 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 proration settings **Charge for the full cycle**, **Don't charge this cycle**, and **Charge based on usage** for the inactivation/cancellation of a product when you are using cycle forward billing refers to refunding. **Charge for the full cycle** means you refund nothing on inactivation/cancellation, **Don't charge this cycle** means you refund all money charged for the cycle, and **Charge based on usage** means you refund the unused portion of the monthly usage fee.
-
-

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 product is purchased.
- Because most special cases occur when you use cycle arrears fees, consider using cycle forward fees instead.

Calculating Prorated Cycle Fees and Refunds for Customized Products

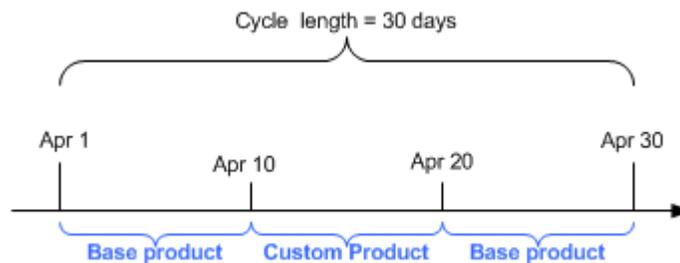
When a customized product is valid during all or part of a cycle, BRM uses it to calculate prorated cycle fees and refunds. See "About Customized Product Validity" in *BRM Managing Customers*.

- When a customized product is valid during the entire cycle, proration occurs just as with any other product.
- When a base product is valid during part of the cycle and one or more customized products is valid during part of the cycle, BRM calculates the fee or refund based on the proportion of time that each product is valid during the cycle or refund period.

Example 4: Applying a Cycle Forward Fee with a Customized Product

In this example, illustrated in [Figure 4-7](#), Product A has a monthly cycle forward fee of \$12. Its cycle fees are effective from January 1 to December 31. A CSR customizes the product to reduce the cycle forward fee by 50% for the period Apr 10–Apr 20.

Figure 4-7 Example 4



BRM calculates the total cycle fee for the month of April by calculating the fees separately for the portions that are covered by the base product (Product A's standard fee) and the customized product (the 50% reduction).

For each product, BRM calculates the scale by comparing the number of days in the cycle to the number of days in the cycle that the product is valid. For April, the number of days in the cycle is 30.

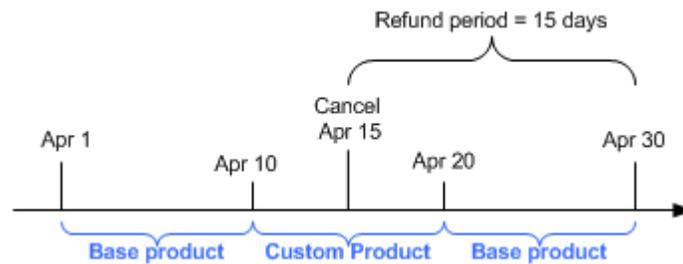
- The base product is valid from Apr 1–Apr 10 and Apr 21–Apr 30, a total of 20 days. Its scale is therefore 20/30. The base product's cycle fee is \$12, so its share of the total cycle fee is $20/30 * \$12 = \8 .
- The customized product is valid from Apr 11–Apr 20, a total of 10 days. Its scale is 10/30. The customized product's cycle fee is \$10, so its share of the cycle fee is $10/30 * \$6 = \2 .

The total cycle fee is the sum of the two individually calculated fees, or \$10.

Example 5: Refunding a Prorated Cycle Forward Fee with a Customized Product

In this example, illustrated in [Figure 4-8](#), the customer cancels the account that owns Product A and its customization on April 15. Cycle fee proration is set to **Calculate the charge based on the amount used**, so BRM must calculate a prorated refund that includes both the base product and the customized product.

Figure 4–8 Example 5



For this calculation, BRM begins with the scale that was originally applied to the cycle fee for each product. For the base product, this is 20/30; and for the customized product, 10/30.

BRM then calculates the proportion of the days that were used to calculate the scale that fall within the refund period. In this case, the refund period is Apr 15–Apr 30.

- For the base product, 10 days (Apr 21–Apr 30) of the period that was used to calculate the original scale fall within the refund period. So the proportion is 10/20.
- For the customized product, 5 days (Apr 11–Apr 15) of the period that was used to calculate the original scale fall within the refund period. So the proportion is 5/10.

To calculate the refund, BRM multiplies the rate by product of the original scale and the proportion of the scale that falls within the refund period.

- For the base product, the refund is $\$12 * 20/30 * 10/30 = \4 .
- For the customized product, the refund is $\$6 * 10/30 * 5/10 = \1 .

The total refund is $\$4 + \$1 = \$5$.

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.

Important: 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.

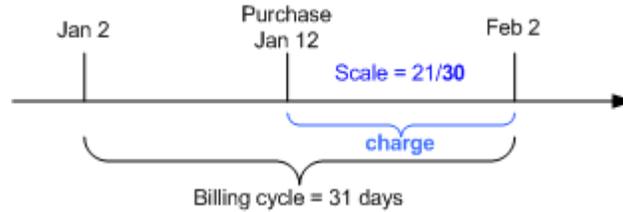
Examples of 30-Day-Based Proration

Based on your proration setting, BRM calculates the prorated cycle amount differently, which may yield different results.

In the examples below, illustrated in [Figure 4-9](#), [Figure 4-10](#), [Figure 4-11](#), and [Figure 4-12](#), February has 28 days and billing occurs on the second day of every month. The monthly cycle forward fee is \$30 and `timestamp_rounding` is set to 1 days in the CM `pin.conf` file.

Example 6: Prorated Purchase Fee with 31-day Billing Cycle

Figure 4-9 Example 6



In this example, BRM calculates the prorated cycle fee as follows:

Using 30-Day proration

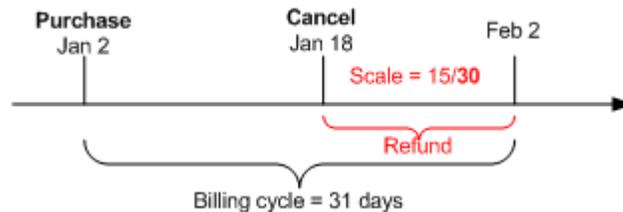
1. Calculates the scale using 30 days as the base:
 $scale = 21/30 = .70$
 where 21 is the number of days from midnight January 12 to midnight February 2.
2. Calculates the prorated cycle fee:
 $cycle\ fee\ amount * scale = \$30.00 * .70 = \$21.00$

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 4-10 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:

$$\text{scale} = 15/30 = .50$$

where 15 is the number of days from midnight January 18 to midnight February 2.

2. Calculates the refund amount:

$$\text{cycle fee amount} * \text{scale} = \$30.00 * .50 = \$15.00$$

Using 31 Days in the Billing Cycle

1. Calculates the scale using 31 days as the base:

$$\text{scale} = 15/31 = .48$$

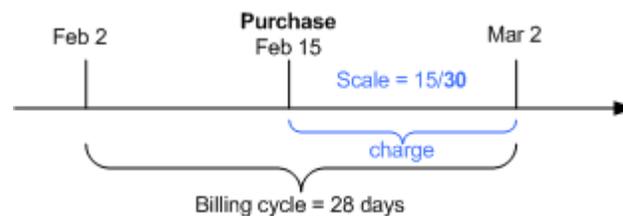
where 15 is the number of days from midnight January 18 to midnight February 2.

2. Calculates the refund amount:

$$\text{cycle fee amount} * \text{scale} = \$30.00 * .48 = \$14.52$$

Example 8: Prorated Purchase Fee with 28-Day Billing Cycle

Figure 4-11 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:

$$\text{scale} = 15/30 = .50$$

where 15 is the number of days from midnight February 15 to midnight March 2.

2. Calculates the prorated cycle fee:

$$\text{cycle fee amount} * \text{scale} = \$30.00 * .50 = \$15.00$$

Using 28 Days in the Billing Cycle

1. Calculates the scale using 28 days as the base:

$$\text{scale} = 15/28 = .54$$

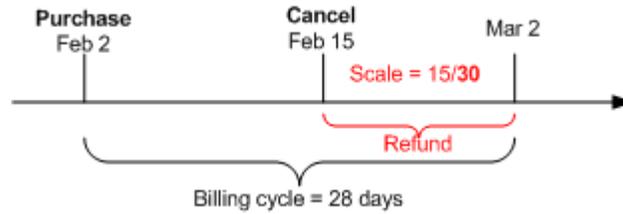
where 15 is the number of days from midnight February 15 to midnight March 2.

2. Calculates the prorated cycle fee:

$$\text{cycle fee amount} * \text{scale} = \$30.00 * .54 = \$16.07$$

Example 9: Prorated Cancel Fee with 28-Day Billing Cycle

Figure 4–12 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:
 $cycle\ fee\ amount * scale = \$30.00 * .54 = \$16.07$

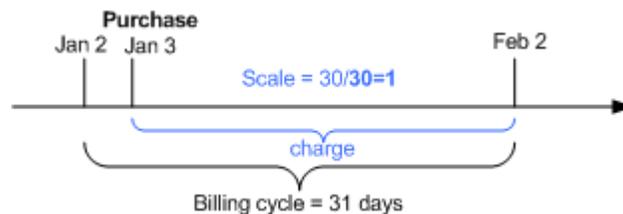
Special Cases

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 4–13, the billing cycle is 31 days. The customer purchases the product on January 3, so the scale is 1. The customer pays the full \$30 cycle fee even though the service was available for one day less.

Figure 4–13 Example 10



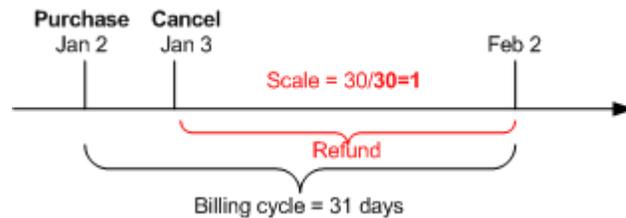
The prorated cycle fee:

$$cycle\ fee * scale = \$30.00 * 1 = \$30.$$

Example 11: Full Cancel Fee Refunded When Service Has Been Used for 1 Day

In this example, illustrated in Figure 4-14, the billing cycle is 31 days and the monthly service fee is \$30. The customer purchases the product on January 1 and cancels the product on January 2. The scale is 1. The customer gets a full \$30 refund even though customer owned the product for one day.

Figure 4-14 Example 11



The refund amount is:

$$\text{cycle fee} * \text{scale} = \$30.00 * 1 = \$30.$$

Important: 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 Products Whose Validity Ends in First Cycle

When prorating a cycle fee for a product whose cycle validity period ends during the same billing cycle in which the product was 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 (product cycle validity end date through billing cycle end date).

To use two events, enable the **CreateTwoEventsInFirstCycle** business parameter in the subscription instance of the `/config/business_params` object with the `pin_bus_param` utility.

For example, by default, if the product is purchased on January 1 and the product's 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. When the **CreateTwoEventsInFirstCycle** business parameter is enabled, 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 use two events to prorate a cycle fee for a product whose cycle validity period ends in the first cycle:

1. Go to the `BRM_Home/sys/data/config` directory, where `BRM_Home` is the directory in which you installed BRM.
2. Run the following command, which creates 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 the XML file named `bus_params_subscription.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_subscription.xml.out` file in a text editor.
4. Search the file for the following line:

```
<CreateTwoEventsInFirstCycle>disabled</CreateTwoEventsInFirstCycle>
```

By default, the **CreateTwoEventsInFirstCycle** parameter is disabled.

5. Change **disabled** to **enabled**.
6. Save this file as `bus_params_subscription.xml`.
7. Go to the `BRM_Home/sys/data/config` directory, which includes support files used by the `pin_bus_params` utility.
8. Run the following command, which loads this change into the `/config/business_params` object:

```
pin_bus_params PathToWorkingDirectory/bus_params_subscription.xml
```

where `PathToWorkingDirectory` is the directory in which `bus_params_subscription.xml` resides.

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

Note: To run this command from a different directory, see "pin_bus_params" in *BRM Developer's Guide*.

9. Read the object with the **testnap** utility or Object Browser to verify that all fields are correct.

For more information, see the descriptions about using **testnap** and about reading objects by using Object Browser in *BRM Developer's Guide*.

10. Stop and restart the CM. For more information, see the description of starting and stopping the BRM System in *BRM System Administrator's Guide*.

Prorating Cycle Fees after a Discount Purchase or Cancellation

When off-cycle discounts are purchased or canceled, and the discount's usage map is configured to support any of the product's cycle forward event types, the cycle fees are discounted for the duration of the cycle for which the discounts are valid.

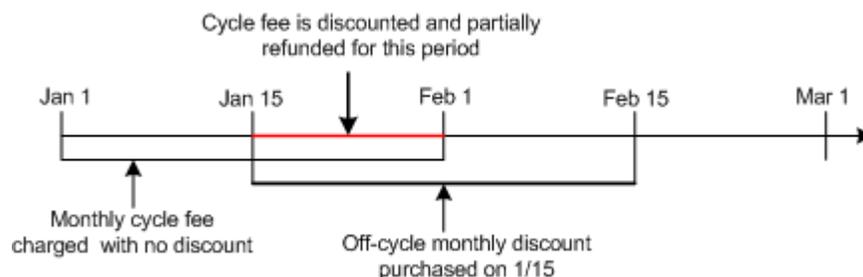
Examples of Cycle Fee Proration

These examples show how cycle fee proration works.

Example 12: Cycle Fee Is Refunded after a Discount Purchase

In this example, illustrated in [Figure 4-15](#), 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.

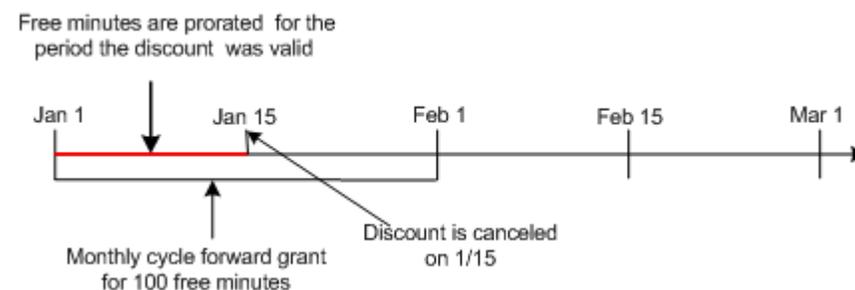
Figure 4-15 Example 12



Example 13: Free Minutes Are Prorated after a Discount Cancellation

In this example, illustrated in [Figure 4-16](#), a cycle forward monthly event grants 100 free 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 free minutes are prorated between January 1 and January 15. The account is granted 50 free minutes and is charged \$0.10 per minute for the remaining 150 minutes.

Figure 4-16 Example 13



Example 14: Canceled Discount Proration Is Not Taken into Account When Product Is Canceled

If you cancel a discount for a cycle fee, and then you cancel the product 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 product 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, and \$3 is charged back. Later, if the product 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.

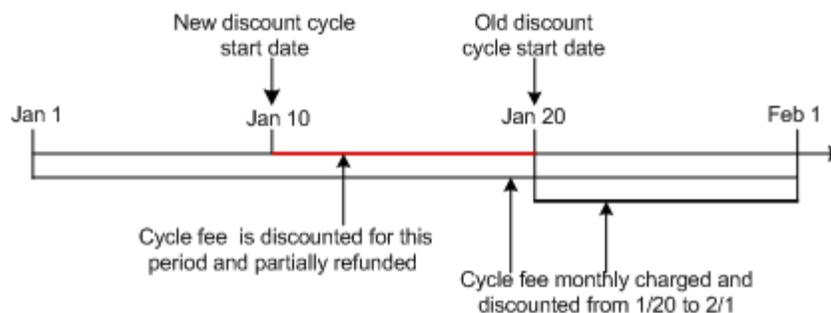
Examples of Cycle Fee Proration

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 4-17](#), 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.

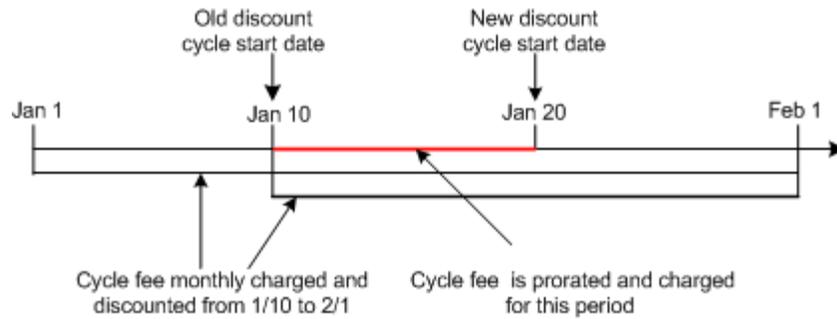
Figure 4-17 Example 15



Example 16: Cycle Fee Is Charged When a Discount's Cycle Start Date Is Changed

In this example, illustrated in [Figure 4-18](#), 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.

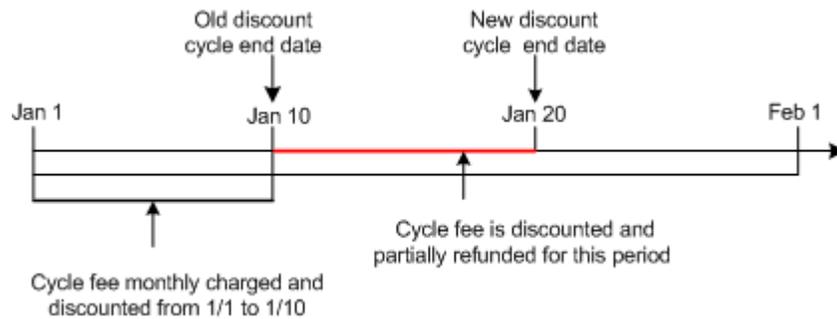
Figure 4–18 Example 16



Example 17: Cycle Fee Is Refunded When Discount's Cycle End Date Is Changed

In this example, illustrated in Figure 4–19, 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.

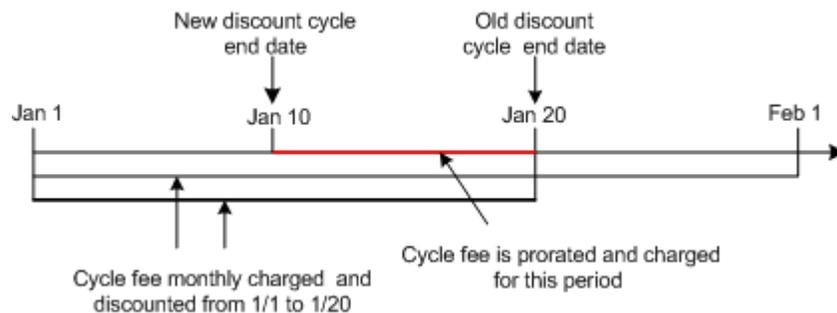
Figure 4–19 Example 17



Example 18: Cycle Fee Is Charged When Discount's Cycle End Date Is Changed

In this example, illustrated in Figure 4–20, 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 4–20 Example 18



Managing Bill Units with Your Custom Application

This chapter describes how to create and manage bill units (**/billinfo** objects) in Oracle Communications Billing and Revenue Management (BRM) by using your custom application.

For information about bill units, see "About Bill Units" in *BRM Managing Accounts Receivable*.

Opcodes Used with Bill Units

Use the following opcodes to manage bill units:

- To create a **/billinfo** object, use PCM_OP_CUST_CREATE_BILLINFO.
See "[Creating /billinfo Objects](#)".
- To update a **/billinfo** object, use PCM_OP_CUST_SET_BILLINFO.
See "[Updating /billinfo Objects](#)".
- To customize how **/billinfo** objects are created, use PCM_OP_CUST_POL_PREP_BILLINFO to prepare **/billinfo** data and PCM_OP_CUST_POL_VALID_BILLINFO to validate **/billinfo** data.
See "[Preparing /billinfo Data](#)" and "[Validating /billinfo Data](#)".
- To delete a **/billinfo** object, use PCM_OP_CUST_DELETE_BILLINFO.
See "[Deleting /billinfo Objects](#)".
- To postpone finalizing a bill until the end of a future billing cycle, suppress billing for the **/billinfo** object.
See "[About Bill Suppression](#)".
- To suspend billing of a **/billinfo** object in a closed account, use PCM_OP_BILL_POL_POST_BILLING.

When you reactivate a closed account, BRM automatically resumes billing for that account.

See "[Suspending and Resuming Billing of Closed Accounts](#)".

Creating /billinfo Objects

Use PCM_OP_CUST_CREATE_BILLINFO to create **/billinfo** objects. This opcode creates the **/billinfo** object by using the information provided in the input flist.

You can set each **/billinfo** object's billing cycle fields, such as the billing type, the billing frequency, and the billing DOM, by passing information in the opcode's input flist fields.

If the object is successfully created, the output flist contains:

- The POID of the **/account** storable object to which the **/billinfo** belongs.
- PIN_FLD_BILLINFO array that specifies the billing information that is created.

For information about creating **/billinfo** hierarchies and sponsorship, see "[Creating /billinfo Object Hierarchy and Sponsorship](#)".

Associating Account Balances with /billinfo Objects

After creating a new **/billinfo** object, you must associate it with the account balances for which the bill is created. To do this, the account must have multiple balance groups so that different balances can be linked to different bills. You create multiple balance groups when you create your plans in Pricing Center. See the discussion about tracking service-level resources in the Pricing Center online help.

To associate account balances with a new bill, see the discussion about moving a balance group to a new **/billinfo** object in the Pricing Center online help.

Updating /billinfo Objects

Use PCM_OP_CUST_SET_BILLINFO to update billing information in existing **/billinfo** objects. This opcode calls policy opcodes to permit customization and perform validations. Set the value of the PIN_FLD_POID field in the PIN_FLD_BILLINFO array to -1, which causes PCM_OP_CUST_SET_BILLINFO to call PCM_OP_CUST_CREATE_BILLINFO before calling the policy opcodes.

PCM_OP_CUST_SET_BILLINFO updates an existing PIN_FLD_BILLINFO array associated with a specified account by setting new values for the array fields as specified in the input flist. Any PIN_FLD_BILLINFO array fields not included in the input flist are left unchanged.

Use the following opcodes to customize PCM_OP_CUST_SET_BILLINFO functionality:

- Use PCM_OP_CUST_POL_PREP_BILLINFO to prepare **/billinfo** data. See "[Preparing /billinfo Data](#)".
- Use PCM_OP_CUST_POL_VALID_BILLINFO to validate **/billinfo** data. See "[Validating /billinfo Data](#)".

Both of these opcodes are called by PCM_OP_CUST_SET_BILLINFO.

You can set each **/billinfo** object's billing cycle fields, such as the billing type, the billing frequency, and the billing DOM, by passing information in the opcode's input flist fields.

For information about creating **/billinfo** hierarchies and sponsorship, see "[Creating /billinfo Object Hierarchy and Sponsorship](#)".

Preparing /billinfo Data

The PCM_OP_CUST_POL_PREP_BILLINFO policy opcode processes the account billing fields in the **/billinfo** storable object during customer registration or while updating billing information to prepare for validation. For information about the PREP opcodes, see "About the PREP and VALID Opcodes" in *BRM Developer's Guide*.

The policy opcode's main responsibility includes:

- Assigning default values for the billing cycle length, primary currency, and accounting type, if they are not passed in the input flist. See "[Assigning Default Billing Information](#)".
- Determining and assigning the billing DOM. See "[Assigning DOMs to /billinfo Objects](#)".

The policy opcode prepares the following billing information:

- Account POID
- Event ending timestamp
- Bill unit POID
- Payment method
- Bill unit name
- New or changed billing DOM
- Old billing DOM
- Accounting type: open item accounting or balance forward accounting
- Next billing date
- Billing frequency (based on the number of accounting cycles)
- Parent bill unit
- Currency used
- Secondary currency used
- Billing segment ID

The PCM_OP_CUST_POL_PREP_BILLINFO policy opcode provides a mechanism for taking the information passed to PCM_OP_CUST_SET_BILLINFO and processing the fields before their validation by the PCM_OP_CUST_POL_VALID_BILLINFO policy opcode. See "[Validating /billinfo Data](#)".

Processing includes adding any missing fields whose values are derived or generated by this opcode, and forcing fields to predefined values independent of what you specified. You specify fields on the input flist, and this opcode returns the processed version of the data on the output flist. Validity of the field values is checked by PCM_OP_CUST_POL_VALID_BILLINFO.

If PCM_OP_CUST_POL_PREP_BILLINFO cannot derive all of the necessary fields because the values you specified are incorrect, no error is returned. Instead, the derived fields are returned on the output flist with a default value, and the PCM_OP_CUST_POL_VALID_BILLINFO policy opcode is called to detect the incorrect customer data and return the validation error to the calling application. This enables the calling application to get the details of the validation error instead of receiving an incorrect **ebuf** error. If PCM_OP_CUST_POL_PREP_BILLINFO cannot generate a necessary field or encounters other internal problems, it returns an **ebuf** error.

For more information, see "About the PREP and VALID Opcodes" in *BRM Developer's Guide*.

Assigning Default Billing Information

The PCM_OP_CUST_POL_PREP_BILLINFO policy opcode prepares the billing information for on-line registration by implementing default values as shown below:

- If not already specified in the in the input flist, the PIN_FLD_BILL_WHEN field is set to the value specified in the CM **pin.conf** file's **bill_when** entry. If no value is found in the **pin.conf** file, it is defaulted to **1**. See "[Setting the Default Billing-Cycle Length](#)".
- If not already specified in the input flist, the PIN_FLD_CURRENCY field is set to the value specified in the CM **pin.conf** file's **currency** entry. If no value is specified, the field is set to the currency associated with the system account. See "Setting the Default Account Currency" in *BRM Managing Customers*.
- If not already specified in the input flist, the PIN_FLD_ACTG_TYPE field is set to the value specified in the CM **pin.conf** file's **actg_type** entry. If no value is found in the **pin.conf** file, the field is set to **2** (balance-forward accounting). See "[Setting the Default Accounting Type](#)".
- If the primary currency is EURO, the PIN_FLD_CURRENCY_SECONDARY field is set to the default secondary currency.

Assigning DOMs to /billinfo Objects

PCM_OP_CUST_POL_PREP_BILLINFO assigns a DOM to a **/billinfo** object based on the following order of priority:

1. **The DOM assigned to the billing segment.** The policy opcode assigns the DOM set for the billing segment in the **/config/billing_segment** object. For more information, see "[Assigning DOM Based on the Billing Segment](#)".

Note: To customize how the policy opcode assigns the DOM according to the billing segment, see "[Customizing the DOM Assignment Process](#)".

2. **The DOM used by first /billinfo object in the account.** If a DOM is not assigned to the billing segment, the policy opcode assigns the DOM to that of the first **/billinfo** object in the account.
3. **Default setting in the Connection Manager pin.conf file.** If a DOM is not assigned to the billing segment nor is available from another **/billinfo** object, the DOM is set to the value assigned in the **actg_dom** entry in the CM configuration file (*BRM_Home/sys/cm/pin.conf*, where *BRM_Home* is the directory where you installed BRM components). To set the default value, see "[Setting the Default Accounting Day of Month \(DOM\)](#)".
4. **The current date.** If a DOM is not available from the billing segment, other **/billinfo** objects, or the CM **pin.conf** file, the policy opcode assigns the DOM to the current date that is passed in the input flist's PIN_FLD_END_T field.

For example, if an account was created with two bill units, **BU1** that has an assigned DOM and **BU2** that does not have a DOM assigned, the policy opcode would assign a DOM to **BU2** as follows:

- If **BU2** is the second **/billinfo** array element in the **/account** object, the policy opcode assigns the DOM to that of the first **/billinfo** array element.
- If **BU2** is the first **/billinfo** array element in the **/account** object, the policy opcode assigns the DOM to the value set in the CM **pin.conf** file. If a value is not set in the CM **pin.conf** file, the DOM is set to the current date that is passed in the input flist's PIN_FLD_END_T field.

Assigning DOM Based on the Billing Segment

When a `/config/billing_segment` object exists in the Connection Manager (CM) cache and contains an array of billing segments, the `PCM_OP_CUST_POL_PREP_BILLINFO` policy opcode performs the following tasks:

- If the input `PIN_FLD_BILLING_SEGMENT` field *does not* contain a value or contains `0`, `PCM_OP_CUST_POL_PREP_BILLINFO` sets the output `PIN_FLD_BILLING_SEGMENT` field to `0`. This triggers BRM to use the default process rather than the bill cycle management process of assigning a billing DOM to the `/billinfo` object.
- If the input `PIN_FLD_BILLING_SEGMENT` field *does* contain a value other than `0` but the input `PIN_FLD_ACTG_FUTURE_DOM` field *does not* contain a value, `PCM_OP_CUST_POL_PREP_BILLINFO` uses the bill cycle management process of assigning a billing DOM to the `/billinfo` object:
 - By default, the opcode assigns the first open DOM in the specified billing segment, starting with the current DOM. See "[About Accounting DOM Status](#)".
 - Alternatively, the opcode can use a weighted average calculation to select a billing DOM. See "[Customizing the DOM Assignment Process](#)".

Note: If the input `PIN_FLD_ACTG_FUTURE_DOM` field *does* contain a value, that value becomes the billing DOM unless it is closed (see "[About Accounting DOM Status](#)"). If it is closed, `PCM_OP_CUST_POL_VALID_BILLINFO` returns a validation error.

The default DOM assignment process rather than the bill cycle management process is also used in these situations:

- The `/config/billing_segment` object is not in the CM cache.
- In the cached `/config/billing_segment` object, no DOMs are associated with the billing segment ID specified in the input `PIN_FLD_BILLING_SEGMENT` field.

Customizing the DOM Assignment Process

To increase the probability that the assigned billing DOM has the lightest billing load of all the open days in a billing segment, this policy opcode can optionally use a weighted average calculation to select a DOM. The calculation should factor in the total billing-run processing time of each open DOM. This information is stored in the `/config/billing_segment` object.

A sample weighted average calculation is included in the opcode file (`fm_cust_pol_prep_billinfo.c`).

To use the calculation, `PCM_OP_CUST_POL_PREP_BILLINFO` must call the `fm_cust_pol_prep_billinfo_get_dom_from_process_t_from_cache` function to assign a DOM to a `/billinfo` object. (By default, the opcode calls `fm_cust_pol_prep_billinfo_get_next_dom_from_cache`.)

The sample calculation works as follows:

Billing segment X, to which `/billinfo` Y belongs, has the following open DOMs and associated billing-run processing times as shown in [Table 5-1](#):

Table 5-1 Sample DOMs

DOM/Process	Value	Value	Value	Value	Value
Open DOMs	5	11	16	20	75
PIN_FLD_TOTAL_PROCESS_T (in seconds)	20000	30000	15000	45000	25000

Using the total number of seconds in two DOMs (172,800) as a constant, PCM_OP_CUST_POL_PREP_BILLINFO divides the constant by each DOM's total billing-run processing time as shown in [Table 5-2](#):

Table 5-2 Sample Bill-Run Processing Times

DOM/Process	Value	Value	Value	Value	Value
Open DOMs	5	11	16	20	27
PIN_FLD_TOTAL_PROCESS_T (in seconds)	20000	30000	15000	45000	25000
172,800/ PIN_FLD_TOTAL_PROCESS_T (results are rounded down to nearest whole number)	8	5	11	3	6

PCM_OP_CUST_POL_PREP_BILLINFO then adds the results and uses their sum as a seed value to generate a random number. In this case, the sum is 33. Assume the random number is 21.

PCM_OP_CUST_POL_PREP_BILLINFO subtracts the random number from the result in the first column of the table. If the remainder is less than 0, the opcode assigns the column's DOM to the /billinfo object. If the remainder is greater than 0, the opcode subtracts the remainder from the result in the next column. It continues this process until it gets a remainder that is less than 0 as shown in [Table 5-3](#):

Table 5-3 Sample Values for DOM

DOM/Process	Value	Value	Value	Value	Value
Open DOMs	5	11	16	20	27
PIN_FLD_TOTAL_PROCESS_T (in seconds)	20000	30000	15000	45000	25000
172,800/ PIN_FLD_TOTAL_PROCESS_T (results are rounded down to nearest whole number)	8	5	11	3	6
First Quotient - Random Number	21 - 8 = 13	NA	NA	NA	NA
Is result less than 0? No	13 > 0	13 - 5 = 8	NA	NA	NA
Is result less than 0? No	NA	8 > 0	8 - 11 = -3	NA	NA
Is result less than 0? Yes	NA	NA	-3 < 0	NA	NA

Based on the final result of this example calculation, PCM_OP_CUST_POL_PREP_BILLINFO sets the PIN_FLD_ACTG_FUTURE_DOM in the output flist for /billinfo Y to 16.

Validating /billinfo Data

The PCM_OP_CUST_POL_VALID_BILLINFO policy opcode validates an account's billing information in the /billinfo storable object passed to it by the PCM_OP_CUST_POL_PREP_BILLINFO policy opcode during customer registration or administrative update.

The billing information can include:

- Payment method
- Parent /billinfo object
- Next bill time
- Currency used
- Billing frequency
- Accounting cycle duration
- Accounting type
- Billing segment ID

For more information, see "About the PREP and VALID Opcodes" in *BRM Developer's Guide*.

By default, PCM_OP_CUST_POL_VALID_BILLINFO validates the PIN_FLD_ACTG_TYPE, PIN_FLD_ACCOUNT_NO, and PIN_FLD_PAY_TYPE fields according to the criteria contained in the /config/fld_validate storable object. Invalid pay types result in an error.

Valid payment methods are listed in the *BRM_Home/include/pin_cust.h* file as BILL_TYPE. Include the *pin_cust.h* file in the applications that call PCM_OP_CUST_POL_VALID_BILLINFO.

All customizing should be done with the Field Validation Editor in Configuration Center. Users should not change the *fm_cust_pol_valid_billinfo.c* file for PCM_OP_CUST_POL_VALID_BILLINFO because it could corrupt the database. The Field Validation Editor is used to create and edit the /config/fld_validate storable object.

Validating Billing Segment Information

If the PIN_FLD_BILLING_SEGMENT field in the PCM_OP_CUST_POL_VALID_BILLINFO input flist contains a value other than 0, the opcode performs the following tasks:

- It verifies that the billing segment identified in the PIN_FLD_BILLING_SEGMENT field is in the /config/billing_segment object.
- If the billing segment is in the /config/billing_segment object, PCM_OP_CUST_POL_VALID_BILLINFO checks the status of the billing DOM in the input flist PIN_FLD_ACTG_FUTURE_DOM field for the specified billing segment. See "[About Accounting DOM Status](#)".

PCM_OP_CUST_POL_VALID_BILLINFO logs a validation error and returns a message to the user interface in these situations:

- The specified billing segment is not in the `/config/billing_segment` object.
- The status of the specified billing DOM is closed.

Suspending and Resuming Billing of Closed Accounts

To suspend and resume billing of a bill unit (`/billinfo` object) in a closed account, see the following:

- [Suspending Billing of Closed Accounts](#)
- [Resuming Billing When Closed Accounts Are Reactivated](#)

For an overview of bill suspension, see "[About Suspending Billing of Accounts and Bills](#)".

Note: To *suppress* billing, see "[About Bill Suppression](#)".

Suspending Billing of Closed Accounts

To suspend billing of a closed account, use the `PCM_OP_BILL_POL_POST_BILLING` policy opcode.

This policy opcode is called by `PCM_OP_BILL_MAKE_BILL`. It enables you to perform custom processing on a bill unit (`/billinfo` object) at the time of billing.

By default, this policy opcode suspends billing of a specified bill unit in a closed account that has the following characteristics:

- Zero balance due for all bill units (total balance due of all open and pending items).

For hierarchical accounts, the balance due amount includes the open and pending item totals of every subordinate bill unit in child accounts.

- No billable activity since the previous bill was generated.

If the bill unit has nonpaying child bill units, they are suspended, too.

For accounts with a multimonth billing cycle, the default implementation of this policy opcode suspends billing at the end of the last accounting cycle.

If delayed billing is enabled, the default implementation of this policy opcode suspends billing at the end of the billing delay interval.

To indicate billing is suspended, this policy opcode sets the `PIN_FLD_BILLING_STATUS` field in the `/billinfo` object to *inactive*.

This policy opcode returns the POID of the `/billinfo` object. For a complete list of all fields returned, see the output flist specification for this opcode.

Resuming Billing When Closed Accounts Are Reactivated

When an account's status is changed from *closed* to *active*, `PCM_OP_CUST_SET_STATUS` calls `PCM_OP_BILL_RESUME_BILLING` to resume suspended billing. Therefore, you do not need to call `PCM_OP_BILL_RESUME_BILLING` directly.

`PCM_OP_BILL_RESUME_BILLING` sets the `PIN_FLD_BILLING_STATUS` field in the account's bill units (`/billinfo` objects) to *active* and resets the billing information. For example, it resets the last billing date (`PIN_FLD_LAST_BILL_T`), next billing date (`PIN_FLD_NEXT_BILL_T`), last accounting cycle (`PIN_FLD_ACTG_LAST_T`), and next accounting cycle (`PIN_FLD_ACTG_NEXT_T`).

Note: The account status is not the same as the account's billing status. For example, an account's status can be *active* while its billing status is *inactive*.

PCM_OP_BILL_RESUME_BILLING does the following:

- Resets the account bill unit billing status to *active* (PIN_BILL_ACTIVE).

Note: A bill unit with suspended billing has the billing status set to *inactive* (PIN_BILL_INACTIVE).

- Resets the account bill unit's next billing date (PIN_FLD_NEXT_BILL_T) based on the account's billing day of month before billing was suspended. For example, if the billing day of month was the first of every month before billing was suspended, the billing day of month is set to the first of every month when billing is resumed.
- Resets the billing start date (PIN_FLD_START_T) in the /bill object to the current time (when billing is resumed).

Note: PCM_OP_BILL_RESUME_BILLING does not automatically resume suspended billing for subordinate bill units. To do that, you must reactivate the subordinate accounts that contain those bill units.

When you resume billing, billing is run on the next scheduled billing date. This is true even if billing would have been run when billing was suspended. That is, billing is not run immediately to accommodate billing that was missed during suspension.

Deleting /billinfo Objects

To delete /billinfo objects, write custom code that calls PCM_OP_CUST_DELETE_BILLINFO.

If the specified /billinfo storable object is a nonsubordinate parent /billinfo, this opcode automatically deletes any subordinate /billinfo objects that are associated with it.

Note: You cannot delete a /billinfo object that has pending payments.

If successful, the output flist contains:

- PIN_FLD_POID field set to the account POID of the /account storable object that is deleted.
- PIN_FLD_BILLINFO array that specifies the billing information that is deleted.

Creating /billinfo Object Hierarchy and Sponsorship

In account sponsorship and hierarchies, the /billinfo objects are the paying or nonpaying entities. When an account in a hierarchy or sponsorship has multiple /billinfo objects, you must designate specific /billinfo objects as parent or child, and

sponsor or sponsoree. For more information, see "About Hierarchical Bill Units" in *BRM Managing Accounts Receivable*.

Note: The /billinfo objects of accounts that have different parent accounts can form a /billinfo hierarchy. A /billinfo object in a child account can be subordinate to a /billinfo object in a different parent account. For more information, see "Creating Hierarchical Bill Units" in *BRM Managing Accounts Receivable*.

To create /billinfo hierarchy and sponsorship for accounts that have multiple /billinfo objects, write custom code that calls PCM_OP_CUST_SET_BILLINFO for each /billinfo object that will participate in the hierarchy or sponsorship. Specify the following relationship fields in the input flist:

- If the /billinfo is a nonpaying (subordinate) /billinfo in a hierarchy:
 - Set the PIN_FLD_PARENT_BILLINFO_OBJ field to specify paying (nonsubordinate) /billinfo in the parent account.
 - Set the PIN_FLD_AR_BILLINFO_OBJ field to specify the AR /billinfo object. This is the parent account's default /billinfo object.

Note: If the AR /billinfo is also the parent /billinfo, the parent and AR /billinfo fields specify the same /billinfo object.

- Set the value of the PIN_FLD_PAY_TYPE field to subordinate (PIN_PAY_TYPE_SUBORD).
- If the /billinfo is a paying /billinfo in a hierarchy, set the PIN_FLD_PARENT_FLAGS field.
- If the /billinfo has charges that are sponsored by another /billinfo object, set the PIN_FLD_SPONSOREE_FLAG field.
- If the /billinfo sponsors charges in another /billinfo object, set the PIN_FLD_SPONSOR_FLAG field.

Changing /billinfo Hierarchy and Sponsorship

When you change an account hierarchy and sponsorship by using Customer Center, all /billinfo objects that participate in that hierarchy or sponsorship are also changed. To add or remove individual /billinfo objects from a hierarchy or sponsorship, write custom code that calls PCM_OP_CUST_SET_BILLINFO and pass the new parent and child or sponsor and sponsoree fields on the input flist.

Billing Delays for Moved /billinfo Objects

When you move a /billinfo object from one parent to another, the future billing date (stored in the PIN_FLD_ACTG_FUTURE_T field) is synchronized with the parent /billinfo. However, the date that the current monthly cycle ends (stored in the PIN_FLD_ACTG_NEXT_T field) for the child /billinfo is not changed to match the parent /billinfo. Therefore, the first billing run following the move might be different for the parent and child /billinfo objects.

For example, a child /billinfo with a billing date of the 15th is moved to a new parent /billinfo that has a billing date of the 30th. If the child is moved on the 20th, it is not

billed on the following 30th when the parent is billed. Instead, it is billed on the 15th of the following month. Thereafter, all billing dates are synchronized.

Offering the Best Price to Your Customers

This chapter provides information on how to set up pricing in the Oracle Communications Billing and Revenue Management (BRM) system to calculate and offer the best price to your customers at billing time.

About Offering the Best Price to Your Customers

You can use the best pricing feature to calculate prices using different alternate deals at billing time and use the deal that offers the best price to the customer. You can either rerate the relevant events in the billing cycle by using the best deal or perform a one-time credit adjustment by applying the difference in charges between the base deal and the alternate best deal to the customer's balance.

For more information on base deals and alternate deals, see "About the Best Pricing Configuration" in *BRM Setting Up Pricing and Rating*.

For more information about how BRM calculates the best price in various situations, see "[How BRM Determines the Best Deal](#)".

Caution: Calculating the best price involves rerating account events multiple times, which affects the performance of the billing process.

To offer the best price to your customers, perform the following tasks:

- Configure BRM to use best pricing. See "[About Configuring BRM to Use Best Pricing](#)".
- Set up a base deal and a set of alternate deals by using Pricing Center. For information on creating the best deals, see the discussion about creating deals in the Pricing Center online help.
- (Optional) Specify a minimum charge for the alternate deal and specify conditions that must be met for the alternate deal to qualify for best pricing by using Pricing Center. See the discussion about defining a minimum charge for an alternate deal and defining conditions for an alternate deal for best pricing in the Pricing Center online help.
- Specify whether you want to rerate the events using the best deal or use the calc-only rating results and apply a credit to reduce the total charges in the bill by using Pricing Center. See the discussion about specifying how to apply savings from the best pricing calculation in the Pricing Center online help.
- Purchase a best pricing deal for an account by using Customer Center. See the discussion about purchasing a deal in the Customer Center help.

- (Optional) Determine the savings from the best pricing calculation by using the PCM_OP_SUBSCRIPTION_CALC_BEST_PRICING opcode or by using Customer Center.

For more information, see "[Calculating the Best Price by Using the Best Pricing Opcode](#)" and the discussion about viewing the savings from best pricing in the Customer Center online help.

- Bill your customers by running the **pin_bill_accts** utility with the options for best pricing or by using the **Bill Now** option in Customer Center.

For information on using the **Bill Now** option, see Customer Center Help.

How BRM Determines the Best Deal

BRM performs the best pricing calculation by using the PCM_OP_SUBSCRIPTION_CALC_BEST_PRICING opcode for each service or subscription service that has best pricing configured. For each account, it performs the calculation one at a time using all the best pricing deals available.

To apply the best price to a customer's balance, BRM follows these steps at billing time, after applying all the charges and discounts and before applying billing-time taxes:

1. Performs a calc-only rerating operation by using each alternate deal and compares the resulting charges with the base deal charges.
2. Determines the best deal to be the deal that offers the lowest charge.
3. If the base deal charges are the lowest, retains that balance impact.
4. If an alternate deal charges are the lowest, depending on how you configured best pricing, performs one of the following steps:
 - Rerates the events based on the best alternate deal and updates the balance.
 - Reduces the balance by the difference between the base deal balance impacts and the best deal balance impacts by applying a one-time credit adjustment.

If the CALC-ONLY flag is set, the balance is not updated.

Important: To perform a best pricing calculation in the middle of a billing cycle, perform the operation in calc-only mode.

Best pricing calculation considers only the alternate deals that are valid at the time of the calculation. It filters out the following types of alternate deals:

- Alternate deals with a minimum charge greater than the charges calculated using the base deal. For information on specifying a minimum charge, see the discussion about defining a minimum charge for an alternate deal in the Pricing Center online help.

The minimum charge is usually the sum of all cycle fees per accounting cycle. If a billing cycle spans multiple accounting cycles, the minimum charge is multiplied by the number of accounting cycles in the billing cycle.

Note: Best pricing does not support an accounting cycle longer than the billing cycle.

- Alternate deals that fail to meet all the conditions that you specify. For more information on specifying conditions for alternate deals, see the discussion about defining conditions for an alternate deal for best pricing in the Pricing Center online help.

Calculating the Best Price by Using the Best Pricing Opcode

To calculate the best price, call `PCM_OP_SUBSCRIPTION_CALC_BEST_PRICING` during billing or trial billing.

You call this opcode to calculate the best price after applying all the charges and discounts and before applying billing-time taxes. This opcode calculates the best price for each service instance or subscription service with the best pricing configuration.

If the `CALC-ONLY` flag is set, this opcode does not apply the balance impacts from the best pricing calculation.

Important: If you calculate the best price in the middle of a billing cycle, ensure that you perform the operation in `calc-only` mode. Otherwise, the balance impacts are committed to the database.

This opcode performs the following tasks to calculate and apply the best price:

1. Takes as input an account or an account and scope, which can include billing information, service, and deal.
2. If best pricing is configured, retrieves the list of deals, services, and balance groups from the `/billinfo` or `/service` objects. Otherwise, returns without doing anything.

Note: If only the account object is passed, this opcode retrieves all the account-level and service-level deals for the account. If the account billing information is passed, it retrieves all the services in the billing information.

3. Finds the best pricing configuration for each account, service, and subscription service.
4. Prepares the services list:
 - a. If the scope is deal level, this opcode finds the service with which the deal is associated and determines that it is not a subscription service or discount sharing group (DSG).
 - b. If the scope is service level, this opcode checks if the service is a parent in a DSG or an owner in an ordered balance group (OBG). If the service is a parent or owner, it includes all the children or the members in the service list. If the service is a child or member, it includes only that service.
 - c. If the scope is billinfo level, this opcode finds all services for this billinfo. If the service is a parent or owner, it includes all the children or the members in the service list. If the service is a child or member, it includes only that service.
5. Calls the `PCM_OP_BAL_GET_BALANCES` opcode to get a snapshot of the balances to compare with alternate deal charges.
6. Calls the `PCM_OP_BAL_RERATE_REBILL` opcode with the `CALC_ONLY` flag to rerate events using alternate deals.

7. Performs a best pricing calculation for the services by comparing the base deal charges with each alternate deal charge.

Note: Alternate deals must meet the specified conditions, have a minimum charge lower than the base deal charge, and be within the validity period to qualify for the best pricing analysis.

8. If this opcode is called at the end of the billing cycle without the CALC_ONLY flag, performs one of the following actions:
 - If full rerating is required, calls the PCM_OP_BAL_RERATE_REBILL opcode without the CALC_ONLY flag.
 - If a one-time balance adjustment is required, calls the PCM_OP_ACT_USAGE opcode with the best deal charges and the base deal charges in the input flist to record the event.

About Calculating the Best Deal When Alternate Deal Has a Best Pricing Configuration

If the best pricing configuration includes an alternate deal that has a best pricing configuration, the best pricing calculation includes only the base deal of that alternate deal.

For example, suppose best pricing deal A contains an alternate deal, Alt1, and Alt1 itself is a best pricing deal. When calculating the best price, the base deal A is compared only to the base deal for Alt1 and not the alternate deals of Alt1 to avoid multilevel recursive best pricing calculation.

About Finding the Best Deal in the Middle of a Billing Cycle

If best pricing is configured, a best pricing calculation is automatically performed at the end of the billing cycle during billing. However, you can find the best deal in the middle of a cycle by performing the best pricing calculation in calc-only mode for all the relevant services without applying the charges. You can find the best deal in the middle of a billing cycle without applying charges by using these methods:

- From Customer Center. The result of the calculation will be displayed in the Best Pricing dialog box. See Customer Center Help.
- By calling PCM_OP_SUBSCRIPTION_CALC_BEST_PRICING from your client application with the CALC-ONLY flag. The results of the calculation will be returned in an flist. See ["Calculating the Best Price by Using the Best Pricing Opcode"](#).

Important: When you calculate the best price in the middle of a billing cycle, cycle arrears events are not generated and are not included in the calculation.

About Rerating Events to Apply the Best Price

To rerate events to apply the best price, call PCM_OP_SUBSCRIPTION_CALC_BEST_PRICING without the CALC-ONLY flag. PCM_OP_SUBSCRIPTION_CALC_BEST_PRICING calls the PCM_OP_SUBSCRIPTION_RERATE_REBILL opcode, which performs the following functions:

- Backs out events for the specified services that occurred in the specified rerating period.
- Rerates them in chronological order.
- Uses the best alternate deal instead of the base deal for the account.
- For events that occurred outside the validity period for the alternate deal, uses the base deal for the account.

When rerating a discount sharing group account, this opcode rerates both owner and member account events using discounts from the best deal. It uses only the products owned by the owner and member accounts to rate events.

About Rerating Events for a Prior Cycle for Which the Best Deal Was Applied

While rerating a prior billing cycle, if a best pricing configuration is found, BRM performs a best pricing calculation for that complete cycle. Best pricing calculation backs out and rerates only events that occurred during the billing cycle.

Adjusting the Account Balance to Apply the Best Deal

If you select a one-time credit adjustment to apply savings, the same audit event is used to apply the balance adjustments for that bill. The adjustment amount is the difference between the charges calculated by using the base deal and the alternate deals and are grouped by resource, general ledger ID (G/L ID), and tax code.

If the account is a discount sharing account with multiple bills for best pricing, the adjustments are made to the bills based on net balance impacts from the corresponding bills.

How BRM Calculates the Best Price for Subscription Groups

To enable multiple services to use best pricing, you can use a subscription group consisting of a subscription service with which the deal is associated and member services that share the best pricing configuration. When you configure best pricing for the subscription group, ensure that the subscription group service type is the parent of the member service types. For example:

- Subscription service: `/service/telco/gsm`
- Member services:
 - `/service/telco/gsm/telephony`
 - `/service/telco/gsm/fax`
 - `/service/telco/gsm/voice`

For more information about subscription services, see "About Subscription Services" in *BRM Managing Customers*.

Calculating the Best Price for Subscription Services

If the subscription service has a best pricing configuration with n alternate deals and all the member services have regular deals, BRM calculates best pricing n times. The best pricing calculations include all the ratable events from the subscription service and the member services.

If you want all member service events to be included in the best pricing calculation, configure the best pricing deal at the subscription service level.

Calculating the Best Price for Member Services

If a member service has best pricing configured with m alternate deals, BRM calculates best pricing m times and includes only the events in that member service. If the subscription service has a best pricing configuration, only its base deal is used in the best pricing calculation of the member service.

To offer the best price only for a specific member service and exclude other member services in the subscription group, configure best pricing at that member service level.

How BRM Calculates the Best Price in Resource Sharing Groups

This section describes how BRM calculates best pricing in resource sharing groups. For more information on resource sharing groups, see "About Resource Sharing Groups" in *BRM Managing Accounts Receivable*.

Calculating the Best Price for a Discount Sharing Group

If best pricing is configured for a discount sharing group owner service, BRM includes all the ratable events from both the owner and member services in calculating the best price. It uses the alternate deals from the owner and base deals from the members to calculate the best price. It ignores the minimum charge values specified for alternate deals because the costlier alternate deal with the minimum charge might offer more savings to the discount sharing group members.

Best pricing calculation can result in rerating the events of the discount sharing group services. Therefore, billing for the discount sharing group members is finalized only after the best pricing calculation is performed.

If a discount sharing group member service has best pricing configured, best pricing calculation is performed for the member service when the member service is billed.

Calculating the Best Price for a Charge Sharing Group

Best pricing calculation ignores the charge sharing configuration and does not include the sponsored amount for the charge share owner and the members.

About Applying Exclusion Rules for Deals in a Best Pricing Configuration

You can specify exclusion rules for deals and discounts so that an account cannot own two mutually exclusive deals or discounts. Deal exclusion rules are applied when a deal is purchased, and discount exclusion rules are applied at billing time.

For more information, see "About Deal Dependencies" in *BRM Setting Up Pricing and Rating* and "About Discount Exclusion Rules" in *BRM Configuring Pipeline Rating and Discounting*.

Mutually Exclusive Deals and Best Pricing

When an add-on deal is purchased, if either the existing deal or the add-on deal is a best pricing deal and the two deals have an exclusive relationship, the exclusion rules also apply to the alternate deals. If the exclusion check results are not consistent across all the deals, the add-on purchase fails.

For example, suppose a service has a deal, A. If an add-on best pricing deal B with alternate deals Alt1 and Alt2 is purchased, exclusion check is performed for deal A against deals B, Alt 1, and Alt 2. If deal B is excluded from deal A, deals Alt1 and Alt2 must also be excluded. Otherwise, the add-on purchase fails.

Mutually Exclusive Discounts and Best Pricing

BRM applies discount exclusion rules after preparing the discount list from all the relevant deals, potentially reducing the number of discounts, before calculating the best price.

About Keeping Track of Best Pricing Information

BRM creates an audit event to record each best pricing calculation unless best pricing is performed in calc-only mode.

The audit event includes the following data:

- The base deal.
- The best deal, if the base deal is *not* the best deal.
- The balance impacts of rating using the base deal and the best deal.
- The start and end times of the best pricing calculation.
- How the best price is applied by rerating the events or by adjusting the balance with the difference in the charges between the best and base deals.

About Configuring BRM to Use Best Pricing

Before you can use best pricing, you must configure BRM to use best pricing by updating the subscription parameter class in the `/config/business_params` object by using the `pin_bus_params` utility. For information on `pin_bus_params`, see *BRM Developer's Guide*.

Enabling Best Pricing

To enable best pricing:

1. Create an editable XML file for the subscription parameter class by using the following command:

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

This command creates an XML file named `bus_params_subscription.xml.out` in your working directory. If you do not want this file in your working directory, specify the full path as part of the file name.

2. In the `BusParamsSubscription` section, enable best pricing as follows:

```
<BusParamsSubscription>
...
  <BestPricing>enabled</BestPricing>
</BusParamsSubscription>
```

3. Load the change into the `/config/business_params` object by using the following command:

```
pin_bus_params bus_params_subscription.xml
```

Run this command from the `BRM_Home/sys/data/config` directory, which includes support files used by the utility. `BRM_Home` is the directory where you installed BRM components. To run it from a different directory, see `pin_bus_params`.

4. Read the object with the `testnap` utility or Object Browser to verify that all fields are correct.

For general instructions on using **testnap**, see "Using testnap" in *BRM Developer's Guide*. For information on how to use Object Browser, see "Reading Objects by Using Object Browser" in *BRM Developer's Guide*.

The resulting flist of the **testnap** utility must resemble this example, with the PIN_FLD_PARAM_VALUE field value set to **1**:

```
0 PIN_FLD_POID          POID [0] 0.0.0.1 /config/business_params 9806 0
0 PIN_FLD_ACCOUNT_OBJ  POID [0] 0.0.0.1 /account 1 0
0 PIN_FLD_DESCR        STR [0] "Business logic parameters for Subscription"
0 PIN_FLD_HOSTNAME     STR [0] "-"
0 PIN_FLD_PARAMS       ARRAY [1] allocated 4, used 4
1 PIN_FLD_DESCR        STR [0] "Parameter to enable or disable best pricing
feature.
                                Enabling this feature will be effective only
if license
                                is loaded for best pricing. 1 means
enabled."
1 PIN_FLD_PARAM_NAME   STR [0] "best_pricing"
1 PIN_FLD_PARAM_TYPE   INT [0] 1
1 PIN_FLD_PARAM_VALUE  STR [0] "1"
```

5. Stop and restart the Connection Manager (CM) after editing the object. See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

Setting Up Pipeline-Triggered Billing

This chapter provides a conceptual overview and instructions on setting up Oracle Communications Billing and Revenue Management (BRM) pipeline-triggered billing.

Before reading this chapter, read the following:

- [About Billing](#)
- "About Pipeline Rating" in *BRM Configuring Pipeline Rating and Discounting*
- [About the Billing Utilities](#)
- "About Standard Recycling" in *BRM Configuring Pipeline Rating and Discounting*

About Pipeline-Triggered Billing

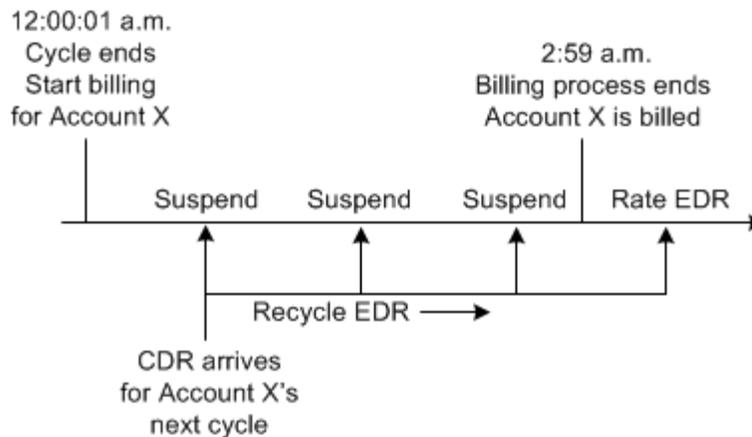
You use pipeline-triggered billing when event data records (EDRs) arrive for the next accounting cycle before the associated accounts have been billed. Pipeline Manager triggers billing for these accounts, enabling the new usage to be rated sooner and reducing the number of EDRs that might need suspending or rerating.

When customers use their services while their accounts are being billed, Pipeline Manager does not rate their usage until the accounts' billing is complete. The BRM billing process can sometimes take several hours. When there is account activity during the billing process, there is a greater possibility that call detail records (CDRs) will need recycling or rerating. When you use pipeline-triggered billing, the accounts are billed in a separate billing process, reducing the billing processing time.

When Pipeline Manager suspends EDRs because it is waiting for the account to be billed, those EDRs must later be recycled for rating. Each time an EDR is recycled, it is suspended until Pipeline Manager receives notification that the billing process for the account is complete.

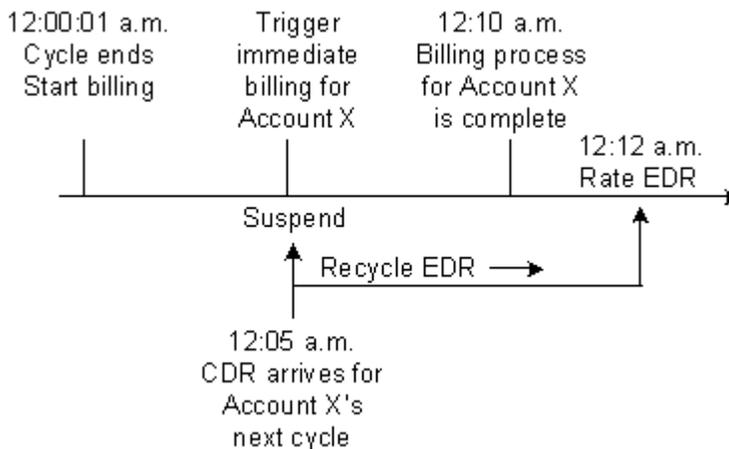
[Figure 7-1](#) shows how a CDR is repeatedly suspended for Account X until the account's billing is complete. In this example, the billing process takes almost three hours.

Figure 7-1 Suspension of CDR Rating During Billing



With pipeline-triggered billing, billing is triggered for Account X when the first new CDR arrives, as shown in [Figure 7-2](#). The account is billed in a separate billing process, reducing the processing time. Pipeline Manager can then rate the recycled CDR and new CDRs that arrive for the next accounting cycle for that account:

Figure 7-2 Pipeline-Triggered Billing



Note: Performance is affected by the number of accounts that need pipeline-triggered billing. If too many accounts are triggered for billing by Pipeline Manager, there is no performance advantage.

Pipeline-Triggered Billing Components

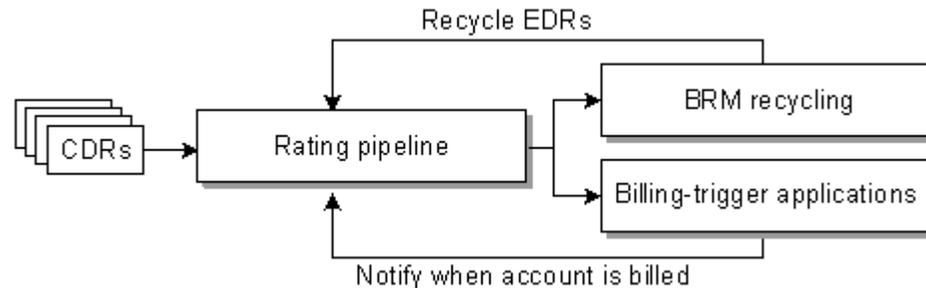
Pipeline-triggered billing comprises several components that work together to bill accounts:

- **Pipeline Manager modules** flag and route EDRs for accounts that require billing. See "[About the Pipeline-Triggered Billing Modules](#)".
- **BRM recycling** suspends EDRs flagged to trigger billing until the accounts are billed. The EDRs are periodically recycled. When the accounts are billed, the recycled EDRs can be rated and are no longer suspended. See "[About Suspending Billing-Trigger EDRs](#)".

- **Billing-trigger applications** bill the triggered accounts. These applications include a billing batch handler (BillHandler) and the BRM billing utility. When the accounts are billed, BRM notifies Pipeline Manager that billing for the accounts is complete. See "[About BillHandler](#)".

Figure 7–3 shows the relationships among Pipeline Manager, BRM standard recycling, and the billing-trigger applications.

Figure 7–3 Pipeline Manager, Recycling and Billing Applications Relationship



How Pipeline Manager Triggers Billing

The following steps describe the entire process of pipeline-triggered billing:

1. BRM runs billing when the accounting cycle ends.
2. An EDR enters a pipeline for processing.
3. The FCT_TriggerBill module checks if the EDR belongs to the next accounting cycle and if partial billing has not yet been triggered for the account. When both of these conditions are true, it sends the EDR to the billing-trigger output stream.
4. The FCT_Reject module sends the EDR to the suspense output stream.
For more information, see "[About the Pipeline-Triggered Billing Modules](#)".
5. When the EDR reaches the output modules, the data in the EDR takes two routes:

Suspend EDR:

- a. The suspense output module suspends the EDR by sending it to Suspended Event (SE) Loader. SE Loader stores the EDR in the BRM database.
- b. The `pin_recycle` utility retrieves the EDR from the BRM database and sends it back through the rating pipeline. The EDR begins the cycle again at step 2: If billing for the account is complete, the EDR is rated. If billing is not complete, the EDR is again suspended.

For more information, see "[About Suspending Billing-Trigger EDRs](#)".

Trigger billing:

- a. The pipeline billing-trigger output module creates a file containing the account and bill units associated with the EDR.
- b. The billing-trigger applications retrieve the file and bill the account associated with the EDR.

For more information, see "[About BillHandler](#)".

- c. BRM uses the Account Synchronization Data Manager (DM) to notify Pipeline Manager that the account is billed.

For more information about the account synchronization process, see "Installing and Configuring the Account Synchronization DM" in *BRM Installation Guide*.

- d. When the recycled EDR is sent back through a rating pipeline (through the Suspend EDR route), the EDR is rated because the account has been billed.

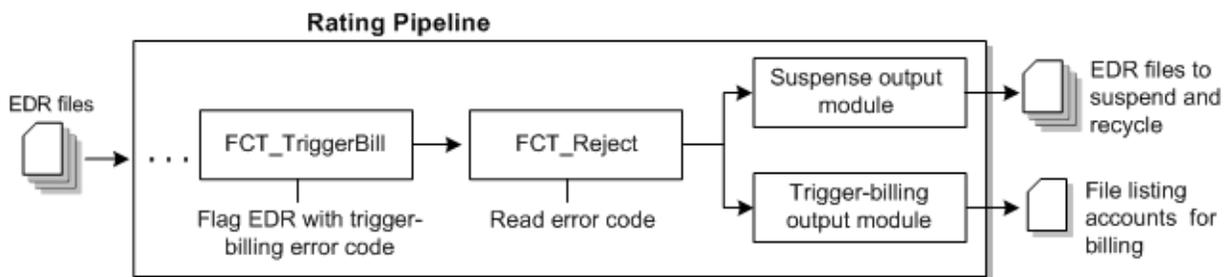
About the Pipeline-Triggered Billing Modules

Pipeline Manager flags EDRs associated with accounts that require immediate billing and sends them to the appropriate output streams. Pipeline-triggered billing uses the following modules:

- The `FCT_TriggerBill` module, which determines whether EDRs should trigger billing based on the accounting cycle date and billing state. To trigger billing, it sets a billing-trigger error code (**Awaiting billing of account**) in the EDRs. To flag the EDRs for recycling, it sets a billing-trigger recycle key value (**Trigger_Billing**).
- The `FCT_Reject` module, which detects the billing-trigger error code and sends the EDR to the suspense output stream.
- The billing-trigger output module, which creates a file containing the accounts and bill units for billing and sends the file to a separate directory.
- The suspense output module, which adds the EDRs that trigger billing to an output file. The EDRs are loaded into the BRM database to be suspended and recycled.

Figure 7-4 shows the path that EDRs take when they are flagged to trigger billing:

Figure 7-4 EDR Path for Triggered Billing

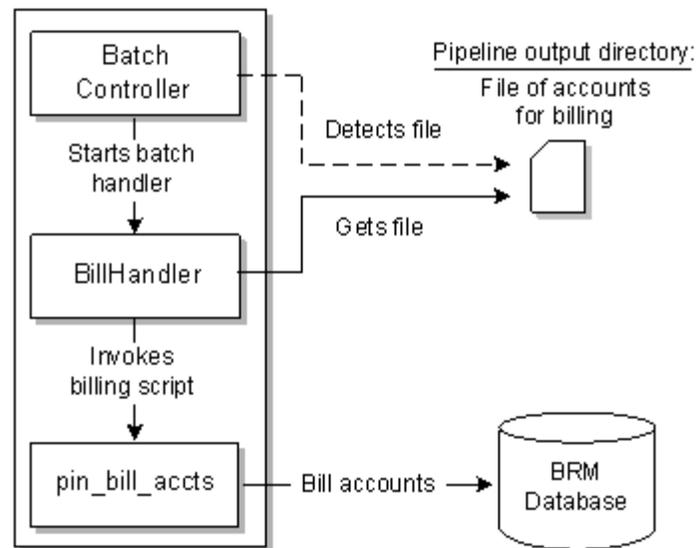


About BillHandler

BillHandler bills the accounts whose EDRs are flagged to trigger billing. BillHandler is used with the following applications:

- **Batch Controller**, which watches for billing-trigger files output by a pipeline. When a file is present, Batch Controller starts BillHandler.
- **The `pin_bill_accts` billing utility**, which bills the accounts and bill units. BillHandler starts the billing utility.

Figure 7-5 shows the batch handler billing process:

Figure 7-5 Batch Handler Billing Process

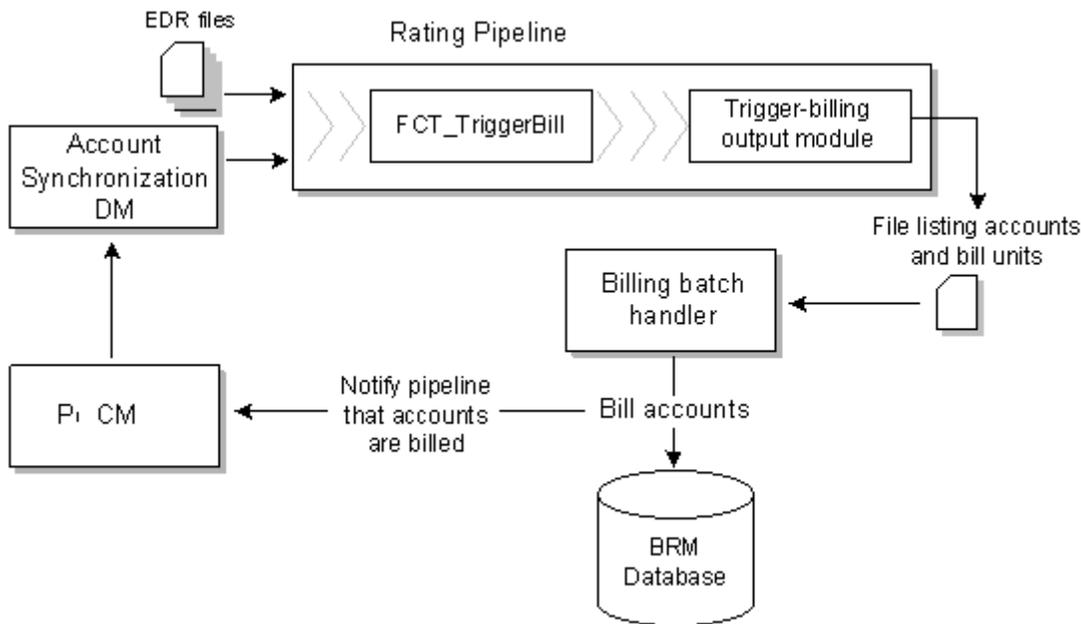
Overview of the Immediate Billing Process

To bill accounts identified for immediate billing, the following actions are performed:

1. Batch Controller starts BillHandler when a file is present in the pipeline billing-trigger output directory.
2. BillHandler reads the accounts and bill units in the file and passes them to the **pin_bill_accts** billing utility to be billed.
3. The **pin_bill_accts** billing utility creates the bills and updates the account information in the BRM database.
4. When the accounts are billed, BRM notifies Pipeline Manager by sending a business event to the Account Synchronization DM.

For information about synchronizing account data, see "Installing and Configuring the Account Synchronization DM" in *BRM Installation Guide*.

Figure 7-6 shows an overview of the processes required to bill accounts that are identified for immediate billing:

Figure 7-6 Immediate Accounts Billing Process

About Suspending Billing-Trigger EDRs

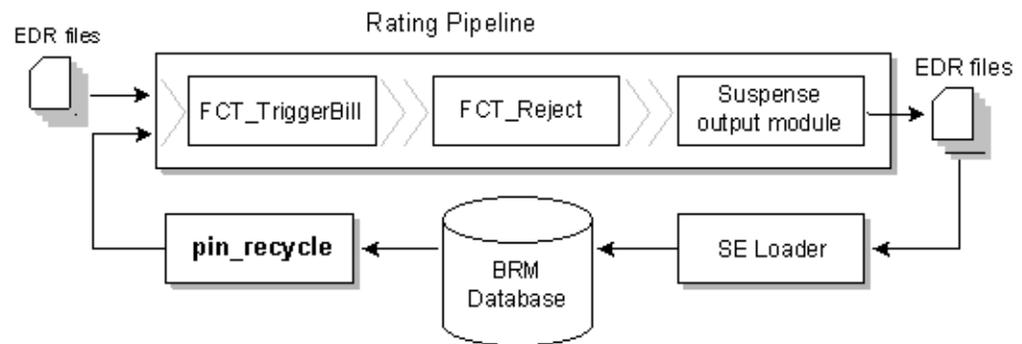
EDRs that are flagged to trigger billing are not rated and must be periodically recycled until the accounts are billed. Pipeline-triggered billing uses BRM standard recycling to suspend and recycle these EDRs.

You perform the following actions to suspend and recycle EDRs:

1. When EDRs contain the **Awaiting billing of account** error code, the FCT_Reject module sends them to the suspense output stream.
2. SE Loader retrieves the EDRs from the output stream and stores them in the BRM database. For more information, see "About Standard Recycling" in *BRM Configuring Pipeline Rating and Discounting*.
3. The **pin_recycle** utility retrieves the EDRs from the BRM database and sends them back through the rating pipeline. To continuously recycle any waiting EDRs, schedule the utility to run periodically. For more information, see "Setting Up pin_recycle to Run Periodically" in *BRM Configuring Pipeline Rating and Discounting*.

EDRs are suspended each time they are recycled until the accounts are billed. After the accounts are billed, the EDRs are rated and are no longer sent to the suspense output stream.

Figure 7-7 shows an overview of the processes required to suspend and recycle EDRs flagged to triggered billing:

Figure 7-7 Suspension and Recycling of EDRs Flagged to Trigger Billing

Configuring Pipeline-Triggered Billing

To configure pipeline-triggered billing:

- Configure Pipeline Manager. See "[Setting Up Pipeline Manager to Trigger Billing](#)".
- Configure Batch Controller. See "[Setting Up the Billing Batch Applications](#)".
- Configure Pipeline Manager recycling. You can use either standard recycling or Suspense Manager with pipeline-triggered billing. For a comparison of these features, see "About the EDR Recycling Features" in *BRM Configuring Pipeline Rating and Discounting*.

Setting Up Pipeline Manager to Trigger Billing

To set up Pipeline Manager to trigger billing:

1. Configure the FCT_TriggerBill module. See "FCT_TriggerBill" in *BRM Configuring Pipeline Rating and Discounting*.

Use the **TriggerBillCreateStream** entry in the module registry to specify the billing-trigger output module.

2. Configure the billing-trigger output stream in the OUT_GenericStream module. See "Configuring EDR Output Processing" and "OUT_GenericStream" in *BRM Configuring Pipeline Rating and Discounting*.

The default output grammar file is **TriggerBilling_OutGrammar.dsc**.

The following is an example of the registry entries for the billing-trigger instance of the OUT_GenericStream and EXT_OutFileManager modules:

```

TriggerBillCreateOutput
{
  ModuleName = OUT_GenericStream
  ProcessType = RATING_PIPELINE
  EventType = /event/delayed/session/telco/gsm
  Module
  {
    Grammar = ./formatDesc/Formats/TriggerBill/TriggerBilling_OutGrammar.dsc
    DeleteEmptyStream = True
    OutputStream
    {
      ModuleName = EXT_OutFileManager
      Module
      {
        OutputPath = ./data/TriggerBill
      }
    }
  }
}
  
```

```

        OutputPrefix      = trigger_billing
        OutputSuffix      = .tb
        TempPrefix        = .
        TempDataPath      = ./data/TriggerBill
        TempDataPrefix    = trigger.billing.tmp.
        TempDataSuffix    = .data
        AppendSequenceNumber = True
        Replace           = True
    }
}
}
}

```

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

3. Add the format and mapping files to the `DataDescription` registry. For information about the `DataDescription` registry, see "Configuring EDR Output Processing" in *BRM Configuring Pipeline Rating and Discounting*.
 - The default **StreamFormats** file is **TriggerBilling.dsc**.
 - The default **OutputMapping** file is **TriggerBilling_OutMap.dsc**.

The following is an example of the billing-trigger entries in the `DataDescription` registry:

```

DataDescription
{
    Standard
    {
        ModuleName = Standard
        Module
        {
            StreamFormats
            {
                TRIGGERBILL_CREATE_OUTPUT
                = ./formatDesc/Formats/TriggerBill/TriggerBilling.dsc
            }
            . . .
            OutputMapping
            {
                TRIGGERBILL_CREATE_OUTPUT =
                    ./formatDesc/Formats/TriggerBill/TriggerBilling_
OutMap.dsc
            }
        }
    }
}

```

Setting Up the Billing Batch Applications

To set up the billing batch applications for pipeline-triggered billing:

- Configure batch controller to start `BillHandler`. See "[Configuring Batch Controller to Start BillHandler](#)".
- Configure `BillHandler`. See "[Configuring BillHandler](#)".

Configuring Batch Controller to Start BillHandler

Use Batch Controller to start BillHandler. You configure Batch Controller to start BillHandler when it detects a file in the pipeline billing-trigger output directory.

To configure Batch Controller:

1. Open the *BRM_Home/apps/batch_controller/Infranet.properties* file, where *BRM_Home* is the directory in which BRM is installed.
2. Add the entries shown in [Table 7–1](#) for BillHandler:

Table 7–1 BillHandler Entries

Entry	Description
batch.random.events	Specify the event identifier. For example: batch.random.events = Bill_Handler_file For pipeline-triggered billing, this event is the appearance of a billing-trigger file that is output by a pipeline.
<i>event_name.name</i>	Specify a description for the event identifier. For example: Bill_Handler_file.name = File passed to BillHandler
<i>event_name.file.location</i>	Specify the full path to the pipeline billing-trigger output directory. This is the directory where the billing-trigger output module deposits the file of accounts to be billed.
<i>event_name.file.pattern</i>	Specify the billing-trigger output file name. When Batch Controller detects a file with this name, it starts BillHandler. Tip: You can use an asterisk (*) to represent zero or more characters in the file name. No other wildcards are supported. For example: Bill_Handler_file.pattern = *.out
<i>event_name.handlers</i>	Specify BillHandler identifier. For example: Bill_Handler_file.handlers = BillHandler
<i>handler_name.name</i>	Specify a description for the BillHandler identifier. For example: BillHandler.name = Bill Handler that executes pin_bill_accounts
<i>handler_name.max.at.lowload.time</i> <i>handler_name.max.at.highload.time</i>	Specify the number of BillHandler instances that can run concurrently during periods of low-load and high-load usage. Typical default settings are 6 at low load and 3 at high load.
<i>handler_name.start.string</i>	Specify the command that starts BillHandler. The default is <i>BRM_Home/apps/pin_bill_handler/BillHandler.pl</i> .

3. Save and close the file.

For more information, see "Controlling Batch Operations" in *BRM System Administrator's Guide*.

Configuring BillHandler

BillHandler retrieves the pipeline-triggered billing output file and sends the account and bill units to the billing utility. After the accounts and bill units are billed, BillHandler deposits the input file to a separate directory.

To configure BillHandler:

1. Open the *BRM_Home/apps/pin_bill_handler/BillHandler_config.values* file.
2. Edit the entries listed in [Table 7-2](#):

Table 7-2 Configuration Values for BillHandler

Entry	Description
\$FILETYPE	Specify the EDR file-name pattern. For example, *.txt.bc. Note: The asterisk (*) represents zero or more characters in the file name. No other wildcards are supported. Batch Controller runs BillHandler for each file with a name that matches this pattern.
\$HANDLER_DIR	Specify the full path to the directory containing BillHandler, log, input, output, and other files. The default is <i>BRM_Home/apps/pin_bill_handler</i> .
\$pinBillActDir	Specify the full path to the directory containing the pin_bill_accts billing utility.
\$STAGING	Specify the full path to the BillHandler input file location. Note: This is typically the same location specified for \$HANDLER_DIR . You configure this location as the output directory.
\$PROCESSING	Specify the full path to the directory from which the billing-trigger files are processed. The default is \$pinBillActDir .

For information about other entries, see the **BillHandler_config.values** file.

3. Save and close the file.

For more information, see "Controlling Batch Operations" in *BRM System Administrator's Guide*.

About Bill Cycle Management

This chapter describes Oracle Communications Billing and Revenue Management (BRM) bill cycle management and explains how to implement it.

Before reading this chapter, read "[About Billing](#)".

About Managing Billing Cycles

The BRM billing process involves two types of cycles:

- **Accounting cycle:** An accounting cycle is always one month long. At the end of an accounting cycle, the balance impacts of all bill units (**/billinfo** objects) associated with that cycle are compiled and stored in bill items.
- **Billing cycle:** A billing cycle comprises one or more complete accounting cycles. For example, a quarterly billing cycle spans three accounting cycles. At the end of a billing cycle, a bill is finalized to request payment for charges accumulated in bill items during the billing cycle.

Both cycles begin on a bill unit's *accounting day of month* (PIN_FLD_ACTG_CYCLE_DOM in a **/billinfo** object). This is also the day on which end-of-cycle tasks associated with the bill unit's previous accounting or billing cycle, if any, are performed. For more information, see "[About Accounting and Billing Cycles](#)".

Note: When referring to the day on which billing is run, the BRM documentation often uses the term *billing day of month*. From a programmer's point of view, this term is synonymous with *accounting day of month*.

By default, the DOM on which an account is created automatically becomes the accounting DOM for all of the account's bill units. This can result in an uneven distribution of a system's billing operations across each month. For example, if most of your accounts are created at the beginning of each month, most of your system's billing operations are also performed at the beginning of each month.

To load-balance billing operations more effectively, bill cycle management provides a systematic way to allocate accounts to accounting DOMs while still permitting key customers to request a billing day that corresponds to their cash flow or other billing preferences. This system is based on billing segments (see "[About Billing Segments](#)").

About Billing Segments

A *billing segment* is a user-defined category, such as wholesale, retail, and senior citizen, that controls the DOM to which a bill unit (**/billinfo** object) can be assigned.

To implement bill cycle management, you create billing segments in your BRM system and then assign bill units to them. The way a billing segment is configured determines which DOMs are available to the bill units that belong to the segment.

Note: You should carefully assess the types of accounts that you handle and the way in which you want to distribute the billing load before you set up billing segments.

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. See "[About Accounting DOM Status](#)".
- 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](#)".

About Accounting DOM Status

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 8–1](#):

Table 8–1 *DOM Status Assignments*

DOM Status	Manually Assignable?*	Automatically Assignable?*
open	Yes	Yes
restricted	Yes	No

Table 8–1 (Cont.) DOM Status Assignments

DOM Status	Manually Assignable?*	Automatically Assignable?*
closed	No	No

* 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 ["How BRM Calculates Bill Due Dates"](#).

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`, where `BRM_Home` is the directory where you installed BRM components, and load its contents into the `/config/billing_segment` object in the BRM database.

Caution: 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 and close 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
```

Important:

- When you run the utility, the `pin_billing_segment.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 Billing Segment Configuration File Edits"](#).
 - This utility needs a configuration (`pin.conf`) file in the directory from which you run the utility. For information about creating configuration files for BRM utilities, see "Creating Configuration Files for BRM Utilities" in *BRM System Administrator's Guide*.
-
-

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. See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

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 testnap" 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 8–2](#):

Table 8–2 XML Elements in Billing Segment

XML Element or Attribute	Description	Possible Values
ID	<p>A number that identifies the billing segment in the BRM database.</p> <p>When a bill unit (<i>/billinfo</i> object) is linked to a billing segment, this number is put in the PIN_FLD_BILLING_SEGMENT field of the <i>/billinfo</i> object.</p> <p>An array of all the billing segment IDs is stored in the <i>/config/billing_segment</i> object.</p>	<p>To use bill cycle management to assign billing DOMs to bill units in the segment, specify any integer greater than or equal to 101.</p> <p>Note: ID 0 triggers BRM to use the non-bill-cycle-management assignment process. See "Assigning DOMs to /billinfo Objects".</p>
<i>string</i>	<p>A character string that describes the type of accounts in the billing segment (for example, wholesale or retail).</p>	<p>Minimum length is 1 character.</p> <p>Maximum length is 1023 characters.</p> <p>Note: This string is mapped to the PIN_FLD_DESCR field in the <i>/config/billing_segment</i> 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.</p>

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 8–3](#).

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. For more information, see ["About Accounting DOM Status"](#).
- 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 8–3 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. For more information, see "About Accounting DOM Status" .	One of the following: <ul style="list-style-type: none"> ■ open ■ restricted ■ closed Important: These values are case sensitive.
dom	A billing DOM.	Any two-digit value from 01 through 31 .

Table 8–3 (Cont.) DOM Assignments

XML Element or Attribute	Description	Possible Values
(Optional) maxAccounts	Maximum number of accounts that can be associated with the DOM–billing segment pair.	Any nonnegative integer.
(Optional) maxServices	Maximum number of services that can be associated with the DOM–billing segment pair.	Any nonnegative integer.
(Optional) NumAccounts	Number of accounts currently associated with the DOM–billing segment pair.	Any nonnegative 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 nonnegative 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>
```

```
</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` file.

For more information, see "About Validating XML Configuration Files" in *BRM System Administrator's Guide*.

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".

Caution: 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. See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.
- Automatically refresh the cache. See "Automatically Refreshing Billing Segment Data".

Important: 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. See "Starting and Stopping the BRM System" in BRM System Administrator's Guide.

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. See "Starting and Stopping the BRM System" in BRM System Administrator's Guide.

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. See "Starting and Stopping the BRM System" in BRM System Administrator's Guide.

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 (`/billinfo` object) to a billing segment, put the ID of the billing segment into the `PIN_FLD_BILLING_SEGMENT` field of the `/billinfo` object.

Caution: Every bill unit in an account and every subordinate 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.

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. See ["About Accounting DOM Status"](#).

- **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 (`/billinfo` object) 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 (see ["About Accounting DOM Status"](#)).

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 (see "[About Accounting DOM Status](#)").

Note: Essentially, the CSR is selecting the bill unit's *billing DOM*. See "[Related Documents](#)".

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 an 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**. (See "[About Accounting DOM Status](#)".)

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 in 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. See "[About Accounting DOM Status](#)".

To change a bill unit's billing segment, see "[Changing a Bill Unit's Billing Segment](#)".

About Bill Run Management

This chapter describes Oracle Communications Billing and Revenue Management (BRM) bill run management and explains how to implement it.

Before reading this chapter, read "[About Billing](#)".

About Managing Billing Runs

Bill run management enables you to accomplish the following tasks:

- Reduce the load and duration of billing runs. See "[About Reducing Billing Run Loads](#)".
- Bill one or more specified accounts. See "[Billing Only Specified Accounts and Bill Units](#)".
- Fine-tune bill due dates. See "[About Managing Bill Due Dates](#)".

About Reducing Billing Run Loads

By default, when you use the `pin_bill_day` script to run billing, the billing run includes *all* bill units (`/billinfo` objects) 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, bill run management enables you to 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.

For more information, see "[Reducing Billing Run Loads](#)".

Note: For information about other ways to make your billing runs more efficient, see "Tuning Billing Performance" in *BRM System Administrator's Guide*.

Billing Only Specified Accounts and Bill Units

To bill a single account or a limited set of accounts when you run the `pin_bill_day` script, you specify the accounts and their bill units (`/billinfo` objects) in a modified version of the billing run configuration file (`BRM_Home/apps/pin_bill/pin_bill_run_control.xml`, where `BRM_Home` is the directory where you installed BRM components).

When you specify the accounts to bill, BRM does not perform a database search but retrieves the account and bill unit information directly.

Important: If you specify a paying bill unit that is part of a billing hierarchy, ensure that all subordinate bill units are billed first.

1. Open the **pin_bill_run_control.xml** file in an XML editor or a text editor.
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 ID of the account and bill unit.

Important: 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 ID 55612 and its bill unit with POID ID 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 ID 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_bill/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 filename is in the same directory from which you run the **pin_bill_day** script, specify only the file name.
- If filename is in a different directory from which you run the **pin_bill_day** script, you must 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. See "[Manually Running the pin_bill_day Script](#)".
- 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 to Run in a Multischema Environment](#)".

Caution: When you run **pin_bill_day** with a filename, do not run it as a **cron** job. If you do, depending on the restrictions in filename, some bill units might never be billed.

About Managing Bill Due Dates

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.

To add flexibility to bill due dates, bill run management uses these features:

- [About Payment Terms](#)
- [About Billing Calendars](#)
- [About Billing Run-Time Due Date Adjustments](#)

About 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.

For more information, see ["Managing Payment Terms"](#).

About 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, 2004 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.

For more information, see ["Managing Billing Calendars"](#).

About Billing Run-Time Due Date Adjustments

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.

For more information, see ["Specifying Due Date Adjustments in a Billing Run"](#).

Reducing Billing Run Loads

This section explains how to split a daily billing run into smaller billing runs. Splitting a billing run enables you to reduce its load and duration.

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 execute 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 (*/billinfo* objects). You execute 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 execute 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 systemwide billing delay.

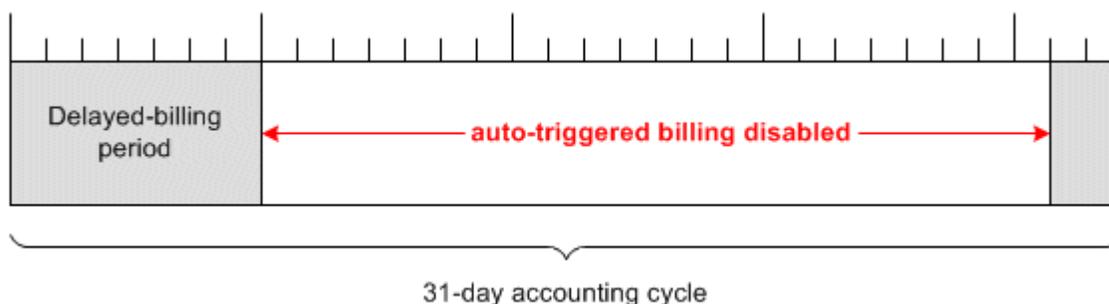
By default, auto-triggered billing is always enabled. To disable it, you must set a billing delay in your BRM system. See ["Disabling Auto-Triggered Billing by Specifying Billing Delay"](#).

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 9-1](#):

Figure 9-1 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 ["Disabling Auto-Triggered Billing by Setting AutoTriggeringLimit"](#).

For more information about auto-triggered billing, see ["About Auto-Triggered Billing"](#).

Splitting a Billing Run into Multiple Runs

To split a billing run triggered by the `pin_bill_day` script 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 (`/billinfo` objects) 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 "[About Billing Segments](#)".

Caution: When you split a large billing run into smaller billing runs:

- Be careful to 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_bill* 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 and close the file.

Tip: 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 copy filename to the same directory from which you run the **pin_bill_day** script, specify only the file name.
- 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/apps/pin_bill/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. See "[Manually Running the pin_bill_day Script](#)".
 - 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 to Run in a Multischema Environment](#)".
-
-

Caution: 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 sponsor group owner's bill one cycle late:

- Sponsor group owner account and member account have the same billing DOM.
- Sponsor 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 ["About Billing Segments"](#).

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 XSD *reference* file *BRM_Home/apps/pin_bill/business_configuration.xsd*.

For more information about the XSD reference file, see ["About Validating XML Configuration Files"](#) in *BRM System Administrator's Guide*.

Managing Bill Due Dates

This section explains how to manage bill due dates by performing the following tasks:

- [Managing Payment Terms](#)
- [Managing Billing Calendars](#)
- [Specifying Due Date Adjustments in a Billing Run](#)

About Configurable Bill Due Dates and Delayed Billing

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.

For information about delayed billing, see ["Setting Up Delayed Billing"](#).

Managing Payment Terms

A *payment term* specifies how to calculate the due date of a bill. See ["About Payment Terms"](#).

To manage payment terms, you perform these tasks:

- [Setting Up Payment Terms](#)
- [Editing the Payment Terms Configuration File](#)
- [Updating Payment Terms](#)
- [Updating the pin.conf File to Use Payment Terms](#)
- [Assigning Payment Terms to Bill Units](#)

Setting Up 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.

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

1. Open the `pin_payment_term.xml` file in an XML editor or a text editor. By default, 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 and close 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
```

Important:

- When you run the utility, the `pin_payment_term.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 Payment Terms Configuration File Edits"](#).
 - This utility needs a configuration (`pin.conf`) file in the directory from which you run the utility. For information about creating configuration files for BRM utilities, see ["Creating Configuration Files for BRM Utilities"](#) in *BRM System Administrator's Guide*.
-
-

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). See ["Starting and Stopping the BRM System"](#) in *BRM System Administrator's Guide*.
6. To verify that the payment term information was loaded, display the `/config/payment_term` object by using one of the following features:
 - Object Browser
 - `robj` command with the `testnap` utility

For general instructions on using `testnap`, see ["Using testnap"](#) 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*.
7. 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 ["Customizing Bill Due Date Calculations for Payment Terms"](#).

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.

To edit this configuration file, open it in an XML editor or a text editor.

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 the following table.

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 9-1](#):

Table 9-1 Payment Term Elements

XML Element or Attribute	Description	Possible Values
ID	The ID of the payment term.	Any nonnegative integer. Note: <ul style="list-style-type: none"> ■ 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 "Customizing Bill Due Date Calculations for Payment Terms" .
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 <code>/config/payment_term</code> 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>
```

Validating Your Payment Terms Configuration File Edits

After editing the contents of the XML file, you use the `"load_pin_payment_term"` utility to load the contents of the file into the `/config/payment_term` object in the BRM

database. See ["Setting Up Payment Terms"](#).

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_payment_term.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

For more information, see "About Validating XML Configuration Files" in *BRM System Administrator's Guide*.

Updating Payment Terms

To update payment term data, re-edit the payment term configuration file, and then run the `"load_pin_payment_term"` utility to load the updated contents of the file into the `/config/payment_term` object in the BRM database (see ["Setting Up Payment Terms"](#)).

Caution: This utility overwrites existing payment terms. When updating payment terms, you cannot load new terms only. You must load the complete set of payment terms each time you run the utility.

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. See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

Assigning Payment Terms to Bill Units

To assign a payment term to a bill unit (`/billinfo` object), 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.

Caution: 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.

Managing Billing Calendars

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. See "[About Billing Calendars](#)".

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).

Caution: The utility that loads billing calendars into the database 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.

1. Open the `pin_calendar.xml` file in an XML editor or a text editor.

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 and close 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
```

Important:

- 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*. See ["Validating Your Payment Terms Configuration File Edits"](#).
 - This utility needs a configuration (`pin.conf`) file in the directory from which you run the utility. For information about creating configuration files for BRM utilities, see "Creating Configuration Files for 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. See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.
6. To verify that the billing calendars were loaded, display the `/config/calendar` objects by using one of the following features:
 - Object Browser
 - `robj` command with the `testnap` utility

For general instructions on using `testnap`, see "Using testnap" 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 Calendar Configuration File

You configure all the billing calendars in your BRM system in the *BRM_Home/sys/data/config/pin_calendar.xml* file.

Important: 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 9–2](#):

Table 9–2 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: <ul style="list-style-type: none"> ■ 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.

Table 9–2 (Cont.) Calendar Elements

XML Element or Attribute	Description	Possible Values
Date	<p>A day on which you <i>do not</i> want bill payments to be due.</p> <p>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).</p> <p>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 <i>Date</i> element for Sundays and Saturdays in calendars used only by that function. See "How BRM Calculates Bill Due Dates".</p>	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 9–3](#):

Table 9–3 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.	<p>One of the following:</p> <ul style="list-style-type: none"> ▪ To associate the date with <i>one year only</i>, specify the appropriate year in <i>yyyy</i> format (for example, 2005). ▪ To associate the date with <i>all years</i> (a <i>recurring</i> 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

For more information, see "About Validating XML Configuration Files" in *BRM System Administrator's Guide*.

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.

Caution: 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. See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

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 "Functions for Calculating Payment Due Dates".

Specifying Due Date Adjustments in a Billing Run

Use billing run-time due date adjustments to add days to the due dates of bills in a billing run triggered by the pin_bill_day script. You can specify the same adjustment for all the bills in the billing run, or you can specify different adjustments for bills associated with different payment terms. See "About Billing Run-Time Due Date Adjustments".

Important: 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 "Setting Up 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_bill*d directory.

Note: You also use this file to split your daily billing run into smaller billing runs. See "Splitting a Billing Run into Multiple Runs".

2. Specify the appropriate due date adjustments in the file. See "Editing the Billing Run Configuration File to Adjust Bill Due Dates".
3. Save and close 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 copy filename to the same directory from which you run the `pin_bill_day` script, specify only the file name.
- 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/apps/pin_bill/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. See ["Manually Running the pin_bill_day Script"](#).
 - 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 to Run in a Multischema Environment"](#).
-
-

Caution: 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_bill/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 ["How BRM Calculates Bill Due Dates"](#).

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 ["About 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_bill/business_configuration.xsd

For more information, see "About Validating XML Configuration Files" in *BRM System Administrator's Guide*.

How BRM Calculates Bill Due Dates

The PCM_OP_BILL_POL_CALC_PYMT_DUE_T policy opcode calculates bill due dates. It is called by the PCM_OP_BILL_MAKE_BILL opcode and the PCM_OP_BILL_MAKE_CORRECTIVE_BILL opcode.

Important: By default, the due date calculation is based on the time that billing is *actually* run, not on the time that a bill unit is ready to be billed.

When called by the PCM_OP_BILL_MAKE_BILL opcode or the PCM_OP_BILL_MAKE_CORRECTIVE_BILL opcode, the PCM_OP_BILL_POL_CALC_PYMT_DUE_T policy opcode checks the PIN_FLD_PAYMENT_TERM value in the **/payinfo** object associated with the bill's **/billinfo** object and then performs the following operations:

- If the field has no payment term ID or if the ID is 0, the PCM_OP_BILL_POL_CALC_PYMT_DUE_T policy opcode uses the default process of calculating the due date rather than using the bill run management process (see "[About Bill Run Management](#)"). The default process involves the **fm_bill_pol_default_calc_due_t** function and either the PIN_FLD_DUE_DOM value or the PIN_FLD_RELATIVE_DUE_T value in the **/payinfo** object.
- If the payment term ID is *greater than 0*, the PCM_OP_BILL_POL_CALC_PYMT_DUE_T policy opcode uses one of the following bill run management functions to calculate the payment due date:
 - **fm_utils_add_n_days**
 - **fm_bill_pol_add_n_bus_days**
 - **fm_bill_pol_get_nthweekdayofmonth**

For more information, see "[Customizing Bill Due Date Calculations for Payment Terms](#)".

After calculating the due date, the PCM_OP_BILL_POL_CALC_PYMT_DUE_T policy opcode checks the due date adjustment value in the PIN_FLD_DUE_DATE_ADJUSTMENT field of its input flist. If the value is greater than 0, it adds the value to the due date.

The PCM_OP_BILL_POL_CALC_PYMT_DUE_T policy opcode then returns the payment due date to PCM_OP_BILL_MAKE_BILL, which puts it in the PIN_FLD_DUE_T field of the **/bill** object.

Customizing Bill Due Date Calculations for Payment Terms

You can customize how PCM_OP_BILL_POL_CALC_PYMT_DUE_T calculates due dates for regular bills as well as for corrective bills.

When the PCM_OP_BILL_MAKE_CORRECTIVE_BILL opcode calls PCM_OP_BILL_POL_CALC_PYMT_DUE_T to calculate the due date for a corrective bill, it identifies the bill from the PIN_FLD_NAME parameter. The possible values in PIN_FLD_NAME

are PIN_OBJ_NAME_CORRECTIVE_BILL, PIN_OBJ_NAME_CORRECTIVE_BILL_NOW, PIN_OBJ_NAME_CORRECTIVE_BILL_ON_DEMAND.

The value in PIN_FLD_NAME can be used to validate that PCM_OP_BILL_POL_CALC_PAYMT_DUE_T is calculating the due date for a corrective bill and, if necessary, provide custom logic to calculate due dates for the corrective bill.

For each payment term in your system, you must customize the PCM_OP_BILL_POL_CALC_PAYMT_DUE_T "fm_bill_pol_calc_pymt_due_t" function to specify which function and parameters to use to calculate the due dates of the bills associated with the payment term. To customize the function:

1. In the function's **switch** statement, add a **case** for each payment term defined in your `/config/payment_term` object.

The ID (constant expression) of each case should correspond to the ID of a payment term element in the PIN_FLD_PAYMENT_TERMS array of `/config/payment_term`. For example, if the array contains payment term element **1001**, add **case 1001** to the statement.

Note: Case numbers **1** through **1000** are for BRM use only.

2. In each **case**, call the appropriate function with the appropriate parameters. See ["Functions for Calculating Payment Due Dates"](#).

Functions for Calculating Payment Due Dates

To calculate payment due dates for bills associated with a payment term, call one of these functions in the corresponding **case** in "fm_bill_pol_calc_pymt_due_t":

- For payment terms that add *any type of day* (business, weekend, holiday, and so on) to the billing cycle end date, use this function:

fm_utils_add_n_days(*n*, &due_t)

For an example, see **case TERM1** in the "fm_bill_pol_calc_pymt_due_t" code sample.

Parameters

- *n* specifies the number of *days* to add to the billing cycle end date. The value can be any positive integer.
- **&due_t** is a pointer to the end date of the current billing cycle (PIN_FLD_ACTG_LAST_T in the `/billinfo` object). *n* is added to this date.
- For payment terms that add *only business days* to the billing cycle end date, use this function:

fm_bill_pol_add_n_bus_days(ctxp, *n*, "default", &due_t, ebufp)

For an example, see **case TERM2** in the "fm_bill_pol_calc_pymt_due_t" code sample.

Parameters

- **ctxp** points to a communication channel between the client application and the database.
- *n* specifies the number of *business days* to add to the billing cycle end date. The value can be any positive integer.

- **default** is the case-sensitive name of the default billing calendar. See "[About Billing Calendars](#)".

To use a different calendar, replace **default** with the value in the PIN_FLD_NAME field of the `/config/calendar` object that you want to use.

Note:

- This function automatically skips weekends, so you do not need to include Sundays and Saturdays in billing calendars used only by this function.
 - If the CM cache does not contain the specified `/config/calendar` object, the function uses the default process of calculating the payment due date rather than the bill run management process.
-
-

- **&due_t** is a pointer to the end date of the current billing cycle (PIN_FLD_ACTG_LAST_T in the `/billinfo` object). *n* is added to this date.
- **ebufp** is a pointer to the error buffer that contains any errors that occur when the function tries to retrieve the specified calendar from the database.
- For payment terms that specify *a particular day of month*, use this function:

fm_bill_pol_get_nthweekdayofmonth(*d*, *n*, &due_t)

For an example, see case **TERM3** in the "`fm_bill_pol_calc_pymt_due_t`" code sample.

Parameters

- *d* is a day of the week. The value can be **0** (Sunday), **1** (Monday), **2** (Tuesday), **3** (Wednesday), **4** (Thursday), **5** (Friday), or **6** (Saturday).
- *n* is the ordinal rank of *d* in a month. The value can be **1** (first *d* of the month), **2** (second *d* of the month), **3** (third *d* of the month), or **4** (fourth *d* of the month).
- **&due_t** is a pointer to the end date of the current billing cycle (PIN_FLD_ACTG_LAST_T in the `/billinfo` object).

Important: If **&due_t** is *after* *d n* of the month, *d n* of the *next* month is used.

For example, if *d* = 2, *n* = 3, and **&due_t** = April 19, 2004, the payment due date is April 20, 2004 (third Tuesday of April).

But, if *d* = 2, *n* = 3, and **&due_t** = April 21, 2004, the payment due date is May 18, 2004 (third Tuesday of the next month).

- You can also create custom functions to calculate payment due dates:
custom_function(custom_parameter, &due_t, ebufp)
 - **&due_t** is a pointer to the end date of the current billing cycle (PIN_FLD_ACTG_LAST_T in the `/billinfo` object). The function must take **due_t** as a parameter.
 - **ebufp** is a pointer to the error buffer that contains any errors that occur when the function tries to retrieve an object from the database. If the function uses BRM objects, this parameter is required.

fm_bill_pol_calc_pymt_due_t

This is the default implementation of the **fm_bill_pol_calc_pymt_due_t** function in the **fm_bill_pol_calc_pymt_due_t.c** file:

```

switch (ptt)
{
    case TERM1: /* Add 17 days to the billing cycle end date */
        fm_utils_add_n_days(17, &due_t);
        break;

    case TERM2: /* Add 14 business days to the billing cycle end date */
        fm_bill_pol_add_n_bus_days(ctxp, 14, "default", &due_t, ebufp);
        if (PIN_ERR_IS_ERR(ebufp))
        {
            PIN_ERR_LOG_EBUF(PIN_ERR_LEVEL_ERROR,
                "fm_bill_pol_calc_pymt_due_t: Error while using Payment Term 2",
ebufp);
            goto cleanup;
        }
        /*
        * If due_t is not changed, it means that there
        * are no /config/calendar objects available in the CM cache.
        * In this case, use the default TERM0 implementation.
        */
        if (due_t == *eff_tp)
        {
            PIN_ERR_LOG_MSG(PIN_ERR_LEVEL_WARNING, "Switching to default
calculation");
            fm_bill_pol_default_calc_due_t(&due_t, eff_tp, r_due_t, due_dom,
ebufp);
        }

        break;

    case TERM3: /* Make the due date the 3rd Tuesday of the month */
        fm_bill_pol_get_nthweekdayofmonth( 2, 3, &due_t);
        break;

    /* Add implementation for handling more payment terms here */

    default: /* TERM0 for backward compatibility */
        fm_bill_pol_default_calc_due_t(&due_t, eff_tp, r_due_t, due_dom,
ebufp);
}

```

About Bill Suppression

This chapter describes Oracle Communications Billing and Revenue Management (BRM) bill suppression and explains how to implement it.

Before reading this chapter, read "[About Billing](#)".

About Suppressing Bills

By default, 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. If necessary, an invoice is generated for 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.

Important: 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:

- [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](#)".

Note: To *suspend* bills, see "[About Suspending Billing of Accounts and Bills](#)". Unlike bill and account suppression, bill suspension *inactivates* bill units.

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 Events Associated with 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 "[Manually Suppressing Bills](#)".

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 deal can accumulate during account suppression. You can use this deal 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.

For more information, see "[Manually Suppressing Accounts](#)".

Suppressed Accounts versus Inactive Accounts

A suppressed account differs from an inactive account as shown in [Table 10-1](#):

Table 10-1 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.
Are the account's bills finalized?	Inactive account	Yes. Charges accrued before the account is inactivated age, get invoiced, and go into collections.

Table 10–1 (Cont.) Differences between Suppressed and Inactive Accounts

Condition	Account Type	Status
Can new charges accrue in the account?	Suppressed account	Yes. Optionally, charges associated with one account-level deal 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. Subordinate (nonpaying) child account bills 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 subordinate (nonpaying) bill units are inactivated.
Is sponsorship affected?	Suppressed account	No. Member accounts' sponsored charges continue to accrue in the suppressed owner account's unfinalized bill. Note: To prevent sponsor owner accounts from being suppressed, add the appropriate logic to the PCM_OP_BILL_POL_CHECK_SUPPRESSION policy opcode.
Is sponsorship affected?	Inactive account	Yes. Sponsorship is suspended. Formerly sponsored charges accrue in member account bills while the owner account is inactive.
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.
Is discount sharing affected?	Inactive account	Yes. Member account events impact member accounts' balances, not the inactive owner account's balance.

Exceptions to Bill Suppression

All types of bill suppression can be overridden by exceptions. The default exceptions are listed in [Table 10–2](#):

Table 10–2 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 "Customizing Bill Suppression Exceptions" .
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 10–2 (Cont.) Exceptions to Bill Suppression

Exception	Description
Maximum consecutive cycle suppressions exceeded	<p>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".</p> <p>Note:</p> <ul style="list-style-type: none"> ■ If the maximum number of consecutive cycles for the customer segment is 0, the bill can <i>never</i> 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 "[Customizing Bill Suppression Exceptions](#)".

Important: The **Bill Now** command, in Customer Center, and on-demand billing for deals and plans 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 automatic bill suppression, perform these tasks:

- [Setting Up Automatic Bill Suppression](#)
- [Associating Bill Suppression Information with Customer Segments](#)
- [Editing the Bill Suppression Configuration File](#)

To implement manual bill suppression, see "[Manually Suppressing Bills](#)".

To implement manual account suppression, see "[Manually Suppressing Accounts](#)".

Setting Up Automatic Bill Suppression

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](#)".

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.

Caution: The utility that loads bill suppression settings into the database overwrites all existing bill suppression settings. When updating the settings, you cannot load new settings only. You must load settings for each customer segment every time you run the utility.

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 and close the file.

- 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
```

Important:

- 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".
 - This utility needs a configuration (`pin.conf`) file in the directory from which you run the utility. For information about creating configuration files for BRM utilities, see "Creating Configuration Files for BRM Utilities" in *BRM System Administrator's Guide*.
-
-

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".

- Stop and restart the Connection Manager (CM). See "Starting and stopping the BRM system" in *BRM System Administrator's Guide*.
- 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 testnap" 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 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 finalize.
- 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 10-3](#):

Table 10–3 Customer Segment Bill Suppression Elements

XML Element or Attribute	Description	Possible Values
ID	<p>The ID of the customer segment.</p> <p>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.</p> <p>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.</p>	<p>Any nonnegative integer.</p> <p>Note:</p> <ul style="list-style-type: none"> 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	<p>Minimum balance required to finalize a bill. If the balance is less than this amount, the bill is automatically suppressed.</p>	<p>Any positive number with two decimal places.</p> <p>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.</p>
MaxSuppressionCycles	<p>Maximum number of consecutive billing cycles for which a bill can be suppressed.</p>	<p>Any integer greater than 0.</p>

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 this XSD *reference* file:

BRM_Home/sys/data/config/business_configuration.xsd

For more information about the XSD reference file, see "About Validating XML Configuration Files" in *BRM System Administrator's Guide*.

Manually Suppressing Bills

To suppress and unsuppress bills manually, call the PCM_OP_BILL_SET_BILL_SUPPRESSION opcode. This opcode enables you to accomplish the following tasks programmatically or through a custom user interface:

- Suppress a bill for a specified number of billing cycles.
- Unsuppress a manually suppressed bill.

If you use customer segments to set the maximum number of consecutive billing cycles that bills can be suppressed, be careful not to suppress bills manually for more than the specified maximum number of cycles (see "Associating Bill Suppression Information with Customer Segments"). When a suppressed bill exceeds the maximum consecutive cycle number, a suppression exception triggers BRM to generate the bill (see "Exceptions to Bill Suppression"). This might confuse customers who requested that their bills be manually suppressed for a longer period of time.

To determine the maximum number of consecutive cycles for which a bill *can* be suppressed:

1. Find out which customer segments the bill's account belongs to by checking the PIN_FLD_CUSTOMER_SEGMENT_LIST field in the /account object.
2. Check the PIN_FLD_MAX_SUPPRESSED_BILL_CYCLES field of the appropriate customer segment in the /config/suppression object.

Note:

- If an account belongs to more than one customer segment, the *lowest* PIN_FLD_MAX_SUPPRESSED_BILL_CYCLES value associated with the segments applies.
 - If an account belongs to no customer segments, the PIN_FLD_MAX_SUPPRESSED_BILL_CYCLES value of the default customer segment (ID 0) applies.
 - If an account belongs to no customer segments *and* your system has no default customer segment, the bill can be suppressed for an unlimited number of consecutive cycles.
-
-

To determine the current number of consecutive billing cycles for which a bill *has been* suppressed, check the PIN_FLD_NUM_SUPPRESSED_CYCLES field of the **/billinfo** object.

For more information, see ["About Manual Bill Suppression"](#).

To implement automatic bill suppression, see ["Automatically Suppressing Bills"](#).

To implement manual account suppression, see ["Manually Suppressing Accounts"](#).

Manually Suppressing Accounts

Manual account suppression enables you to accomplish the following tasks programmatically or through a custom user interface:

- Suppress an account immediately or on a specified future date. To do this, call the PCM_OP_BILL_SET_ACCOUNT_SUPPRESSION opcode.
 - Optionally, this opcode can purchase one account-level deal for the account to handle any purchase and cycle fees you want to associate with account suppression. The deal POID must be passed to the PIN_FLD_DEAL_OBJ field of the opcode's input list.
 - If you use customer segments to set the maximum number of consecutive billing cycles that bills can be suppressed, be careful not to suppress accounts manually for more than the specified maximum number of cycles (see ["Associating Bill Suppression Information with Customer Segments"](#)). When a suppressed bill exceeds the maximum consecutive cycle number, a suppression exception triggers BRM to generate the bill (see ["Exceptions to Bill Suppression"](#)). This might confuse customers who requested that their bills be manually suppressed for a longer period of time.

To determine the maximum number of consecutive cycles for which a bill can be suppressed:

Find out which customer segments the bill's account belongs to by checking the PIN_FLD_CUSTOMER_SEGMENT_LIST field in the **/account** object.

Check the PIN_FLD_MAX_SUPPRESSED_BILL_CYCLES field of the appropriate customer segment in the **/config/suppression** object.

Note:

- If an account belongs to more than one customer segment, the *lowest* PIN_FLD_MAX_SUPPRESSED_BILL_CYCLES value associated with the segments applies.
 - If an account belongs to no customer segments, the PIN_FLD_MAX_SUPPRESSED_BILL_CYCLES value of the default customer segment (ID 0) applies.
 - If an account belongs to no customer segments *and* your system has no default customer segment, the bill can be suppressed for an unlimited number of consecutive cycles.
-
-

To determine the current number of consecutive billing cycles for which a bill has been suppressed, check the PIN_FLD_NUM_SUPPRESSED_CYCLES field of the **/billinfo** object.

- Unsuppress an account immediately or on a specified future date. To do this, call the PCM_OP_BILL_SET_ACCOUNT_SUPPRESSION opcode.

Note: If a suppression deal was purchased when the account was manually suppressed, the deal's POID must be passed to the FLD_DEAL_OBJ field of the opcode's input flist to cancel the deal.

For more information, see ["About Manual Account Suppression"](#).

To implement bill suppression, see ["Automatically Suppressing Bills"](#) or ["Manually Suppressing Bills"](#).

How Bill Suppression Works

The following opcodes are used to suppress bills and accounts:

- PCM_OP_BILL_POL_CHECK_SUPPRESSION. See ["How BRM Determines Whether Bills Should Be Suppressed"](#).
- PCM_OP_BILL_SET_BILL_SUPPRESSION. See ["How BRM Suppresses Bills"](#).
- PCM_OP_BILL_SET_ACCOUNT_SUPPRESSION. See ["How BRM Suppresses Accounts"](#).
- PCM_OP_BILL_REMOVE_ACCOUNT_SUPPRESSION. See ["How BRM Ends Manual Account Suppression"](#).

How BRM Determines Whether Bills Should Be Suppressed

After performing the accounting activities at the end of a bill unit's billing cycle, the PCM_OP_BILL_MAKE_BILL opcode calls the PCM_OP_BILL_POL_CHECK_SUPPRESSION policy opcode to find out whether the bill should be finalized or suppressed. The PCM_OP_BILL_POL_CHECK_SUPPRESSION policy opcode performs these tasks:

1. Checks the PIN_FLD_PAY_TYPE field in the input flist to determine if the bill unit is subordinate. If it is, the opcode determines that it *should not* be suppressed.
2. From the cached `/config/suppression` object, the PCM_OP_BILL_POL_CHECK_SUPPRESSION policy opcode gets the following information for the customer segment or segments specified in the PIN_FLD_CUSTOMER_SEGMENT_LIST field of the opcode's input flist:
 - The minimum balance required for the bill to be generated.
 - The maximum number of consecutive billing cycles for which the bill can be suppressed.

Note:

- If the account belongs to multiple customer segments, the PCM_OP_BILL_POL_CHECK_SUPPRESSION policy opcode gets the lowest minimum balance and the lowest maximum cycle settings associated with the segments. The lowest settings do not have to be associated with the same customer segment.
 - If the input PIN_FLD_CUSTOMER_SEGMENT_LIST field contains a customer segment ID that is not associated with suppression information, the PCM_OP_BILL_POL_CHECK_SUPPRESSION policy opcode uses bill suppression information associated with the default customer segment (ID 0).
 - If the input PIN_FLD_CUSTOMER_SEGMENT_LIST field is empty, the PCM_OP_BILL_POL_CHECK_SUPPRESSION policy opcode uses bill suppression information associated with the default customer segment (ID 0).
 - If either of the two preceding items is true and the cached configuration object has no default customer segment *or* if the object is not in the cache, the bill does not qualify for automatic bill suppression (see "[About Automatic Bill Suppression](#)").
-
-

3. The PCM_OP_BILL_POL_CHECK_SUPPRESSION policy opcode checks whether any of the following is true:
 - The amount due (PIN_FLD_TOTAL_DUE) on the bill is less than the minimum balance specified in the customer segment.

Note: This check is not done on bills that do not qualify for automatic bill suppression. See step 2.

- The account is suppressed.
When an account is manually suppressed, the PIN_FLD_ACCT_SUPPRESSED field in each **/billinfo** object associated with the account is set to **1**. This value is put in the PIN_FLD_ACCT_SUPPRESSED field of the opcode's input flist.
4. The number of remaining manually suppressed billing cycles is greater than 0.
This value comes from the PIN_FLD_SUPPRESSION_CYCLES_LEFT field of the **/billinfo** object.
 5. The PCM_OP_BILL_POL_CHECK_SUPPRESSION policy opcode makes one of the following determinations:
 - If none of the preceding conditions is true, the bill *should not* be suppressed.
 - If at least one is true, the bill *should* be suppressed.
 6. If the bill *should* be suppressed, the PCM_OP_BILL_POL_CHECK_SUPPRESSION policy opcode checks for any exceptions to that suppression.

Note:

- For a list of default exceptions, see ["Exceptions to Bill Suppression"](#).
- To add or delete exceptions, see ["Customizing Bill Suppression Exceptions"](#).

To check for exceptions, the PCM_OP_BILL_POL_CHECK_SUPPRESSION policy opcode gets and uses the following information:

- Adjustment events associated with the bill's account. If an adjustment occurred after the last bill was generated, the bill must be finalized.

Note: In addition to adjustment events, if the *payment received* exception is in effect, the PCM_OP_BILL_POL_CHECK_SUPPRESSION policy opcode gets *payment* events associated with the account. In such cases, if a payment was received after the last bill was generated, the bill must be finalized. By default, this exception is commented out of the opcode. To uncomment it, see ["Adding Bill Suppression Exceptions"](#).

- The value in the input flist PIN_FLD_NUM_SUPPRESSED_CYCLES field, which indicates for how many consecutive billing cycles, the bill has been suppressed. If the value is equal to the maximum number specified in the customer segment, the bill must be finalized. See ["Associating Bill Suppression Information with Customer Segments"](#).

Note: If the maximum number of consecutive billing cycles for which the bill can be suppressed is 0 or missing (see step 2), this exception does not apply, and the bill can be suppressed for an unlimited number of consecutive billing cycles.

- The value in the input flist PIN_FLD_LAST_BILL_OBJ field. If NULL, it means that this is the bill unit's first bill and it must be finalized.
 - The value in the PIN_FLD_STATUS field in its input flist. If the value indicates that the status of the bill's account is closed, this is the bill unit's last bill and it must be finalized.
7. The PCM_OP_BILL_POL_CHECK_SUPPRESSION policy opcode then makes one of the following determinations:
- If an exception exists, the bill *cannot* be suppressed.
 - If no exception exists, the bill *can* be suppressed.

The PCM_OP_BILL_POL_CHECK_SUPPRESSION output flist contains the values listed in [Table 10-4](#). PCM_OP_BILL_MAKE_BILL uses these values to determine whether the bill should be suppressed or finalized:

Table 10-4 Output Flist for PCM_OP_BILL_POL_CHECK_SUPPRESSION

Output Flist Field	Value	Meaning
PIN_FLD_RESULT	0	Bill should not be suppressed.

Table 10–4 (Cont.) Output Flist for PCM_OP_BILL_POL_CHECK_SUPPRESSION

Output Flist Field	Value	Meaning
PIN_FLD_RESULT	1	Bill's balance is below the minimum required to finalize it, so bill should be automatically suppressed.
PIN_FLD_RESULT	2	Bill is manually suppressed.
PIN_FLD_RESULT	3	Account is manually suppressed.
PIN_FLD_EXCEPTION	0	No exception.
PIN_FLD_EXCEPTION	1	Payment, adjustment, or credit exception.
PIN_FLD_EXCEPTION	2	First bill exception.
PIN_FLD_EXCEPTION	3	Closed account exception.
PIN_FLD_EXCEPTION	4	Maximum cycle exception.

All types of bill suppression can be overridden by exceptions. For more information about the exceptions, see ["Exceptions to Bill Suppression"](#).

How BRM Suppresses Bills

The PCM_OP_BILL_SET_BILL_SUPPRESSION policy opcode handles manual bill suppression. See ["About Manual Bill Suppression"](#).

This opcode performs the following tasks:

- Suppresses a bill for a specified number of billing cycles.
 If the opcode's input flist PIN_FLD_SUPPRESSION_CYCLES_LEFT field contains a value greater than 0, the opcode sets the PIN_FLD_SUPPRESSION_CYCLES_LEFT field in a specified **/billinfo** object with that value.
 This value represents the number of consecutive billing cycles for which the bill is to be suppressed. At the end of each billing cycle, PCM_OP_BILL_MAKE_BILL subtracts 1 from this value in the **/billinfo** object. When the value reaches 0, bill suppression ends.
- Unsuppresses a manually suppressed bill.
 If the PIN_FLD_SUPPRESSION_CYCLES_LEFT field in a **/billinfo** object contains a value greater than 0, the bill is manually suppressed for the specified number of billing cycles. To end the suppression early, call PCM_OP_BILL_SET_BILL_SUPPRESSION, specify the appropriate **/billinfo** object in the input PIN_FLD_POID field, and set the input PIN_FLD_SUPPRESSION_CYCLES_LEFT field to 0.
- Generates an **/event/audit/suppression/bill** object each time it suppresses or unsuppresses a bill.

How BRM Suppresses Accounts

The PCM_OP_BILL_SET_ACCOUNT_SUPPRESSION opcode suppresses an account immediately or on a specified future date. See ["About Manual Account Suppression"](#).

This opcode performs the following tasks:

- Inactivates all services and products associated with the account specified in the PIN_FLD_POID field of its input flist.
- Optionally purchases an account-level deal for the account.

The deal POID must be specified in the PIN_FLD_DEAL_OBJ field of the opcode's input flist. It is the only *active* deal that can be associated with a suppressed account. You can use it to handle any purchase and cycle fees that you want to charge for suppressing the account.

- In each **/billinfo** object associated with the account, sets the PIN_FLD_ACCT_SUPPRESSED field to 1.
- Generates an **/event/audit/suppression/account/on** object.

To suppress the account on a future date, set the PIN_FLD_END_T field in the PCM_OP_BILL_SET_ACCOUNT_SUPPRESSION input flist to the appropriate date. When a date is specified in this field, the opcode schedules a call to itself on that date.

How BRM Ends Manual Account Suppression

The PCM_OP_BILL_REMOVE_ACCOUNT_SUPPRESSION opcode ends manual account suppression immediately or on a specified future date.

This opcode performs the following tasks:

- Removes any suppression deal associated with the account.
- Reactivates all services and products associated with the account specified in the PIN_FLD_POID field of its input flist.
- In each **/billinfo** object associated with the account, sets the PIN_FLD_ACCT_SUPPRESSED field to 0.
- Generates an **/event/audit/suppression/account/off** object.

To end account suppression on a future date, set the PIN_FLD_END_T field in the PCM_OP_BILL_REMOVE_ACCOUNT_SUPPRESSION input flist to the appropriate date. When a date is specified in this field, the opcode schedules a call to itself on that date.

Customizing Bill Suppression Exceptions

To manage bill suppression exceptions in your BRM system, customize the PCM_OP_BILL_POL_CHECK_SUPPRESSION policy opcode to perform these tasks:

- [Adding Bill Suppression Exceptions](#)
- [Deleting Bill Suppression Exceptions](#)

Important: To perform these tasks, you should be familiar with the following topics:

- C or C++ programming
- The Portal Communication Module (PCM) application programming interface (API)
- The Portal Information Network (PIN) library

See "Understanding the PCM API and the PIN Library" in *BRM Developer's Guide*.

Before reading this section, read "[Exceptions to Bill Suppression](#)".

Adding Bill Suppression Exceptions

To add a bill suppression exception to your system:

1. In the `fm_bill_pol_check_suppression.c` file, make the appropriate modifications to these functions:

- **fm_bill_pol_get_suppression_reason**

This function checks both for reasons to suppress a bill and for exceptions that override reasons to suppress.

- **fm_bill_pol_get_payment_adjustment_event**

If the new exception is based on an event, add the appropriate event class to the `eventsBuf` array in this function:

```
char eventsBuf[2000] = "/event/billing/adjustment/account',\
                        '/event/billing/refund/cash', '/event/billing/refund/cc'
,\
                        '/event/billing/refund/check', '/event/billing/refund/dd
',\
                        '/event/billing/refund/payorder', '/event/billing/refund
/postalorder',\
                        '/event/billing/refund/wtransfer', '/event/billing/rever
sal/cc',\
                        '/event/billing/reversal/check', '/event/billing/reversa
l/dd',\
                        '/event/billing/reversal/payorder',\
                        '/event/billing/reversal/postalorder',\
                        '/event/billing/reversal/wtransfer'";

/*****
* Payment exceptions are being commented/taken out of
the
* eventBuf buffer. If required these event ids can be
* included into the buffer in order to consider the
payment
* exceptions.
*****/

/*****/
/*'/event/billing/payment/cash',\
'/event/billing/payment/cc', '/event/billing/payment/che
ck',\
'/event/billing/payment/dd', '/event/billing/payment/pay
order',\
'/event/billing/payment/postalorder',
'/event/billing/payment/wtransfer'*/
```

Note: By default, the payment-received suppression exception is commented out of the preceding code and is thus disabled. To enable it, remove the comment symbols (`/*` and `*/`) enclosing the payment events.

2. In the `BRM_Home/include/pin_bill.h` file, add the appropriate enumerated name and value to the `bill_suppression_exceptions` variable:

```
typedef enum bill_suppression_exceptions {
    PIN_NO_EXCEPTION = 0,
    PIN_DUE_TO_PAYMENT_ADJUSTMENT_MADE = 1,
```

```

PIN_DUE_TO_FIRST_BILL = 2,
PIN_DUE_TO_ACCOUNT_CLOSED = 3,
PIN_DUE_TO_MAX_ALLOWED_SUPPRESSION_COUNT_REACHED = 4
} bill_suppression_exceptions_t;

```

Do not duplicate a value, and do not delimit the last name-value pair with a comma. If you add a name-value pair to the end of the existing list, add a comma to the end of the preceding name-value pair.

The values are used to populate the PIN_FLD_EXCEPTION field of the PCM_OP_BILL_POL_CHECK_SUPPRESSION policy opcode's output flist.

Deleting Bill Suppression Exceptions

To delete a bill suppression exception from your system:

1. In the `fm_bill_pol_check_suppression.c` file, delete or comment out the appropriate code in these functions:

- **fm_bill_pol_get_suppression_reason**

This function checks both for reasons to suppress a bill and for exceptions that override reasons to suppress.

Caution: Do not remove logic used by processes that check for other exceptions.

- **fm_bill_pol_get_payment_adjustment_event**

If the deleted exception is based on an event, remove the appropriate event class from the `eventsBuf` array in this function:

```

char eventsBuf[2000] = "/event/billing/adjustment/account',\
                        '/event/billing/refund/cash', '/event/billing/refund/cc'\
,\
                        '/event/billing/refund/check', '/event/billing/refund/dd'\
,\
                        '/event/billing/refund/payorder', '/event/billing/refund\
/postalorder',\
                        '/event/billing/refund/wtransfer', '/event/billing/rever\
sal/cc',\
                        '/event/billing/reversal/check', '/event/billing/reversa\
l/dd',\
                        '/event/billing/reversal/payorder', \
                        '/event/billing/reversal/postalorder', \
                        '/event/billing/reversal/wtransfer'";

/*****
****
the
payment
*****/

* Payment exceptions are being commented/taken out of
* eventBuf buffer. If required these event ids can be
* included into the buffer in order to consider the
* exceptions.

*****/

/*'/event/billing/payment/cash',\
'/event/billing/payment/cc', '/event/billing/payment/che\
ck',\

```

```

                                '/event/billing/payment/dd', '/event/billing/payment/pay
order', \
                                '/event/billing/payment/postalorder',
                                '/event/billing/payment/wtransfer'*/

```

2. In the *BRM_Home/include/pin_bill.h* file, delete or comment out the appropriate enumerated name and value from the **bill_suppression_exceptions** variable:

```

typedef enum bill_suppression_exceptions {
    PIN_NO_EXCEPTION = 0,
    PIN_DUE_TO_PAYMENT_ADJUSTMENT_MADE = 1,
    PIN_DUE_TO_FIRST_BILL = 2,
    PIN_DUE_TO_ACCOUNT_CLOSED = 3,
    PIN_DUE_TO_MAX_ALLOWED_SUPPRESSION_COUNT_REACHED = 4
} bill_suppression_exceptions_t;

```

If you remove the last name-value pair, also remove the comma at the end of the preceding name-value pair.

The values are used to populate the PIN_FLD_EXCEPTION field of the PCM_OP_BILL_POL_CHECK_SUPPRESSION policy opcode's output flist.

Creating Custom Bill Items

This chapter describes how to create custom bill items and how Oracle Communications Billing and Revenue Management (BRM) assigns custom bill items to events. To use this feature, you must understand the following BRM concepts and tasks:

- Bill items. See "[About Bill Items](#)".
- How to set up and configure batch pipeline rating. See "Configuring Pipeline Rating" in *BRM Configuring Pipeline Rating and Discounting*.
- How to customize policy opcodes. See "Adding and Modifying Policy Facilities Modules" in *BRM Developer's Guide*.

About Custom Bill Items

Bill items enable you to track a customer's balance for a type of billable event. For example, it tracks all charges for service usage or all charges for cycle fees during a billing cycle.

Bill items are also included in customer invoices, BRM reports, and customer service representative (CSR) applications as a way to summarize account activity in a billing cycle. For example, the Item Summary window in the Customer Center application shown in [Figure 11-1](#) summarizes the total charges for service usage and cycle forward fees for an account:

Figure 11-1 Invoice Bill Items for Service Usage and Cycle Forward Fees

Item Summary		
Item No.	Description	\$ Total
Accounts/Receivable Items		
B-7,1	Cycle forward	19.50
B-7,2	Cycle forward	6.00
B-7,3	Usage	0.00

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 "[About 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](#)".

About 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 Internet 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 11–2](#):

Figure 11–2 Rolled-Up Internet 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 product. A separate */item* object is created for each event of the same type. For example, if a customer purchases three ring tones 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 11–3](#):

Figure 11–3 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 11–1](#):

Table 11–1 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 11–1](#), 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 billinfo 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 poides 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. See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

For information about configuration files, see "Using Configuration Files to Connect and Configure Components" 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`.
- Roaming calls in a custom item object named `/item/roaming`.

In the example shown in [Figure 11-4](#), the customer's invoice displays an itemized list of GSM usage:

Figure 11-4 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 creating a custom iScript (batch rating) or by customizing a policy opcode (real-time rating). See "[Assigning Item Tags Based on Event Attributes](#)".
- Map items 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.

BRM can assign custom bill items to events during both real-time rating and batch rating. For details, see the following:

- [How Batch Rating Assigns Custom Bill Items](#)
- [How Real-Time Rating Assigns Custom Bill Items](#)

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.

How Batch Rating Assigns Custom Bill Items

BRM batch rating uses the FCT_ItemAssign pipeline module to assign items based on event and service combinations and uses custom iScripts to assign items based on event attributes.

To assign items to events, Pipeline Manager performs these tasks:

1. During initialization, the DAT_ItemAssign module loads the specified */config/item_types* object into memory and reserves a pool of POID IDs. If the information in the DAT_ItemAssign config object changes, you can use the DAT_itemAssign module's **Reload** semaphore to refresh the configuration changes.
2. Your custom iScript assigns an item tag based on event attributes to the *DETAIL.ITEM_TAG* EDR field.
3. FCT_ItemAssign calls DAT_ItemAssign with the item tag in the *DETAIL.ITEM_TAG* EDR field.
4. DAT_ItemAssign retrieves the item POID list from the DAT_AccountBatch module.
5. DAT_ItemAssign retrieves the item type for the given item tag from the *config/item_types* object in memory and searches the POID list for a matching item type.
6. If DAT_ItemAssign finds a matching POID, it returns that item POID (for example, *1/item/new_york m m*) to FCT_ItemAssign.

If DAT_ItemAssign does not find a matching POID, it creates a new POID ID from the POID pool it reserved and returns the new POID ID to FCT_ItemAssign.

When the DAT_ItemAssign module creates new items, it updates DAT_AccountBatch with the new items it created; for example, **1 /item/new_york m m**.

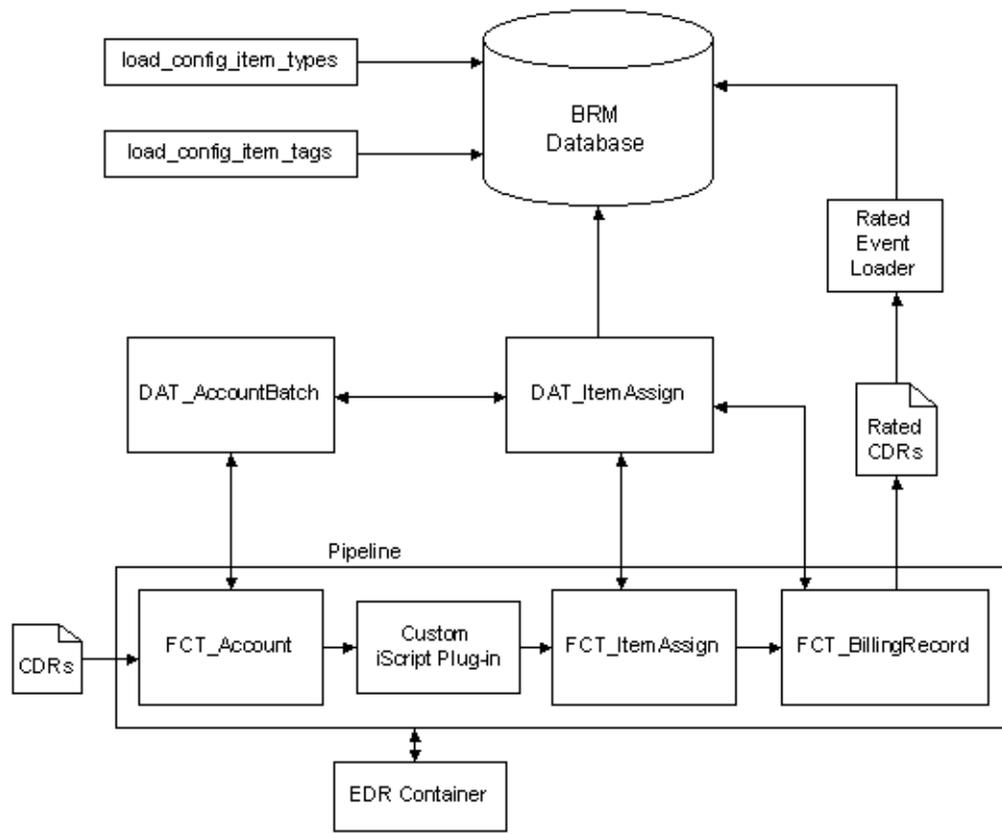
If the DETAIL.ITEM_TAG field is NULL, DAT_ItemAssign returns a default item POID from the item POID list.

7. The FCT_ItemAssign module assigns the item POID that it retrieves to the DETAIL.CUST_A.PRODUCT.SERVICE_USED_ITEM_POID EDR field for the product used for rating the event.
8. For sponsored usage events, the following occurs:
 - a. The FCT_BillingRecord module queries the DAT_ItemAssign module for items when required.
 - b. The DAT_ItemAssign module returns the pre-created items of type **/item/sponsor** to FCT_BillingRecord for sponsored events.
 - c. When the DAT_ItemAssign module creates new items, it updates DAT_AccountBatch with the new items it created; for example, **1 /item/sponsor m m**.
9. Pipeline Manager generates the rated events.

You use Rated Event (RE) Loader to load the rated events into the BRM database and to update the account balances and create or update bill items. For more information, see "Understanding Rated Event Loader" in *BRM Configuring Pipeline Rating and Discounting*.

Figure 11-5 shows how items are assigned to events:

Figure 11–5 Assignment of Items to Events



How Real-Time Rating Assigns Custom Bill Items

BRM real-time rating uses the `PCM_OP_ACT_USAGE` opcode to assign items by event and service combination and uses the `PCM_OP_BILL_POL_GET_ITEM_TAG` policy opcode to assign items by event attribute.

To assign items to usage events, BRM performs these tasks:

1. `PCM_OP_ACT_USAGE` uses the appropriate `/config/item_tags` object to assign the item tag associated with the event and service combination.
2. `PCM_OP_ACT_USAGE` calls the `PCM_OP_BILL_POL_GET_ITEM_TAG` policy opcode. If configured to do so, the policy opcode assigns a different item tag based on the event's attributes. By default, this policy opcode does nothing.
3. `PCM_OP_ACT_USAGE` uses the appropriate `/config/item_types` object to assign the item type associated with the item tag.
4. `PCM_OP_ACT_USAGE` assigns the event to an `/item` object, based on the item type:
 - **Cumulative items:** If the item already exists in the database, BRM assigns the event to the existing `/item` object. If the `/item` object does not already exist, BRM creates an `/item` object and then assigns the event to it.
 - **Individual items:** BRM creates a new `/item` object and assigns the event to it.

Note: If no match is found, BRM assigns the event to either the account-level or the service-level */item/misc* object, depending on whether the event belongs to an account or service.

5. PCM_OP_ACT_USAGE updates the item balance totals and then records the event in the database.

About Bill Items and Universal Event Loader

If you use Universal Event (UE) Loader to load events that correspond to a cumulative bill item, you might see more than one bill item in BRM. To ensure that UE Loader handles cumulative bill items correctly, UE Loader must group events by account. You define this setting when you create the event template in the UE Mapper. For more information, start Developer Center and see UE Mapper Help.

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.

Caution: 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. The method for assigning item tags differs for batch rating and real-time rating:

- For batch rating, see "[Setting Up Batch Rating to Assign Items Based on Event Attributes](#)".
- For real-time rating, see "[Setting Up Real-Time Rating to Assign Items Based on Event Attributes](#)".

Setting Up Batch Rating to Assign Items Based on Event Attributes

To set up Pipeline Manager to assign item tags to events based on event attributes:

1. Configure the following Pipeline Manager modules:
 - DAT_ItemAssign
 - FCT_ItemAssign
 - FCT_BillingRecord
2. Create a custom iScript that assigns item tags based on event attributes. See "[Creating a Custom iScript to Assign Item Tags](#)".
3. Load the rated events into the BRM database to update account balances and to create or update bill items. See "[Loading the Rated Events into the BRM Database](#)".

Creating a Custom iScript to Assign Item Tags

Create a custom iScript that assigns item tags based on event attributes and fills in the **DETAIL.ITEM_TAG** field in the EDR container. For information about creating an iScript and for the scripting language reference, see "Creating iScripts and iRules" in *BRM Developer's Guide*.

To enable your custom iScript to run in a pipeline, you must add an entry for it in the **wireless.reg** registry file. Configure this iScript to run *after* **FCT_Account** and *before* **FCT_BillingRecord**.

Sample registry entry:

```
# iScript to populate DETAIL.ITEM_TAG
#
```

```
iScript
{
  ModuleName = FCT_I_Script
  Module
  {
    Active      = TRUE
    Source      = File
    Scripts
    {
      ItemTag
      {
        #iScript file that you created
        FileName = ./ISC_ItemTag.isc
      }
    }
  }
}
# end of iScript
```

Loading the Rated Events into the BRM Database

You use Rated Event (RE) Loader to load rated events into the BRM database. Before updating items in the database, RE Loader checks the updater flag in the RE Loader **Infranet.properties** file. If the flag value is **1**, RE Loader creates in the database the new item objects that were added. By default, the flag is set to **1**.

For information on running RE Loader, see "Loading Prerated Events" in *BRM Configuring Pipeline Rating and Discounting*.

Setting Up Real-Time Rating to Assign Items Based on Event Attributes

You set up real-time rating 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.

For information about customizing policy opcodes, see "Adding and Modifying Policy Facilities Modules" in *BRM Developer's Guide*.

Mapping Item Tags to Item Types

You map the *item tags* that you defined in the `config_item_tags.xml` file, custom iScript, 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.

Caution: 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 "[About Tracking Charges in Bill Items.](#)"

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*.

Note: Tag names must be unique.

Note: Associate the event with the EAI framework publishing opcode (opcode number 1301).

To verify that the item types 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*.

Verifying Item-Tag-to-Item-Type Mapping

You can generate a log file that contains the item-tag-to-item-type mapping information from the **DAT_ItemAssign** memory.

To generate a log file of the mapping:

1. Create a semaphore registry file with following entry:

```
ifw.DataPool.ItemAssignDataModule.Module.PrintData=TagTypeMap.txt
```

2. Copy the file into the semaphore directory. The default directory for semaphore files is *BRM_Home/opt/ifw/semaphore*.

Pipeline Manager generates the **TagTypeMap.txt** file, which contains the tag and type mapping from the **DAT_ItemAssign** module memory.

For example, the file contains entries as follows:

Total number of Tag and Type Mapping entries: 3

 Tag : Type

cycle_forward : /item/cycle_forward
 misc : /item/misc
 newyork: /item/newyork

For more information, see "Semaphore File Syntax" in *BRM System Administrator'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 as well as 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 "Adding and Modifying Policy Facilities Modules" in *BRM Developer's Guide*.

4. Assign your custom items to event balance impacts during the rating process as shown in "[How Batch Rating Assigns Custom Bill Items to Events for Balance Impacts](#)".

By default, any item tag specified for a sponsor balance impact is ignored and an item of type `/item/sponsor` is used. A custom item type can only be assigned for a sponsor balance impact if:

- The `SplitSponsorItemByMember` business parameter is enabled. See "[Splitting Sponsored Charges into Multiple Items](#)".
- A custom item tag is specified for the sponsor balance impact in the `PIN_FLD_ITEM_TAGS` array.

How Batch Rating Assigns Custom Bill Items to Events for Balance Impacts

BRM batch rating uses the `FCT_ItemAssign` pipeline module to assign items based on event and service combinations and uses custom iScripts to assign items based on event attributes and balance impacts. The `FCT_BillingRecord` pipeline module converts impacts from products (`ChargePacket`), discounts (`DiscountPacket`), and taxes (`TaxPacket`) to balance impacts.

For more information on creating a custom iScript for balance impacts, see "[Creating a Batch Rating iScript for Balance Impacts](#)".

To assign custom bill items to events for balance impacts, Pipeline Manager performs the following tasks:

1. During initialization, the `DAT_ItemAssign` module loads the appropriate `/config/item_types` object into memory and reserves a pool of POID IDs. If the information in the `DAT_ItemAssign` config object changes, you can use the `DAT_itemAssign` module's **Reload** semaphore to refresh the configuration changes.
2. Your custom iScript does the following:
 - a. Assigns an item tag based on event attributes to the `DETAIL.ITEM_TAG` EDR field.
 - b. Changes the item POID for a specific balance impact in the event, using the following `ChargePacket`, `DiscountPacket`, and `TaxPacket` EDR tag container fields:
 - `DETAIL.ASS.CBD.CP.ITEM_TAG`
 - `DETAIL.ASS.CBD.DP.ITEM_TAG`
 - `DETAIL.ASS.CBD.TP.ITEM_TAG`
3. If the `ChargePacket`, `DiscountPacket`, and `TaxPacket` EDR container tag field values are `NULL`, `FCT_ItemAssign` calls `DAT_ItemAssign` with the item tag from the `DETAIL.ITEM_TAG` EDR field.

If the `ChargePacket`, `DiscountPacket`, and `TaxPacket` EDR container tag field values are not `NULL`, `FCT_ItemAssign` calls `DAT_ItemAssign` with the item tag from the `ChargePacket`, `DiscountPacket`, and `TaxPacket` EDR container fields and the item tag from the `DETAIL.ITEM_TAG` EDR field.

4. DAT_ItemAssign retrieves the item POID list from the DAT_AccountBatch module.
5. DAT_ItemAssign retrieves the item type for the given item tag from the **config/item_types** object in memory and searches the POID list for a matching item type.
6. If DAT_ItemAssign finds a matching POID, it returns that item POID (for example, **1 /item/new_york m m**) to FCT_ItemAssign.

If DAT_ItemAssign does not find a matching POID, it creates a new POID ID from the POID pool it reserved and returns the new POID ID to FCT_ItemAssign.

When DAT_ItemAssign creates new items, it updates DAT_AccountBatch with the new items it created; for example, **1 /item/new_york m m**.

If the DETAIL.ITEM_TAG field is NULL, DAT_ItemAssign returns a default item POID from the item POID list.

7. The FCT_ItemAssign module does the following:
 - a. Assigns the item POID that it retrieves to the DETAIL.CUST_A.PRODUCT.SERVICE_USED_ITEM_POID EDR field for the product used for rating the event.
 - b. Assigns the item POID that it retrieves from ChargePacket, DiscountPacket, and TaxPacket EDR container tag fields respectively to:
 - DETAIL.ASS.CBD.CP.ITEM_POID
 - DETAIL.ASS.CBD.DP.ITEM_POID
 - DETAIL.ASS.CBD.TP.ITEM_POID

Note: If no item tags are configured for ChargePacket, DiscountPacket, and TaxPacket, FCT_ItemAssign replaces the corresponding packet's item POID with the updated POID from DETAIL.CUST_A.PRODUCT.SERVICE_USED_ITEM_POID.

8. For sponsored usage events, the following occurs if the SplitSponsorItemByMember business parameter is enabled:
 - a. Pipeline Manager receives input from the **config_item_tags.xml** and **config_item_type.xml** files to determine the sponsor item type.
 - b. Assigns a type-only POID; for example, **/item/sponsor/usage-1**.
 - c. The RE Loader assigns an appropriate sponsor item instance to the sponsor balance impacts.
9. Pipeline Manager generates the rated events.

You use RE Loader to load the rated events into the BRM database and to update the account balances and create or update bill items. For more information, see "Understanding Rated Event Loader" in *BRM Configuring Pipeline Rating and Discounting*.

Creating a Batch Rating iScript for Balance Impacts

BRM batch rating uses custom iScripts to assign items based on event attributes.

Create a custom iScript that does the following:

1. Assigns an item tag based on event attributes to the `DETAIL.ITEM_TAG` EDR field.
2. Changes the item POID for a specific balance impact in an event, using the following fields in the EDR container:
 - `DETAIL.ASS_CBD.CP.ITEM_TAG`
 - `DETAIL.ASS_CBD.DP.ITEM_TAG`
 - `DETAIL.ASS_CBD.TP.ITEM_TAG`

To enable your custom iScript to run in a pipeline, you must add an entry for it in the `wireless.reg` registry file. Configure this iScript to run *after* `FCT_Apply_Balance` and *before* `FCT_ItemAssign`.

See ["Creating a Custom iScript to Assign Item Tags"](#) and ["Assigning Item Tags Based on Event Attributes"](#) for more information.

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 sponsored members of the account.

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"](#).

Important: When configuring a custom item type for sponsor balance impacts, specify the base type without the component `sponsor` in the item type string. For example, to use `peak_usage` for sponsor 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 sponsor items to event balance impacts during the rating process. See ["How Batch Rating Assigns Custom Bill Items to Events for Balance Impacts"](#).

Splitting Sponsored Charges into Multiple Items

By default, splitting sponsored charges into multiple sponsor items is disabled. You can enable the `SplitSponsorItemByMember` business parameter for real time rating, batch rating, or both.

To enable splitting sponsored charges into multiple items:

1. Go to the `BRM_Home/sys/data/config` directory, where `BRM_Home` is the directory in which you installed BRM components.
2. Run the following command, which creates an editable XML file from the `billing` instance of the `/config/business_params` object:

```
pin_bus_params -r SplitSponsorItemByMember -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. Search for the following line.

```
<SplitSponsorItemByMember>disabled</SplitSponsorItemByMember>
```

5. Do one of the following:
 - To enable splitting sponsor charges for both real time rating and batch rating, change `disabled` to `enabled`.
 - To enable splitting sponsored charges for only real time rating, change `disabled` to `onlyRealTime`.
 - To enable splitting sponsored charges for only batch rating, change `disabled` to `onlyBatch`.

Important: If you set `SplitSponsorItemByMember` to `onlybatch`, 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 sponsor items to events when the splitting option is enabled.

6. Save this file as `bus_params_billing.xml`.
7. Go to the `BRM_Home/sys/data/config` directory, which includes support files used by the `pin_bus_params` utility.
8. Run the following command, which loads this change into the appropriate `/config/business_params` object.

```
pin_bus_params PathToWorkingDirectory/bus_params_billing.xml
```

where `PathToWorkingDirectory` is the directory in which `bus_params_billing.xml` resides.

Caution: BRM uses the XML in this file to overwrite the existing `billing` 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 billing configurations.

Note: To run this command from a different directory, see "pin_bus_params" in *BRM Developer's Guide*.

9. Read the object with the **testnap** utility or Object Browser to verify that all fields are correct.
See "Using testnap" in *BRM Developer's Guide* for general instruction on using the **testnap** utility. See "Reading Objects by Using Object Browser" in *BRM Developer's Guide* for information on how to use Object Browser.
10. Stop and restart the CM. For more information, see "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.
11. (Multischema systems only) Run the **pin_multidb** script with the **-R CONFIG** parameter. For more information, see "pin_multidb" in *BRM System Administrator's Guide*.

Remitting Funds to Third Parties

This chapter describes how to use Oracle Communications Billing and Revenue Management (BRM) to remit a share of revenues to third parties. To use the remittance feature, you should be familiar with basic BRM functions, including pricing, rating, billing, and account creation. For an overview of these features, see *BRM Concepts*.

About Remittance

Use the BRM remittance feature to share the revenue you receive with third parties. You can direct BRM to calculate the amount of remittance in various ways: for example, you can pay a percentage of subscriber fees or a flat amount per new subscriber to third parties such as resellers or service providers.

Note: *Settlements* is a widely used industry term for *remittance*.

Table 12–1 lists some scenarios for using the BRM remittance feature:

Table 12–1 *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.
Branded Service Provider Payments	A host service provider needs to share revenues with branded service providers. For example, if subscribers pay their fees to the host provider, the host provider then remits a percentage of those fees to the branded service provider. Or the host provider pays a flat fee, similar to a sales commission, for every new subscriber the branded service provider signs up.
Sales Commissions	An ISP pays a commission for each subscriber its sales people sign 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 12–1 (Cont.) Remittance Scenarios

Scenario	Description
Telephony Settlements	<p>IP telephony calls connect with gateways and other telephony networks. A portion of the revenues collected for these calls go to the other carriers.</p> <p>For example, you must pay a call termination fee to the carrier that completes a call.</p> <p>Note: Implementing this example in BRM requires customizing the remittance fields file. See "Defining Custom Remittance Fields".</p>

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. For information about products and prices lists, see "About Creating a Price List" in *BRM Setting Up Pricing and Rating*.

When you create a remittance product, you must 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 "Ways to Rate Events" in *BRM Setting Up Pricing and Rating*.

Important: You cannot mix remittance and non-remittance products in a deal or plan.

Note: 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.

Important: 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.

Important: You must define an event type as one of your remittance criteria.

These fields are defined in the remittance fields file. You must load this file into BRM before you define remittance criteria.

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 or a brand name, you must 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 Utilities](#)".

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.

Important: 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.

Use each metric as shown in [Table 12-2](#):

Table 12-2 *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 5megabytes (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 branded service provider.

Note: You can also create custom metrics. See "About Setting Up RUMs for Real-Time Rating" in *BRM Setting Up Pricing and Rating*.

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.

Important: 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 must 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 the 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 "Creating Configuration Files for 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.

In place of *file_name*, enter 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 of **pin_remittance_flds** 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 testnap" 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`, and for an externally initiated payment when you submit a batch of payments through Payment Tool. You can also close a bill item when you transfer amounts between bill items through the Bill Details panel in Customer Center.

For more information on `pin_collect`, see *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 "Creating Configuration Files for 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 testnap" 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:

Important: You must follow this procedure every time you change the remittance specification file.

1. Combine the remittance specifications for all database schemas into one master remittance specification file.
2. Go to a directory that contains a valid configuration file (`pin.conf`) for connecting to your BRM database schemas. See "Creating Configuration Files for 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 master 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 testnap" 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 Utilities"](#).

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 Utilities"](#).

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 "Creating Configuration Files for 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 with 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 Configuring and Running Billing* or ["pin_inv_accts"](#) 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 12–3 contains an example of this report:

Table 12–3 Remittance Report Example

Acct No.	Start Date	End Date	Amount Remitted
0.0.0.1-9929	06/07/2001	08/13/2001	-676.2
0.0.0.1-10057	06/07/2001	08/13/2001	-9382.25
0.0.0.1-10185	06/07/2001	08/13/2001	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** in *BRM Configuring and Running Billing*.

Caution: 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 "About Adjustments" in *BRM Managing Accounts Receivable*.

Using Remittance with Brands

If your BRM system has Brand Manager, you can set up remittance to work with brands. Typically, you want to pay a different third party depending on which brand a subscriber account belongs to.

For example, if a subscriber account belongs to Brand A, you want to remit funds to Remittance Account A. But if the subscriber account belongs to Brand B, you want to pay Remittance Account B.

To do this:

1. Define a field in the remittance fields file called, for example, **brand_name**. For more information, see ["Defining Custom Remittance Fields"](#).
2. Set up a separate remittance specification for each brand. In each specification, include the following:
 - As one of the criteria, make **brand_name** equal to a particular brand.
 - Make the **remittance_account** equal to the account that receives remittance from subscribers of this brand.

This is a sample brand-specific remittance specification:

```
ACCOUNT_BEGIN

remittance_account_number    0.0.0.1-9617
remittance_type              B
remittance_product_name      Product 7a - Brand A Remittance
CRITERIA_BEGIN
  field  service_type        =          /service/ip
  field  product_name        =          IP async bulk 10
  field  event_type          =          /event/session/dialup
  field  brand_name          contains  Brand A
CRITERIA_END
```

For more information on creating specifications, see ["Defining Remittance Specifications"](#).

Using Remittance with Multiple Database Schemas

If you have multiple BRM database schemas, you must run the remittance utility (**pin_remitance**) 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 12-4](#):

Table 12-4 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

- In Pricing Center, create a deal and plan, as shown in [Table 12-5](#):

Table 12-5 Example Deal and Plan

Field	Deal	Plan
Name	Remittance Deal 1	Remittance Plan 1
Include	Remittance Product 1	Remittance Deal 1

- In Pricing Center, add Remittance Plan 1 to the CSR-new plan list and commit the plan list to your BRM database.
- In Customer Center, create an account that includes the settings shown in [Table 12-6](#):

Table 12-6 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.

- Load the default remittance field file into BRM. Go to `BRM_Home/data/pricing/examples` and enter:

```
load_pin_remittance_flds pin_remittance_flds
```
- In a text editor, open the remittance specification file, `BRM_Home/sys/data/pricing/example/pin_remittance_spec`.
- 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 "Creating Ratable Usage Metrics" in *BRM Setting Up Pricing and Rating*.

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 **brand_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.

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. You must 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
```

Important: 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, you must 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.

Part III

Running Billing

Part III describes how to run the Oracle Communications Billing and Revenue Management (BRM) billing utilities. It contains the following chapters:

- [Running Billing Utilities](#)
- [About Trial Billing](#)

Running Billing Utilities

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

For background information about BRM billing, see ["About Billing"](#).

For background information about corrective billing in BRM, see ["About Corrective Billing"](#).

To run billing, you should know how to use the **cron** and **crontab** commands.

Before running billing, you must set the billing configuration defaults, as described in ["Setting Business Policies for Billing"](#).

About Billing Your Customers

The way in which you bill customers depends on whether they are to receive a regular bill or a corrective bill for a prior bill.

Regular Billing Process

To bill customers, you run a set of billing utilities by running billing scripts automatically or manually on a daily, weekly, and monthly basis. See ["About Running the Billing Scripts"](#).

When you run the daily billing script, BRM does the following tasks:

1. Runs the **pin_deferred_act** utility to execute scheduled (deferred) actions. See ["Executing Deferred Actions with the pin_deferred_act Utility"](#).
2. Runs the **pin_bill_accts** utility to create bills for accounts, and to perform accounting cycle functions, such as creating new bill items. See ["Billing Accounts with the pin_bill_accts Utility"](#).
3. Runs the **pin_collect** utility to collect credit card payments. See "About Collecting BRM-Initiated Payments" in *BRM Configuring and Collecting Payments*.
4. Runs the **pin_refund** utility to generate refunds for BRM-initiated payments. See "About Refunds" in *BRM Managing Accounts Receivable*.
5. Runs the **pin_inv_accts** utility to create invoices. See ["Generating Invoices with the pin_inv_accts Utility"](#).
6. Runs the **pin_deposit** utility to deposit pre-authorized credit card payments, such as payments authorized when issuing a charge in Customer Center. See "About Collecting BRM-Initiated Payments" in *BRM Configuring and Collecting Payments*.

7. Runs the **pin_cycle_fees** utility to prorate balance impacts for cycle forward fees. See "[Prorating Cycle-Forward Fees and Canceling Products with the pin_cycle_fees Utility](#)".

In addition to running daily billing, you also run weekly and monthly billing scripts that run the **pin_collect** utility to collect payments that the daily billing was not able to collect.

For more information about the billing utilities, see "[About the Billing Utilities](#)". For information about handling billing failures, see "[Handling Billing Failures](#)".

Corrective Billing Process

The corrective billing process in BRM does not support the above mentioned scripts. The general steps in the process are:

1. Run the **pin_make_corrective_bill** utility to generate the corrective bill.
2. Run the **pin_inv_accts** to generate Corrective Invoices. See *BRM Configuring and Generating Invoices*.
3. Run the **pin_collect** utility. See "About Collecting BRM-Initiated Payments" in *BRM Configuring and Collecting Payments*.
4. Run the **pin_collections_process** utility. For more information on this utility see *BRM Developer's Reference*.

About the Billing Utilities

The billing scripts run the following utilities:

- The **pin_bill_accts** utility generates regular bills for the selected accounts. See "[Billing Accounts with the pin_bill_accts Utility](#)".
- The **pin_make_corrective_bills** utility generates corrective bills for the selected accounts. See "[Billing Accounts with the pin_make_corrective_bill Utility](#)".
- The **pin_inv_accts** utility generates both regular and corrective invoices. See "[Generating Invoices with the pin_inv_accts Utility](#)".
- The **pin_cycle_fees** utility prorates cycle-forward fees and cancels products. See "[Prorating Cycle-Forward Fees and Canceling Products with the pin_cycle_fees Utility](#)".
- The **pin_deferred_act** utility executes deferred actions. See "[Executing Deferred Actions with the pin_deferred_act Utility](#)".

The billing scripts also run payment utilities. See the following topics:

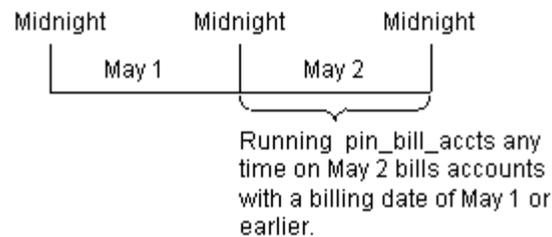
- "About Collecting BRM-Initiated Payments" in *BRM Configuring and Collecting Payments*
- "About Depositing BRM-Initiated Payments" in *BRM Configuring and Collecting Payments*
- "About Resolving Failed BRM-Initiated Payment Transactions" in *BRM Configuring and Collecting Payments*
- "About Recovering BRM-Initiated Payment Transactions" in *BRM Configuring and Collecting Payments*

Billing Accounts with the `pin_bill_accts` Utility

The `pin_bill_accts` utility is used to generate regular bills.

The `pin_bill_accts` utility calculates the balance due for each account bill unit, including all usage and cycle fees, 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 13-1](#).

Figure 13-1 Accounts Included when Running `pin_bill_accts`



The balance due amount is the amount requested as a payment by the `pin_collect` utility, and the amount that is shown on the invoice. For more information about the due amount, see "Fields in an Item" in *BRM Managing Accounts Receivable*.

The `pin_bill_accts` utility also performs the accounting cycle activity, such as creating new bill items. For more information, see "About accounting and billing cycles".

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 the subordinate hierarchy, you must run the `pin_bill_accts` utility to bill subordinate bill units before the 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.

Increasing Performance of the `pin_bill_accts` Utility

To increase billing performance, you run multiple threads of the `pin_bill_accts` utility simultaneously. See "Tuning Billing Performance" in *BRM System Administrator's Guide*.

Customizing the `pin_bill_accts` Utility

The `pin_bill_accts` utility uses the BRM MTA framework. You can customize `pin_bill_accts` by using the callback function and policy opcode hooks provided in the MTA framework.

For more information, see "Customizing BRM Multithreaded Client Applications" in *BRM Developer's Guide*.

Billing Accounts with the `pin_make_corrective_bill` Utility

The `pin_make_corrective_bills` utility generates corrective bills.

This utility is used to generate corrective bills for prior bills that have corrections or to process corrections on prior corrective bills.

The **pin_make_corrective_bills** utility validates that it can generate corrective bills for the selected bill, 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 accounts 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 more information about the due amount, see "Fields in an Item" in *BRM Managing Accounts Receivable*.

The **pin_make_corrective_bills** utility sets up an event for the type of corrective invoice to associate with the corrective bill.

For information about the **pin_make_corrective_bills** utility syntax, see "[pin_make_corrective_bills](#)".

When to Run the **pin_make_corrective_bills** Utility

Run the **pin_make_corrective_bills** utility when you have allocated all adjustments to prior bills.

When you generate corrective bills for an account hierarchy, you must run the **pin_make_corrective_bills** utility for the 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.

Increasing Performance of the **pin_make_corrective_bill** Utility

To increase billing performance, you run multiple threads of the **pin_make_corrective_bills** utility simultaneously. See "Tuning Billing Performance" in *BRM System Administrator's Guide*.

Customizing the **pin_make_corrective_bill** Utility

The **pin_make_corrective_bills** utility uses the BRM MTA framework. You can customize **pin_make_corrective_bills** by using the callback function and policy opcode hooks provided in the MTA framework.

For more information, see "Customizing BRM Multithreaded Client Applications" in *BRM Developer's Guide*.

Generating Invoices with the **pin_inv_accts** Utility

The **pin_inv_accts** utility generates regular invoices for regular bills and corrective invoices for corrective bills. Use the **pin_inv_accts** utility to generate regular invoices and corrective invoices and store them in the BRM database or in a separate database.

For more information about:

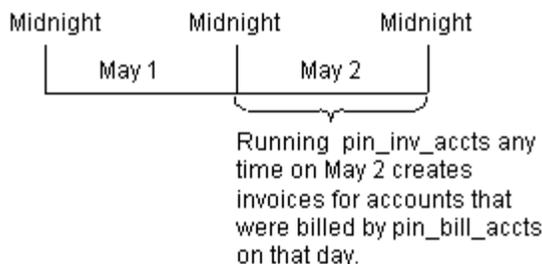
- Regular invoicing, see "Designing and Generating Invoices" in *BRM Designing and Generating Invoices*.
- Corrective invoicing, see "Corrective Billing and Invoicing Invoices" in *BRM Configuring and Running Billing*.
- The **pin_inv_accts** utility syntax, see "pin_inv_accts" in *BRM Designing and Generating Invoices*.

When to Run `pin_inv_accts`

For regular bills, typically, you run invoicing each day for accounts that had a regular bill created by the `pin_bill_accts` utility. For information, see "Running Daily Billing".

If you miss any billing days, the `pin_inv_accts` utility still generates invoices for accounts whose billing day was missed. This is because the `pin_bill_accts` utility creates bills for the missed billing days, and the `pin_inv_accts` utility generates invoices for those bills as shown in Figure 13-2.

Figure 13-2 Invoices Created when Running `pin_inv_accts`



For corrective bills, you run generate a corrective invoice when you create the corrective bill using the `pin_make_corrective_bill` utility.

Viewing Invoices

By default, you can view invoices through Customer Center, and your customers can view them on your Web pages. You can use XSL style sheets or other methods to design your invoices. For more information, see "Designing and Generating Invoices" in *BRM Designing and Generating Invoices*.

Emailing or Printing Invoices

To email invoices, or to print invoices for faxing or mailing, run the `pin_inv_send` utility. This utility is not included in the daily billing script by default. You can either add it to the daily billing script or run it separately.

For information, see "Sending Invoices to Customers" in *BRM Designing and Generating Invoices*.

Prorating Cycle-Forward Fees and Canceling Products with the `pin_cycle_fees` Utility

The `pin_cycle_fees` utility performs two tasks:

- Use this utility to 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 balance group.
- Use this utility to cancel products that have an expired pending cancellation. For example, if a product is set to cancel at a future date, the `pin_cycle_fees` utility cancels the product.

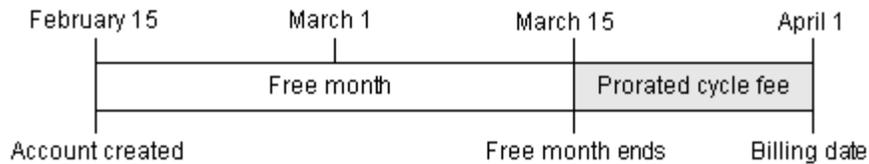
Important: These two tasks are performed by running the `pin_cycle_fees` utility twice with different parameters.

How the `pin_cycle_fees` Utility Prorates Cycle Forward Fees

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 13-3](#).

Figure 13-3 Proration of Cycle Forward Fees by `pin_cycle_fees`



The `pin_cycle_fees` utility checks for all free cycle-forward fees that have expired.

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.

Informing Customers That a Free Period Has Ended

You can find accounts that contain an expired free cycle-forward fee, and inform those customers that their free period has ended. To do so, run the `pin_cycle_fees` utility with the `verbose` and `test` options. Then use a custom utility to notify the customers. See "[pin_cycle_fees](#)".

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.

Executing Deferred Actions with the `pin_deferred_act` Utility

The `pin_deferred_act` utility enables you to execute scheduled actions. Using `/schedule` objects, you can schedule in advance when to activate, inactivate, or close an account or service. You can also schedule future changes to account groups. To ensure that the account status is correct before running billing, the `pin_deferred_act` utility makes all scheduled status and hierarchical group changes before running the other billing utilities.

For more information about the `pin_deferred_act` utility syntax, see "[pin_deferred_act](#)".

When to Run the `pin_deferred_act` Utility

Use the `pin_bill_day` script to run the `pin_deferred_act` utility daily. It is the first billing utility run by the `pin_bill_day` script.

About Running the Billing Scripts

BRM supports billing scripts for regular bills only.

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`, where `BRM_Home` is the directory where you installed BRM components. Table 13–1 shows the billing utilities in each script:

Table 13–1 Utilities in Billing Scripts

Billing Script	Description
<code>pin_bill_day</code>	Run daily. Runs the following billing utilities: <ul style="list-style-type: none"> ▪ <code>pin_deferred_act</code> ▪ <code>pin_bill_accts</code> ▪ <code>pin_collect</code> ▪ <code>pin_refund</code> ▪ <code>pin_inv_accts</code> ▪ <code>pin_deposit</code> ▪ <code>pin_cycle_fees</code>
<code>pin_bill_week</code>	Run weekly. By default, runs the <code>pin_collect</code> utility with the <code>rebill</code> option on all active accounts with a payment collection date 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.
<code>pin_bill_month</code>	Run monthly. By default, runs the <code>pin_collect</code> utility with the <code>rebill</code> option on all closed and inactive accounts with a payment collection date at least 31 days old. Collects outstanding balances from closed or inactive accounts.

Customizing the Billing Scripts

Billing scripts are located in `BRM_Home/bin`.

You can customize the billing scripts in the following ways:

- Specify which billing utilities to run.
- Use the parameters for each billing utility to specify how to run them.
- Set the error logging level.

Running Billing

Run billing as `pin_user`, not as `root`. Running billing as `pin_user` provides the read/write permissions for billing.

Note: Avoid running other BRM utilities while running billing as it may impact billing performance because of database contention.

What Time to Run Billing Scripts

Since billing generates a lot of system activity, it's best to run billing scripts at night.

Important: Use a different time for all three scripts so you do not run billing utilities simultaneously.

Manually Running the `pin_bill_day` Script

If you do not use bill run management, run the following command:

```
pin_bill_day
```

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 ["Splitting a Billing Run into Multiple Runs"](#).
- 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, you use this syntax:

```
pin_bill_day -file filename
```

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

The script passes the name of the file to the `pin_bill_accts` utility, which validates the file and then uses its contents to configure a billing run.

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` option.
 - When running `pin_bill_day` 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.
-
-

Caution: When you use a cron job to run `pin_bill_day`, *do not* include the configuration file name. If you do, depending on the restrictions in the configuration file, some bill units might never be billed.

Customizing the pin_bill_day Script for Best Pricing Options

To support best pricing, the **pin_bill_day** script includes the following entries for **pin_bill_accts**:

```
pin_bill_accts -discount -cycle_charge_only $1 $2 $3
pin_bill_accts -pay_type 10007 -cycle_charge_only $1 $2 $3
pin_bill_accts -sponsorship -cycle_charge_only $1 $2 $3
pin_bill_accts -cycle_charge_only $1 $2 $3
pin_bill_accts -pay_type 10007 -finalize_bill $1 $2 $3
pin_bill_accts -finalize_bill $1 $2 $3
```

\$1, **\$2**, and **\$3** are the parameters passed to the **pin_bill_accts** utility by the **pin_bill_day** script:

- **\$1** specifies the file to read for managing the billing run.
- **\$2** is the name of the billing run configuration file.
- **\$3** specifies whether to collect audit revenue data by item type or not.

For more information about these parameters, see "[About Bill Run Management](#)".

You can customize the **pin_bill_day** script to run **pin_bill_accts** with the options that suit your configuration.

To Bill Subordinates before Member Discounts

If a member of the discount sharing group is also a parent of a subordinate hierarchy, you can bill subordinates before discount sharing group members by using the following sequence in the **pin_bill_day** script:

```
pin_bill_accts -pay_type 10007 -cycle_charge_only $1 $2 $3
pin_bill_accts -sponsorship -cycle_charge_only $1 $2 $3
pin_bill_accts -pay_type 10007 -finalize_bill $1 $2 $3
pin_bill_accts -finalize_bill $1 $2 $3
```

This ensures that all subordinate accounts are billed before their parent accounts are billed.

To Bill Member Discounts before Subordinates

If a subordinate account is the owner of a discount sharing group, you can bill discount sharing group members before subordinates by using the following sequence in the **pin_bill_day** script:

```
pin_bill_accts -sponsorship -cycle_charge_only $1 $2 $3
pin_bill_accts -pay_type 10007 -cycle_charge_only $1 $2 $3
pin_bill_accts -pay_type 10007 -finalize_bill $1 $2 $3
pin_bill_accts -finalize_bill $1 $2 $3
```

Important: Ensure that the **billing_flow_discount** parameter in the **business_params** object is set to **2** to specify that member discounts must be billed before parents.

This ensures that all discount sharing group members are billed before the discount sharing group owner account is billed.

To Apply Cycle Fees for Subordinates and Sponsorship Members in One Run

If you have accounts where cycle fees can be applied independently for parents and children from the subordinate and sponsorship hierarchies, you can use the following sequence in the **pin_bill_day** script:

```
pin_bill_accts -discount -cycle_charge_only $1 $2 $3
pin_bill_accts -cycle_charge_only $1 $2 $3
pin_bill_accts -pay_type 10007 -finalize_bill $1 $2 $3
pin_bill_accts -finalize_bill $1 $2 $3
```

To Apply Cycle Fees for Subordinates, Sponsorship Members, and Discount-Sharing Group Members in One Run

If you have accounts where cycle fees can be applied independently for parents and children for all hierarchies, you can use the following sequence in the **pin_bill_day** script:

```
pin_bill_accts -cycle_charge_only $1 $2 $3
pin_bill_accts -pay_type 10007 -finalize_bill $1 $2 $3
pin_bill_accts -finalize_bill $1 $2 $3
```

Specifying Start and End Times

With the following billing utilities, you can specify a date range for account billing or payment collection dates:

- **pin_deposit** (billing date)
- **pin_collect** (payment collection date)
- **pin_cycle_fees** (billing date)

For example, you can specify to run **pin_deposit** on accounts whose billing date falls in a particular week. You typically use date parameters when you run the billing utilities manually, but you can include them in scripts.

The default billing scripts do not use date ranges. You might need to set the date range to rerun billing, or to increase performance by limiting billing activity.

The syntax for the start and end parameters is:

```
-start [ mm/dd/yy or yyyy | number_of_days ]
```

```
-end [ mm/dd/yy or yyyy | number_of_days ]
```

You can specify the exact start and end dates, or specify a number of days before the current date for the start and end time. Note that the end date is automatically the current date if you do not specify a value for the **-end** parameter.

The following examples run **pin_deposit** for accounts that have a billing date from March 20 through March 23 if the current date is March 24:

```
pin_deposit -start 03/20/01 -end 03/23/01
pin_deposit -start 4 -end 1
```

If a start date is specified, the entire day is included.

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.

Setting Start and End Dates for pin_collect

The `pin_collect` utility collects payments for 2 days: the day before the utility is run and the day on which the utility is run: when any of the following conditions are met:

- The **start** and **end** parameters are not set (the default).
- The **start** and **end** parameters are set to the same value.
- The **start** parameter is set to the current date, and the **end** parameter is not set.

To collect payments *only* on the day you run `pin_collect`, set the **start** parameter with a value of **0**. For example:

```
pin_collect -start 0
```

You can also specify exact start and end dates, and you can specify a number of days before the current date for the start and end time calculation. The `pin_collect` utility only processes bills with a collection date within the start and end date range.

Note: For open item accounting, the end date of the bill is not used to determine whether the bill falls within the specified range and qualifies for collection: only the start date is used.

Editing Billing Utility Configuration Files

All billing utilities share a common configuration file located in `BRM_Home/apps/pin_billd`.

- For information on billing defaults, see "[About the Billing Utilities](#)".
- For information about configuration files, see "Using Configuration Files to Connect and Configure Components" in *BRM System Administrator's Guide*.

Editing the Billing Scripts

To make changes to how the billing utilities run, you must edit the scripts with a text editor. Each script includes editing instructions.

For information about billing utilities, see "[About the Billing Utilities](#)". For information about billing utility syntax, see "[Running Billing Utilities Manually](#)".

Changing the Path in the Billing Scripts

The default billing scripts include a command that points to the directory that contains the billing utilities. To change the path for the billing utilities, edit the path entries 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}
```

Testing Billing

You can test billing by running the utilities without actually generating any bill.

For regular billing, use

```
pin_bill_accts -test -verbose
```

See "[pin_bill_accts](#)".

For corrective billing, use

```
pin_make_corrective_bills -validate_only
```

See "[pin_make_corrective_bills](#)".

You can also test credit card and direct debit processing. See "Testing Paymentech Credit Card Processing" in *BRM Configuring and Collecting Payments*.

Running Daily Billing

The `pin_bill_day` script performs most of the billing operations for regular bills. See "[About Running the Billing Scripts](#)".

Use a `cron` job with a `crontab` entry to run the `pin_bill_day` script. The following `crontab` entry runs `pin_bill_day` at 1:00 a.m. daily:

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

Running Weekly Billing

The `pin_bill_week` script runs the `pin_collect` utility with the `rebill` option to process outstanding bills from active credit card or direct debit accounts that could not be collected during regular daily billing runs. This enables you to collect overdue payments.

For example, the 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.

Caution: 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.

Use a `cron` job with a `crontab` entry to avoid conflicts with the `pin_bill_month` script. The following `crontab` entry runs `pin_bill_week` every Sunday at 12:05 a.m.

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

Running Monthly Billing

The `pin_bill_month` script runs the `pin_collect` utility with the `rebill` option to process outstanding bills from closed or inactive accounts.

Use a `cron` job with a `crontab` entry to run the `pin_bill_month` script. 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 &
```

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.

For information about troubleshooting BRM components, see "Resolving Problems in Your BRM System" in *BRM System Administrator's Guide*.

You handle billing failures differently depending on the billing utility that was affected, and how the failure occurred:

- [If the Billing Utility Was Interrupted](#)
- [If You Miss a Daily Billing Run](#)

If the Billing Utility Was Interrupted

If the **pin_collect** utility or the **pin_deposit** utility is interrupted 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 You Miss a Daily Billing Run

If the billing utilities were not run at all, for example, if the database was offline, you can 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** utility and **pin_inv_accts** utility 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.

Running Billing Utilities Manually

You can run regular billing utilities manually or using the **pin_bill_day**, **pin_bill_week**, or **pin_bill_month** billing scripts. However, you must run corrective billing utilities manually. When you do, ensure that you maintain the same order as they run in the billing scripts.

When you create your test database, it's a good idea to run the billing utilities manually, by using the **verbose** parameter. This enables you to see whether you get the expected results.

For information on billing utilities, see "[About the Billing Utilities](#)" and the following reference pages:

- **pin_bill_accts** for regular bills. Or, **pin_make_corrective_bills** for corrective bills
- **pin_collect**
- **pin_inv_accts**
- **pin_deposit**

- [pin_cycle_fees](#)

Note: It is not possible to run multiple copies of the same billing program simultaneously.

Monitoring Billing Activity

Since billing errors can have a negative impact on your business, you must be especially vigilant in checking for errors every time you run any of the BRM billing utilities. Check the log file for the billing utilities (**pin_bill.d.pinlog**) to quickly spot any errors.

Checking for Payment-Processing Errors

BRM keeps track of transactions with BRM-initiated payment processing services, such as Paymentech, and waits for a confirmation that each transaction is processed. You should check for transaction errors daily and resolve transaction failures. See "Resolving Failed BRM-Initiated Payment Transactions" in *BRM Configuring and Collecting Payments*.

Maintaining Transmission Logs for Billing Transactions

The **pin_collect** utility creates transmission log files to record the billing transactions sent to and received from Paymentech. The files for information sent have the prefix **fusas** (Paymentech), and the files for information received have the prefix **fusar** (Paymentech).

The Paymentech transmission log files are stored in the system temporary directory. If that directory is not defined or does not exist, BRM looks for a different folder, in the following order:

- The Directory defined by the *temp_dir* entry in the Paymentech DM configuration file (*BRM_Home/sys/dm_fusa/pin.conf*)
- **/var/tmp**
- **/tmp**

You must delete or archive billing transmission logs periodically to prevent the file system from overflowing. If data security is an issue, delete or archive the files to a secure location immediately after you run billing. Good business practice suggests archiving the files for at least 30 days before discarding them.

About Trial Billing

This chapter provides an overview of Oracle Communications Billing and Revenue Management (BRM) trial billing and explains when and how to run trial billing.

Before you implement trial billing, you should be familiar with:

- BRM billing. See ["About Billing"](#).
- BRM billing applications. See ["Billing Utilities"](#).

About Trial Billing

Trial billing is a process that simulates BRM billing. You use trial billing to validate billing results without running actual billing.

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 use trial invoices to validate customer charges before creating the final bills.

You can also collect revenue assurance data for trial bills and display the data by generating Revenue Assurance Billing reports. You use revenue assurance data to validate overall billing results.

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. Trial billing searches the BRM database for accounts with an expired billing date. For each account it finds, it computes the balance impacts and creates and stores a trial invoice in the BRM database.

For information on running trial billing, see ["Running Trial Billing"](#).

Comparing Billing and Trial Billing

[Table 14–1](#) summarizes the similarities and differences between regular billing and trial billing:

Table 14–1 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.

Table 14–1 (Cont.) Comparisons between Regular Billing and Trial Billing

Billing with pin_bill_accts	Trial Billing with pin_trial_bill_accts
Creates cycle forward and usage item objects in the BRM database.	Creates cycle forward and usage item objects, but the objects are not recorded in the BRM database.
Creates bill objects in the BRM database.	Creates bill 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.
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](#)".

How Trial Billing Works

When you run the "[pin_trial_bill_accts](#)" utility to perform trial billing, the PCM_OP_BILL_MAKE_TRIAL_BILL opcode is called to create the trial invoices and collect revenue assurance data for the trial billing run.

Note: To collect revenue assurance data for trial billing, you must enable the trial billing utility to generate revenue assurance data. See "Enabling Billing Utilities to Generate Revenue Assurance Data" in *BRM Collecting Revenue Assurance Data*.

PCM_OP_BILL_MAKE_TRIAL_BILL calls the PCM_OP_BILL_MAKE_BILL opcode to compute the balance impacts and the balance due for the accounts specified in the input flist. The input flist includes the following fields:

- The POID of the /account object for trial billing.
- The name of the program that called PCM_OP_BILL_MAKE_TRIAL_BILL.
- The start and end dates that specify the billing cycles for trial billing.
- LAST_BILL_STATE_TO_PROCESS with the value 2 to indicate that the /billinfo state is final.
- Two optional flag fields:
 - PIN_FLD_PREINVOICE_MODE specifies whether to generate trial invoices for the trial billing run. See "[Generating Trial Invoices](#)".

- PIN_FLD_CHECK_SPLIT_FLAGS specifies whether to split the revenue assurance data collected for trial billing into detailed categories. See "[Collecting Split Revenue Assurance Data](#)".

Note: Trial billing stops and reports a warning message when it encounters an account or bill unit (**/billinfo**) with inactive status.

Generating Trial Invoices

By default, PCM_OP_BILL_MAKE_TRIAL_BILL calls the PCM_OP_INV_MAKE_INVOICE opcode to create the trial invoice. This opcode stores the trial invoice in the primary database schema as an **/invoice** object (POID type **/invoice/trial**) and returns an array of trial invoice POIDs for the invoices that were created for the account specified in the input flist. PCM_OP_BILL_MAKE_TRIAL_BILL opens a separate transaction to create the **/invoice/trial** objects.

If the PIN_FLD_PREINVOICE_MODE field is present in the input flist and has a value of **1**, PCM_OP_BILL_MAKE_TRIAL_BILL does not call the invoicing opcode to create trial invoices and only revenue assurance data is generated for the account. If the PIN_FLD_PREINVOICE_MODE field is not present in the input flist or has a value of **0**, trial billing creates trial invoices.

Note: If a start date is not provided in the input flist, PCM_OP_BILL_MAKE_TRIAL_BILL performs trial billing for all billing cycles completed before the end date. For accounts with skipped billing cycles, more than one trial invoice might be created.

Collecting Split Revenue Assurance Data

If the PIN_FLD_CHECK_SPLIT_FLAGS field is present in the input flist and has a value of **1**, PCM_OP_BILL_MAKE_TRIAL_BILL passes this flag to PCM_OP_BILL_MAKE_BILL, which returns amounts associated with each item type and service type combination so that the revenue assurance data collected for trial billing can be split into detailed categories. The item and service type details are returned in the PIN_FLD_REVENUES array in the output flist, along with the total number of subscription services associated with the account. If any bills were suppressed, the amount suppressed and the suppression reason are also returned.

If PIN_FLD_CHECK_SPLIT_FLAGS has a value of **0** or is not present in the input flist, PCM_OP_BILL_MAKE_TRIAL_BILL does not return item type and service type details.

Split revenue assurance data can be viewed by generating a Revenue Assurance Billing Detail report.

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.

Figure 14–1 shows a trial invoice displayed in the Sample Invoice Viewer using the default invoice template:

Figure 14–1 Trial Invoice

Software Inc.
100 De Anza Boulevard

Cupertino, CA 95014

John M. Wells
10573 John Way
Cupertino, CA 95014
USA

Bill Date	Invoice Number	Account Number	Payment Due	Amount Due	Amount Enclosed
Mar 01 2002	TRIAL-1	0.0.0.1-14158	Mar 31 2002	40.85	\$

Account Summary	
Balances	\$ Total
Previous Balance:	25.90
Subordinate Accounts:	0.00
Current Balance:	14.95
Total Balance Due:	40.85

Item Summary		
Item No.	Description	\$ Total
Accounts/Receivable Items		
TRIAL-1,3	Usage	0.00
TRIAL-1,2	Cycle forward	9.95
TRIAL-1,1	Cycle forward	3.00

Event Details			
Date	Description	Rate Description	\$ Total
Mar 01 2002	Cycle Forward Fees (acct) (srvc):	\$9.95 a month	9.95
	Product 1a - Internet Access		
Mar 01 2002	Cycle Forward Fees (acct) (srvc):	\$3 a month	3.00
	Product 1b - Email Account		

Note: 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.

For more information about invoice templates, see "Designing Invoice Templates" in *BRM Designing and Generating Invoices*.

For more information about customizing invoice information, see "Customizing the Information Included in Invoices" in *BRM Designing and Generating Invoices*.

You can also purge and export trial invoices. See "[Purging Trial Invoices](#)" and "[Exporting Trial Invoices](#)".

How Trial Invoices Are Stored

Trial invoices are stored in the BRM database in the same formats as regular invoices. By default, trial invoices are stored in **pin_flist** format. For more information about invoice storage formats, see "About Formats for Storing Invoices" in *BRM Designing and Generating Invoices*.

Trial invoices and regular invoices are different in the following ways:

- Trial invoices and regular invoices are stored in the **/invoice** object with different POID types. Regular invoices are stored with the POID type **/invoice**. Trial invoices are stored with the POID type **/invoice/trial**, a subclass of the **/invoice** base class. If you have a custom **/invoice** class: for example, **/invoice/custom_subclass**: BRM stores trial invoices with the POID type **/invoice/custom_class/trial**.
- Trial invoices are *always* stored in the primary BRM database schema. In a multischema system, regular invoices can be stored in a separate schema.

Using Trial Invoices for Validating Billing Charges

The following examples show how to use trial invoices for validating billing charges:

- **Verifying a new price plan**

You can use trial billing to verify that a new price plan is loaded in the system and set up correctly. For example, you can designate a few accounts as test accounts for specified products. Run trial billing on the billing date for these accounts by specifying their account POIDs. The trial invoices that are generated for these accounts can then be reviewed by your customer service representatives or billing operations personnel to verify that charges have been applied correctly based on a predefined output.

In case rate fees such as usage charges from an obsolete price plan have been applied to *real* accounts, run rerating to make new rates effective retroactively, before your billing run.

- **Verifying miscellaneous charges, such as refunds**

If your company issues a mass refund to subscribers, you can run trial billing at any time during the billing cycle to verify that the refunds were applied correctly. Select a few accounts and run trial billing by specifying their account POIDs.

You can perform similar verifications for taxes, late fees, or finance charges; for example, when a new tax rate is introduced, when a late fee changes from \$20 to

\$25, or when the finance rate changes from 6% to 7%. You can run trial billing to verify that the new rates are being applied.

Review the trial invoices. If any discrepancies are found, you can make corrections, such as changing the price plan, making billing adjustments, or running rerating. Certify your results by rerunning trial billing before sending final invoices to your subscribers.

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`, where `BRM_Home` is the directory where you installed BRM components). 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.

Note: The `-split` parameter for `pin_trial_bill_accts` is valid only if Revenue Assurance Manager is installed.

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*.

For more information about split revenue assurance data, see "About Splitting Revenue Assurance Data into Detailed Categories" in *BRM Collecting Revenue Assurance Data*.

For information about Revenue Assurance Manager reports, see "Revenue Assurance Manager Reports" in *BRM Collecting Revenue Assurance Data*.

For information about generating reports, see "Running BRM Reports" in *BRM Reports*.

Running Trial Billing

To run trial billing, you run the `pin_trial_bill_accts` utility. It connects to the default database schema specified in the trial billing configuration file (`pin.conf`) located in the `BRM_Home/apps/pin_trial_bill` directory.

For a description of the syntax and parameters for `pin_trial_bill_accts`, see "[pin_trial_bill_accts](#)".

Important: If you use the `pin_bill_day` script to run the `pin_bill_accts` utility, you must run `pin_trial_bill_accts` *before* running `pin_bill_day`.

When you run trial billing, you can specify search criteria such as the billing cycles, the status of the accounts, and the list of accounts and bill units for which to create trial bills. `pin_trial_bill_accts` creates trial bills for all accounts and bill units that meet the search criteria you specify. If you do not specify any search criteria, `pin_trial_bill_accts` generates trial bills for all billing cycles that have not been billed and are complete at the time you run the utility.

You can configure `pin_trial_bill_accts` to select accounts randomly or you can provide a list of accounts and bill units in an input file. See ["Specifying Accounts for Trial Billing"](#).

You can provide a list of accounting days of month (DOMs) and billing segments in an input file to select bill units with the specified DOMs or segments. You can also use this input file to provide a list of specific accounts and bill units. See ["Specifying Bill Units, Billing Segments, and DOMs for Trial Billing"](#).

If the revenue assurance data collected from trial billing is sufficient for validating your billing charges, you can run the trial billing utility without creating trial invoices. This improves the performance of trial billing. See ["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. The flist must use the following structure, where each array element corresponds to one account and bill unit:

```
0 PIN_FLD_RESULTS          ARRAY [0] allocated 1, used 1
1   PIN_FLD_ACCOUNT_OBJ    POID [0] 0.0.0.1 /account 9605
1   PIN_FLD_POID           POID [0] 0.0.0.1 /billinfo 2342
0 PIN_FLD_RESULTS          ARRAY [1] allocated 1, used 1
1   PIN_FLD_ACCOUNT_OBJ    POID [0] 0.0.0.1 /account 9765
1   PIN_FLD_POID           POID [0] 0.0.0.1 /billinfo 2570
0 PIN_FLD_RESULTS          ARRAY [2] allocated 1, used 1
1   PIN_FLD_ACCOUNT_OBJ    POID [0] 0.0.0.1 /account 11973
1   PIN_FLD_POID           POID [0] 0.0.0.1 /billinfo 3987
```

Note:

- Trial billing stops and reports a warning message when it encounters an account or bill unit (**/billinfo**) with inactive status.
 - All accounts listed in the input file will be trial billed.
 - If a selected account has a subordinate bill unit, **pin_trial_bill_accts** does not create a trial bill for it. Subordinate 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_bill/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 ["Splitting a Billing Run into Multiple Runs"](#).)
-

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_bill* 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_bill/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.

Note:

- If *filename* is in the same directory from which you run `pin_trial_bill_accts`, specify only the file name.
 - If *filename* is in a different directory from which you run `pin_trial_bill_accts`, you must 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 ID of the account, and add a **Billinfo** child element that specifies the POID ID of the bill unit.

Important:

- Trial billing stops and reports a warning message when it encounters an account or bill unit (**billinfo**) 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 ID 64295 that belongs to the account with POID ID 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 ID 17763 and one trial bill for the account with POID ID 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 "[About Billing Segments](#)".

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.

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.

Hierarchical and Sponsor Groups for Trial Billing

Hierarchical and sponsorship groups enable customers to pay other customers bills. Hierarchical accounts form a parent-child relationship, and parent and child accounts are on the same billing cycle. Trial billing for hierarchical accounts is handled differently than regular billing.

In regular billing, charges for the subordinate accounts are calculated first and then rolled up to the parent account. For each subordinate account, BRM creates a database lock, computes the balance impacts, and then releases the lock.

In trial billing, the parent and all its subordinate accounts are locked simultaneously. The parent account as well as the subordinate accounts remain locked until all the subordinate account balance impacts have been computed and rolled up to the parent account.

Note: As with regular billing, the invoicing threshold value for hierarchical groups is considered during trial billing. If the threshold value is exceeded: that is, the number of subordinate bill units is greater than the threshold value: a separate invoice is generated for each subordinate bill unit by using multiple threads, and the invoice for the accounts receivable account's invoice will not contain the data. For more information, see "About Invoicing for Hierarchical Account Groups" in *BRM Designing and Generating Invoices*.

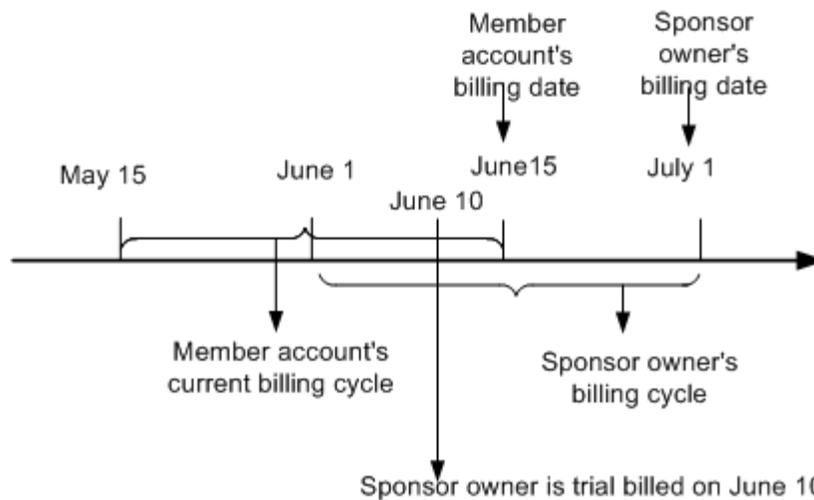
Important: When hierarchical accounts are selected for trial billing, access to the parent and child accounts is prevented and system performance is decreased as a result of the database locks.

In contrast, sponsorship occurs when an account is added to a sponsored group and a sponsored product is added to that account. The sponsored group account owner and the member account can have different billing cycles. When a sponsored group owner is selected for trial billing, the trial invoice may not contain all the fees from member accounts.

In regular billing, when a sponsored group owner account is billed, BRM processes the member accounts to determine the sponsored fees to apply to the sponsor owner. The sponsored cycle events of member accounts are accumulated in the `/item/sponsor` object of the sponsor owner account and appear on the sponsor owner's bill.

In trial billing, when a sponsor owner account is selected, trial billing is not processed for member accounts. A sponsor owner can be processed for trial billing without creating trial bills for member accounts. In this case, the sponsor owner's trial invoice may not contain the sponsored system events of the member accounts.

In the example shown in [Figure 14-2](#), the trial invoice created for the sponsor owner account does not contain the sponsored events (cycle events) of the member account:

Figure 14–2 Included Charges in a Sponsor Owner Trial Invoice

For more information about hierarchical and sponsored group accounts, see "About Account Groups" in *BRM System Administrator's Guide*.

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.

Caution: 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*.

Viewing Trial Invoices

You can view trial invoices in the following ways:

- Use the Sample Invoice Viewer to view invoices stored in the database.

- Use a Web browser to view invoices in HTML format.

The Sample Invoice Viewer is shipped with BRM and can be found in the *BRM_Home/source/apps/sampleviewer* directory. Follow the instructions in the Sample Invoice Viewer **readme** file on how to configure and run the Sample Invoice Viewer to view trial invoices.

You can also use the **pin_inv_export** utility to export trial invoices in HTML format and view them in your Web browser. For more information, see "[Exporting Trial Invoices](#)".

Note: You cannot view trial invoices by using the Invoice Viewer in Customer Center.

Improving Trial Billing Performance

The trial billing application is a multithreaded application that uses a set of configuration entries similar to the **pin_bill_accts** billing application.

Important: 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*.

For information on how to configure these entries for optimum performance, see "Tuning Billing Performance" in *BRM System Administrator's Guide*.

For information about multithreaded application, see "Configuring Your Multithreaded Application" in *BRM Developer's Guide*.

Part IV

Billing Utilities

Part IV provides reference information for Oracle Communications Billing and Revenue Management (BRM) billing utilities. It contains the following chapter:

- [Billing Utilities](#)

15

Billing Utilities

This chapter provides reference information for Oracle Communications Billing and Revenue Management (BRM) billing utilities.

load_config_item_tags

Use this utility to load item tags into the BRM database.

You define item tags in the *BRM_Home/sys/data/pricing/example/config_item_tags.xml* file or another file that uses the same format. (*BRM_Home* is the directory where you installed BRM components.) The format of the XML file is specified in the *config_item_tags.xsd* schema file in the *BRM_Home/xsd* directory.

By default, this utility loads item tags into your system's default item tag configuration (*/config/item_tags* object whose *PIN_FLD_NAME* value is **Default**).

If the *config_item_tags.xml* file contains a value in its optional Name tag other than **Default**, this utility does one of the following:

- If no other */config/item_tags* object has the same name, the utility creates a new */config/item_tags* object and sets the *PIN_FLD_NAME* field of the new object to the value specified in the Name tag.
- If another */config/item_tags* object with the same name already exists, the utility replaces that object.

For information about item tags, see "[Creating Custom Bill Items.](#)"

Note: You cannot load separate */config/item_tags* objects for each brand. All brands use the same objects.

Caution: When you run *load_config_item_tags*, it overwrites the existing item tags in the */config/item_tags* object whose *PIN_FLD_NAME* value matches the Name value in the *config_item_tags.xml* file. If the XML file does not contain a *PIN_FLD_NAME* value, the Default item tag configuration is replaced.

If you are updating a set of item tags, you cannot load the new item tags only. You must load the entire set of tags in the *config_item_tags.xml* file.

Important: To connect to the BRM database, the *load_config_item_tags* utility requires a configuration file. See "Creating Configuration Files for BRM Utilities" in *BRM System Administrator's Guide*.

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. 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.

You define these items in the *BRM_Home/sys/data/pricing/example/config_item_types.xml* file, or another file that uses the same format. The format of the XML file is specified in the *config_item_types.xsd* schema file in the *BRM_Home/xsd* directory.

By default, this utility loads mappings into your system's default item type configuration (*/config/item_types* object whose *PIN_FLD_NAME* value is **Default**).

If the *config_item_types.xml* file contains a value in its optional *Name* tag other than **Default**, this utility does one of the following:

- If no other */config/item_types* object has the same name, the utility creates a new */config/item_types* object and sets the *PIN_FLD_NAME* field of the new object to the value specified in the *Name* tag.
- If another */config/item_types* object with the same name already exists, the utility replaces that object.

For information about item types, see "[Creating Custom Bill Items](#)."

Caution: When you run **load_config_item_types**, it overwrites the existing mappings in the */config/item_types* object whose *PIN_FLD_NAME* value matches the *Name* value in the *config_item_types.xml* file. If the XML file does not contain a *PIN_FLD_NAME* value, the Default item type configuration is replaced.

If you are updating a set of mappings, you cannot load the new mappings only. You must load complete sets of mappings each time you run the **load_config_item_types** utility.

When you run **load_config_item_tags**, it overwrites the existing item tags in the */config/item_tags* object whose *PIN_FLD_NAME* value matches the *Name* value in the *config_item_tags.xml* file. If the XML file does not contain a *PIN_FLD_NAME* value, the Default item tag configuration is replaced.

If you are updating a set of item tags, you cannot load the new item tags only. You must load the entire set of tags in the *config_item_tags.xml* file.

Note: To connect to the BRM database, the **load_config_item_types** utility requires a configuration file. See "Creating Configuration Files for BRM Utilities" in *BRM System Administrator's Guide*.

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. 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 mappings. 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 `/config/suppression` object in the BRM database.

You enter this information in the bill suppression configuration file (`BRM_Home/sys/data/config/pin_bill_suppression.xml`).

For more information, see the following topics:

- [Associating Bill Suppression Information with Customer Segments](#)
- [Editing the Bill Suppression Configuration File](#)

Note: You cannot load separate `/config/suppression` objects for each brand. All brands use the same object.

Caution: This utility overwrites all existing data in the `/config/suppression` object. When updating the data, you cannot load new data only. You must load bill suppression data for each customer segment every time you run the utility.

Important: To connect to the BRM database, this utility needs a configuration file in the directory from which you run the utility. For information about creating configuration files for BRM utilities, see "Creating Configuration Files for BRM Utilities" in *BRM System Administrator's Guide*.

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, names 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 copy filename to the same directory from which you run the load utility, specify only the file name. 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*.

Important: You must stop and restart the Connection Manager (CM) to make new bill suppression data available. See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

load_pin_billing_segment

Use this utility to load billing segment definitions into the `/config/billing_segment` object in the BRM database.

You define billing segments in the billing segment configuration file (`BRM_Home/sys/data/config/pin_billing_segment.xml`).

For more information, see the following topics:

- [About billing segments](#)
- [Editing the Billing Segment Configuration File](#)
- [Setting Up Billing Segments](#)

Note: You cannot load separate `/config/billing_segment` objects for each brand. All brands use the same object.

Caution: This utility overwrites all existing data in the `/config/billing_segment` object. When updating the data, you cannot load new data only. You must load bill suppression data for each customer segment every time you run the utility.

Important: To connect to the BRM database, this utility needs a configuration file in the directory from which you run the utility. For information about creating configuration files for BRM utilities, see "Creating Configuration Files for BRM Utilities" in *BRM System Administrator's Guide*.

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, names 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 copy filename to the same directory from which you run the load utility, specify only the file name. 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 `/config/calendar` objects in the BRM database.

You configure billing calendars in the billing calendar configuration file (`BRM_Home/sys/data/config/pin_calendar.xml`). See ["Editing the Billing Calendar Configuration File"](#).

Note: You cannot load separate `/config/calendar` objects for each brand. All brands use the same object.

Caution: This utility overwrites all existing data in the `/config/calendar` objects. When updating billing calendars, you cannot load new calendars only. You must load data for each calendar every time you run the utility.

Important: To connect to the BRM database, this utility needs a configuration file in the directory from which you run the utility. For information about creating configuration files for BRM utilities, see "Creating Configuration Files for BRM Utilities" in *BRM System Administrator's Guide*.

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 copy filename to the same directory from which you run the load utility, specify only the file name. 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.

To verify that the billing calendars were loaded, display the **/config/calendar** objects 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*.

Important: You must stop and restart the Connection Manager (CM) to make new billing calendar data available. See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

load_pin_payment_term

Use this utility to load payment terms into the `/config/payment_term` object in the BRM database.

You configure payment terms in the payment terms configuration file (*BRM_Home/sys/data/config/pin_payment_term.xml*). See ["Editing the Payment Terms Configuration File"](#).

Note: You cannot load separate `/config/payment_term` objects for each brand. All brands use the same object.

Caution: This utility overwrites all existing data in the `/config/payment_term` object. When updating the data, you cannot load new data only. You must load data for each payment term every time you run the utility.

Important: To connect to the BRM database, this utility needs a configuration file in the directory from which you run the utility. For information about creating configuration files for BRM utilities, see ["Creating Configuration Files for BRM Utilities"](#) in *BRM System Administrator's Guide*.

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 (see ["Validating Your Payment Terms Configuration File Edits"](#)). 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. See "[Validating Your Payment Terms Configuration File Edits](#)".

If you copy filename to the same directory from which you run the load utility, specify only the file name. 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.

To verify that the payment terms were loaded, display the `/config/payment_term` 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*.

Important: You must stop and restart the Connection Manager (CM) to make new payment term data available. See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

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.

For information about remittance, see ["Remitting Funds to Third Parties"](#) and ["About Customizing Remittance"](#).

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.

Note: To connect to the BRM database, the `load_pin_remittance_flds` utility requires a configuration file. See "Creating Configuration Files for BRM Utilities" in *BRM System Administrator's Guide*.

Location

BRM_Home/bin

Syntax

```
load_pin_remittance_flds [-d] [-t] [-v] [-h] remittance_flds
```

Parameters

remittance_flds

The name and location of the remittance fields file. By default, this file is *BRM_Home/sys/data/pricing/example/pin_remittance_flds*.

-d

Writes additional information for debugging purposes to the utility log file **default.pinlog**. This file is located either in the directory from which the utility was started, or in a directory specified in the configuration file.

-t

Runs in test mode. In this mode, `load_pin_remittance_flds` 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_fds any_other_parameter **-v** > *filename.log*

-help

Displays the syntax and parameters for this utility.

Results

The **load_pin_remittance_fds** utility notifies you only if it encounters errors. Use the **-v** option to display additional status information.

If this utility encounters errors, look in the log file (**default.pinlog**) for error messages. 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_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 define remittance specifications in the *BRM_Home/sys/data/pricing/example/pin_remittance_spec* file, or another file that uses the same format. You must load this file to use the BRM remittance feature.

load_pin_remittance_spec checks the validity of products in the remittance specification.

Caution: When you run **load_pin_remittance_spec**, it overwrites remittance specifications currently in the BRM database.

If you are updating remittance information, you cannot load only new remittance specifications. Ensure that you load your complete specification file, including specifications that have not changed.

Note:

- Ensure that you load the remittance fields file before you load the remittance specification file. See "[load_pin_remittance_flds](#)".
 - To connect to the BRM database, the **load_pin_remittance_spec** utility requires a configuration file. See "Creating Configuration Files for BRM Utilities" in *BRM System Administrator's Guide*.
-
-

For information about remittance, see "[Remitting Funds to Third Parties](#)".

Location

BRM_Home/bin

Syntax

```
load_pin_remittance_spec [-d] [-t] [-v] [-h] remittance_spec
```

Parameters

remittance_spec

The name and location of the remittance specification file. By default, this file is *BRM_Home/sys/data/pricing/example/pin_remittance_spec*.

-d

Writes error information for debugging purposes to the utility log file **default.pinlog**. This file is located either in the directory from which the utility was started, or in a directory specified in the configuration file.

-t

Runs in test mode. In this mode, **load_pin_remittance_spec** 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: 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_spec any_other_parameter **-v** > *filename.log*

-help

Displays the syntax and parameters for this utility.

Results

The **load_pin_remittance_spec** utility notifies you only if it encounters errors. Use the **-v** option to display additional status information.

If this utility encounters errors, look in the log file **default.pinlog** for error messages. The log file is either in the directory from which the utility was started, or in a directory specified in the configuration file.

Important: You must stop and restart the Connection Manager (CM) to make the new specification available. See "Starting and Stopping the BRM System" in *BRM System Administrator's Guide*.

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. This utility should be used for regular billing only. For corrective bills, use "[pin_make_corrective_bills](#)".

For information about billing, see:

- Billing accounts with the pin_bill_accts utility
- About billing

Note: To connect to the BRM database, the **pin_bill_accts** utility needs a configuration file in the directory from which you run the utility. See "Creating Configuration Files for BRM Utilities" in *BRM System Administrator's Guide*.

Location

BRM_Home/bin

Syntax

```
pin_bill_accts [-active|-close|-inactive]
               [-pay_type ID]
               [-end mm/dd/yy|number_of_days]
               [-remit include|exclude|only]
               [-sponsorship|-discount]
               [-retry]
               [-cycle_charge_only] [-finalize_bill]
               [-from_file file_name]
               [-file billing_run_config_file_name]
               [-split]
               [-test]
               [-verbose]
               [-help]
```

Parameters

-active|-close|-inactive

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 15–1](#).

Table 15–1 Payment Method ID

Payment Method	ID
credit card	10003
debit card	10002
direct debit	10005
guest	10010

Table 15-1 (Cont.) Payment Method ID

Payment Method	ID
invoice	10001
prepaid	10000
SEPA	10018
subordinate	10007
undefined	0

Important: 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
```

-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 01/31/01** includes accounts with a due date on or before January 31, 2001) 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 include/exclude/only

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 sponsor groups will be billed. When you specify this option, **pin_bill_accts** will do the following depending on the *billing_flow_sponsorship* parameter in */config/business_params*:

If the value of *billing_flow_sponsorship* is:

- **0**, it bills sponsor and sponsoree accounts in any order.

- 1, it bills sponsor accounts before sponsorees.
- 2, it bills sponsoree accounts before sponsors.

Before using this option, ensure that sponsorship billing is enabled. See "Setting up billing for sponsorship".

Note: You cannot use this parameter with the **-discount** parameter.

-discount

Specifies how discount sponsor groups will be billed. When you specify this option, **pin_bill_accts** will do the following depending on the *billing_flow_discount* parameter in */config/business_params*:

If the value *billing_flow_discount* is:

- 0, it bills discount group owner and discount group member accounts in any order.
- 1, it bills discount group owner accounts before discount group member accounts.
- 2, it bills discount group member accounts before discount group owner accounts.

Before using this option, ensure that sponsorship billing is enabled. See "Setting up billing for sponsorship".

Note: You cannot use this parameter with the **-sponsorship** parameter.

-cycle_charge_only

If best pricing is configured, runs billing on accounts with billinfo states 0 and 1. At the end of billing, the billinfo state advances to the next state. See "About Bill States" in *BRM Managing Accounts Receivable*.

For information on best pricing, see "[Offering the Best Price to Your Customers](#)".

-finalize_bill

If best pricing is configured, runs billing on accounts with billinfo state 2. At the end of billing, the billinfo state is set back to 0. See "About Bill States" in *BRM Managing Accounts Receivable*.

For information on best pricing, see "[Offering the Best Price to Your Customers](#)".

-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 the **pay_type 10007** with the **-retry** option to bill the failed subordinate accounts 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
```

-from_file file_name

Specifies that the accounts to bill will be read from a file, and the associated file name. This parameter is used when you set up pipeline-triggered billing. The input file is

passed to the **pin_bill_accts** utility by the BillHandler billing batch handler to trigger billing for the accounts specified in the file. See "Setting up pipeline-triggered billing".

-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_bill* 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). For more information, see the following topics:

- Splitting a billing run into multiple runs
- Specifying due date adjustments in a billing run
- [Billing Only Specified Accounts and Bill Units](#)

Note: When running **pin_bill_accts** with **-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 to Run in a Multischema Environment](#)".

Caution: 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.

-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 has not been completed for the **/billinfo** object, **pin_bill_accts** aborts with an error and sets the **PIN_FLD_BILLING_STATUS** field of the **/billinfo** object to **PIN_BILL_ERROR**. In addition, it also sets the **PIN_FLD_BILLING_STATUS_FLAGS** field of the **/billinfo** object to **PIN_BILL_FLAGS_CF_NOT_APPLIED** (bit value 0x1000).

Important: If any subordinate 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 subordinate and parent bill units.

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 products that have an expired pending cancellation date. For example, if a product is set to cancel at a future date, the **pin_cycle_fees** utility cancels the product.
- Bill products with a delayed purchase start time.

For more information about fees, see "[Prorating Cycle-Forward Fees and Canceling Products with the pin_cycle_fees Utility](#)".

Note: To connect to the BRM database, the **pin_cycle_fees** utility needs a configuration file in the directory from which you run the utility. See "Creating Configuration Files for BRM Utilities" in *BRM System Administrator's Guide*.

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]
```

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 products.

-defer_purchase

Bills products 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. For information on using start and end parameters, see "[Specifying Start and End Times](#)".

-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 products.

-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.

Results

If the **pin_cycle_fees** 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_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)

Important: If any subordinate 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 subordinate 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 aborts with an error because the cycle fee processing has 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_cycle_forward

Use this utility to charge cycle forward fees.

Note: The **-regular_cycle_fees** parameter of the **pin_cycle_fees** utility replaces the functionality of the **pin_cycle_forward** utility.

Note: To connect to the BRM database, the **pin_cycle_forward** utility needs a configuration file in the directory from which you run the utility. See "Creating Configuration Files for BRM Utilities" in *BRM System Administrator's Guide*.

Location

BRM_Home/bin

Syntax

```
pin_cycle_forward [-verbose] [-test] [-help]
```

Parameters

-verbose

Displays information about successful or failed processing as the utility runs.

Note: 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_forward -verbose > filename.log
```

-test

Runs in test mode to find the accounts that meet the criteria for cycle forward fee, but does not apply any cycle forward fees. The test does not affect the resource balances (currency and non-currency) of the accounts.

-help

Displays the syntax and parameters for this utility.

Results

If the **pin_cycle_fees** 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_deferred_act

Use this utility as part of your daily billing to execute deferred actions. For example, if a CSR has scheduled an account to become inactive, the **pin_deferred_act** utility performs the status change on the scheduled date. By default, this utility is included in the **pin_bill_day** script.

For more information, see ["Executing Deferred Actions with the pin_deferred_act Utility"](#).

For information about scheduling status changes, see "Managing Deferred Actions" in *BRM Managing Customers*.

Note: To connect to the BRM database, the **pin_deferred_act** utility needs a configuration file in the directory from which you run the utility. See "Creating Configuration Files for BRM Utilities" in *BRM System Administrator's Guide*.

Location

BRM_Home/bin

Syntax

```
pin_deferred_act [-report|-purge|-retry] [-opcode opcode_name]
                 [-status pending|done|error] [-start mm/dd/yy]
                 [-end mm/dd/yy] [-verbose] [-test] [-help]
```

Parameters

-report

Displays the progress and current state of **/schedule** objects.

This parameter can take one or more of these options as search criteria to filter the results of your report:

- **-opcode** *[opcode_name]*
- **-status** **pending** | **done** | **error**
- **-start** *mm/dd/yy* or *yyyy*
- **-end** *mm/dd/yy* or *yyyy*

For example:

```
pin_deferred_act -report -start 01/10/03 -end 01/24/03 -verbose
```

-purge

Purges from the BRM database all **/schedule** objects whose actions have been executed successfully. This helps reduce the size of your database.

This parameter can take one or more of these options as search criteria for purging:

- **-opcode** *[opcode_name]*
- **-status** **pending** | **done** | **error**
- **-start** *mm/dd/yy* or *yyyy*

- **-end** *mm/dd/yy* or *yyyy*

For example:

```
pin_deferred_act -purge -start 01/10/03 -end 01/24/03 -verbose -opcode PCM_OP_
BILL_MAKE_BILL_NOW
```

-retry

Retries all the **/schedule** objects whose schedule actions have failed to execute and whose status is marked as **ERROR**.

This parameter can take one or more of these options as search criteria for purging:

- **-opcode** [*opcode_name*]
- **-status** **pending** | **done** | **error**
- **-start** *mm/dd/yy* or *yyyy*
- **-end** *mm/dd/yy* or *yyyy*

For example

```
pin_deferred_act -retry -start 01/10/03 -end 01/24/03 -verbose
```

-opcode [*opcode_name*]

Used as search criteria by the **-report**, **-purge**, and **-retry** parameters for retrieving **/schedule** objects containing the specified opcode responsible for the deferred action.

For example:

```
pin_deferred_act -report -start 01/10/03 -end 01/24/03 -verbose -opcode PCM_OP_
BILL_MAKE_BILL_NOW
```

-status pending|done|error

Used as search criteria by the **-report**, **-purge**, and **-retry** parameters for retrieving **/schedule** objects having the specified status.

For example:

```
pin_deferred_act -retry -start 01/10/03 -end 01/24/03 -verbose -status ERROR
```

-start *mm/dd/yy* or *yyyy*

-end *mm/dd/yy* or *yyyy*

Used as search criteria by the **-report**, **-purge**, and **-retry** parameters for retrieving **/schedule** objects with an execution date matching the **start** and **end** dates specified. The value you supply for the **start** date is inclusive, but the value for the **end** date is non-inclusive and also defaults to the current date. If a **start** date is not specified, this utility retrieves all valid **/schedule** objects up to the specified **end** date. If an **end** date is not specified, this utility uses the current date as the end date and retrieves all valid **/schedule** objects until the current date.

For information on using this parameter, see ["Specifying Start and End Times"](#).

-verbose | **-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:

```
pin_deferred_act any_other_parameter -v > filename.log
```

-test

Runs a test to find out how many accounts meet the criteria without performing the action. The test has no effect on the accounts. This is most useful when run with the **-verbose** option.

-help | -h

Displays the syntax and parameters for this utility.

Results

If the 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_deferred_act** utility logs error information in the **pin_mta.pinlog** file.

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. This utility is a multithreaded application which runs the **PCM_OP_BILL_MAKE_CORRECTIVE_BILL** opcode in each working thread.

This utility should be used for corrective billing only. For regular bills, use "[pin_bill_accts](#)".

To connect to the BRM database, the **pin_make_corrective_bills** utility uses the configuration file you use to run the **pin_bill_accts** utility. The default file is **pin_bill_run_control.xml** in the *BRM_Home/apps/pin_billd* directory. For more information, see the description on creating configuration files for BRM utilities in *BRM System Administrator's Guide*.

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/xxx]
    [-adj_reason D R]
    [-threshold_amount amount]
    [-correction_reason D R]
    [-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_bills** 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 associated 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 detail 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

This BRM command-line utility calculates 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.

For information on billing, see "[About Running the Billing Scripts](#)".

For information about remittance, see "[Remitting Funds to Third Parties](#)".

Note: To connect to the BRM database, the **pin_remittance** 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_remit*. See "Creating Configuration Files for BRM Utilities" in *BRM System Administrator's Guide*.

Location

BRM_Home/bin

Syntax

```
pin_remittance [-acct account_number] [-end date] [-output file_name]
               [-calonly] [-verbose] [-billing_cycle_alignment]
               [-help]
```

Tip: You can use just the first character of each parameter when you run **pin_remittance**. For example, you can enter either **pin_remittance -help** or **pin_remittance -h**.

Parameters

-acct account_number

Calculates 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.

For the default account number format, see the Glossary entry for account number.

-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 02/13/2001, **pin_remittance** includes all remittance events that took place through midnight of 02/12/2001.

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 remittance without writing the results to the BRM database. Use this parameter for testing and verification.

-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_remittance any_other_parameter **-v** > *filename.log*

-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.

-help

Displays the syntax and parameters for this utility.

pin_rollover

The **pin_bill_day** script runs this BRM utility to roll over all expired resource sub-balances that have not been rolled over.

For more information, see "When Rollover Events Occur" in *BRM Setting Up Pricing and Rating*.

Note: To connect to the BRM database, the **pin_rollover** utility needs a configuration file in the directory from which you run the utility. See "Creating Configuration Files for BRM Utilities" in *BRM System Administrator's Guide*.

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 roll over. The test does not affect the resource balances (currency and non-currency) 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_trial_bill_accts

Use this utility to calculate the balance due and create a *trial* invoice for each account.

Important: Trial billing stops and reports a warning message when it encounters an account or bill unit (**/billinfo**) 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 "Creating Configuration Files for BRM Utilities" in *BRM System Administrator's Guide*.

Important: 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]
```

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.

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
```

-f inputFile

Specifies the text file that contains the list of account POIDs for trial billing.

Example:

Create trial invoices for accounts in *inputFile* and whose billing date is less than **4/1/2002** (current date is 3/15/2002):

```
pin_trial_bill_accts -end 04/01/2002 -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_bill* directory.

This file can contain a list of account and bill unit POIDs or a list of billing segments, accounting days of the month (DOMs), or both.

For more information, see "[Specifying Bill Units, Billing Segments, and DOMs for Trial Billing](#)".

Note: You should not specify bill segments or DOMs along with bill units (**/billinfo** object POIDs) in the input file. If bill segments or DOMs are specified along with bill units, the **pin_trial_bill_accts** utility considers only the bill units for trial billing.

-activel-inactivel-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 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 15–2](#).

Table 15–2 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 accounts, you must run the subordinate accounts before running the parent account. For example, to generate trial invoices for hierarchical accounts whose payment method is by check, use the following syntax:

```
pin_trial_bill_accts -pay_type 10007
pin_trial_bill_accts -pay_type 10012
```

Hierarchical Account Limitations:

- The **-active**, **-closed**, or **-inactive** parameters are not supported for subordinate accounts when combined with the **-pay_type** parameter. These parameters limit the amount of subordinates that are invoiced, which leads to errors when the parent account is trial billed.
- Do not set a threshold value. Setting a threshold value limits the amount of subordinates that are invoiced, which leads to errors when the parent account 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 accounts when used with the **-pay_type** parameter.

For more information on payment methods, see "About Payment Methods" in *BRM Configuring and Collecting Payments*.

For more information on creating hierarchical account trial invoices, see the discussion on creating hierarchical account trial invoices in *BRM Designing and Generating Invoices*.

-retry

Runs trial billing for hierarchical accounts that were previously not billed during a trial invoicing run due to some error. After the errors have been resolved, use this parameter to rebill the hierarchical accounts that were previously not billed.

You must run the subordinate accounts before running the parent account. You must also regenerate trial invoices on the parent accounts that did not contain an error. For example, to run trial billing on accounts that contained an error and whose payment method is by 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
```

For more information, see the discussion on working with hierarchical account trial billing errors in *BRM Designing and Generating Invoices*.

-split

Generates detail 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 for the **-f_control** 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 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 (type */invoice/trial*) created with the "[pin_trial_bill_accts](#)" utility.

For information about trial billing, see "[About Trial Billing](#)."

The `pin.conf` file for this utility is in `BRM_Home/apps/pin_trial_bill`.

Location

`BRM_Home/bin`

Syntax

```
pin_trial_bill_purge [-start mm/dd/yy|+/- numberOfDays|0]
                    [-end mm/dd/yy|+/- numberOfDays|0]
                    [-active | -inactive | -closed]
                    [-f inputFile | -all]
                    [-verbose] [-help]
```

Parameters

-start mm/dd/yy or yyyy|+/- numberOfDays|0

The **start** and **end** dates determine the billing cycles for which trial invoices are purged. Billing cycles must fall entirely within the start and end date range. 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` purges trial invoices for *all* billing cycles before the specified **end** date.

-end mm/dd/yy or yyyy|+/- numberOfDays|0

The utility selects all accounts whose billing date is before 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.

Examples:

Purge trial invoices for accounts whose billing date is before **4/1/2002** (current date is 3/15/2002):

```
pin_trial_bill_purge -end 4/1/2002
pin_trial_bill_purge -end +16
```

Purge trial invoices for accounts whose billing date is before **3/1/2002** (current date is 3/15/2002):

```
pin_trial_bill_purge -end 3/1/2002
pin_trial_bill_purge -end -14
```

Purge trial invoices for accounts whose billing date is before the current date:

```
pin_trial_bill_purge -end 0
```

pin_trial_bill_purge

Purge trial invoices for accounts with complete billing cycles between **2/1/2002** and **3/15/2002** (current date is 3/15/2002):

```
pin_trial_bill_purge -start 2/1/2002 -end 3/15/2002  
pin_trial_bill_purge -start -45 -end 0
```

-activel-inactivel-closed

Searches for accounts whose status is active, inactive, or closed. By default, all accounts are included in the search.

Note: The **active**, **inactive**, or **closed** parameter does not apply when you use the **-f** parameter.

-f inputFile

Specifies the text file that contains the list of account POIDs for the accounts to process.

Example:

Purge trial invoices for accounts in the *inputFile* and whose billing date is less than **4/1/2002** (current date is 3/15/2002):

```
pin_trial_bill_purge -end 4/1/2002 -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 redirect the output to a log file, use the following syntax, where *filename.log* is 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 a 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_items_journals

Use the **pin_update_items_journals** utility to process the temporary item and journal data and update the main item and journal tables. This utility is run when the value of the **StagedBillingFeeProcessing** business parameter specifies that the service charges should be aggregated to a single account-level item. See "[About Applying Cycle Forward Fees in Parallel](#)" for more information.

To connect to the BRM database, the **pin_update_items_journals** utility needs a configuration file in the directory from which you run the utility. See "Creating Configuration Files for BRM Utilities" in *BRM System Administrator's Guide*.

Location

BRM_Home/bin

Syntax

```
pin_update_items_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_items_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_items_journals** utility encounters an error while processing items and journal data in the temporary tables, 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:

- Updating journal objects: `PIN_BILL_FLAGS_UPDATE_JOURNALS_ERROR` (bit value 0x2000)
- Updating item objects: `PIN_BILL_FLAGS_UPDATE_ITEMS_ERROR` (bit value 0x4000)

Important: If any subordinate bill unit caused the failure, **pin_update_items_journals** updates the billing statuses set in the `PIN_FLD_BILLING_STATUS` and `PIN_FLD_BILLING_STATUS_FLAGS` fields of the **/billinfo** object for both subordinate and parent bill units.

After you have resolved the processing errors, you can reprocess the items and journals data by running the **pin_update_items_journals** utility again.