

Oracle Utilities Customer Care and Billing

Administration Guide

Volume 3

Release 2.3.1

E18368-01

September 2010

Copyright © 2000, 2010, 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 software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle America, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

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 which 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.

This software or hardware and documentation may provide access to or information on 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. 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.

Defining Loan Options

The topics in this section describe how to set up the system to enable loan functionality.

Loans are optional. The system configuration requirements described in this section are only relevant if your organization loans money to customers.

Contents

- [The Terms Of A Loan Are Stored On A Service Agreement](#)
- [Payoff Balance and Current Balance for Loans](#)
- [Booking The Principal Amount Using An Adjustment](#)
- [Loan Amortization Schedule Calculation](#)
- [Billing For Loans And Interest Calculation](#)
- [Paying What Is Owed](#)
- [Overpayments On Loans](#)
- [Adjusting Loan Amounts](#)
- [Writing Off Loans](#)
- [Distribution Codes for Loans](#)
- [Setting Up The System To Enable Loans](#)

The Terms Of A Loan Are Stored On A Service Agreement

Loans are initiated by creating a loan service agreement for a customer. The loan service agreement (and its SA Type) contains the loan's terms:

- The **loan amount** is held in the service agreement's Total Amount to Bill.
- The **customer's periodic payment amount** is held in the service agreement's Recurring Charge Amount.
- The **number of amortization periods** (e.g., 36 months, 240 months, etc.) is held in the service agreement's Number of Payment Periods.
- If the **interest rate** is the same for all customers with this type of loan service agreement, the interest rate is defined on the service agreement's SA type (using a bill factor). A specific interest rate can be defaulted from a start option contract value. If a specific interest rate applies to the customer, the SA type's interest rate can be overridden by specifying a bill factor value on the customer's service agreement (where the bill factor value contains the specific interest rate for the customer).
- The SA type controls the **periodicity of the bills** (e.g., monthly or bi-weekly).

Because a loan is defined using a service agreement, the typical functionality that is controlled by the service agreement's SA type is supported, including:

- How and when it is billed.
- How payments are booked in the GL (and the payment priority relative to other service agreements).
- How its debt is monitored by credit and collections.

- How late payment charges are calculated.

Loan service agreements are created using [start / stop](#) just like all other service agreements. The start/stop transaction has special loan functionality that allows an operator to specify the service agreement-specific loan terms described above. A start option can be specified to override the SA type's interest bill factor.

Automatic calculation of periodic payment amount / number of periods. The system calculates a loan's periodic payment amount or number of payments (whichever is left blank). You can have the system do this on [Start/Stop Maintenance](#) (using the **Calculate** button that appears on the [start service confirmation window](#)), and on [Loan - Main](#) (by clicking the **Calculate** button). Regardless of where you do this, the calculation is performed by an algorithm on the loan's SA type. Refer to the [LPDA-SI](#) algorithm type for more information about the base package algorithm.

Payoff Balance and Current Balance for Loans

As described under [Current Amount versus Payoff Amount](#), a loan service agreement's current balance and payoff balance differ during the lifetime of the loan. Current balance contains how much the customer owes to remain current (typically their periodic payment amount), and payoff balance contains the amount the customer would have to pay to payoff the loan (typically the principal balance plus any accrued interest charges).

Unlike other SAs, loans have two accounts receivable distribution codes: long term and short term. These two codes allow the general ledger to differentiate between unbilled loan receivables (long term) and billed loan receivables (short term). The current balance for a loan is always the amount of the short-term receivables. The payoff balance for a loan is always the net of the short-term receivables and the long-term receivables.

If the SA has a [special role](#) of **Loan**, the financial transaction algorithms supplied with the base package transfer the current amount between the long-term receivables and the short-term receivables in the GL. For example, when a bill segment is generated for the loan SA, the amount of the periodic payment is transferred from the long-term receivables to the short-term receivables. Don't worry, the examples in the following sections show exactly what these transactions look like.

An operator can see the how the bills, payments and adjustments have affected the GL, current balance and payoff balance using [SA Financial History](#).

The following sections provide examples of how adjustments, bills and payments are recorded in the GL and the subsequent effect on the current and payoff balances. When reading the examples, remember that the payoff balance is always the net of the short-term receivable and the long-term receivable balances.

Booking The Principal Amount Using An Adjustment

When a loan service agreement is activated (i.e., when its status changes from *pending start* to *active*), an adjustment is created to book the principal amount. If the customer takes out a loan of 10,000, the adjustment's financial transaction looks as follows:

Event	GL Accounting	Effect On Current Balance	Effect On Payoff Balance	Current Balance	Payoff Balance
Loan service agreement is activated (and an adjustment is created to book the principal)	Long Term Loan Receivable 10,000 Cash <10,000>	0	10,000	0	10,000

This adjustment is issued if:

- The service agreement's SA type indicates a [special role](#) of **Loan**.
- The loan service agreement's Total Amount to Bill contains an amount (i.e., the loan amount).
- The loan service agreement was created using a [start option](#) on [Start/Stop Maintenance](#) AND the start option references an [adjustment type](#) and this adjustment type has been set up as follows:
 - The adjustment type's distribution code should reference the GL account to credit (e.g., Cash).
 - The adjustment type's FT algorithm reference **$\text{Payoff Amt} = \text{Adj Amt} / \text{Current Amt} = 0$** (booking principal only impacts a customer's payoff balance).

Note that because this financial transaction doesn't have a current amount (the customer doesn't actually owe a current amount yet), there is no need to book anything to the short-term receivables distribution code.

Loan Amortization Schedule Calculation

The amortization schedule is a projection of the amount of principal and interest in each payment over the life of the loan. The amortization schedule may change, for example if the interest rate changes or the customer makes an overpayment (reducing the principal balance).

A loan's amortization schedule is calculated when an operator clicks the **Calculate** button on [Loan - Main](#). Please be aware that when this button is clicked, an algorithm plugged in on the loan [SA type](#) actually calculates the amortization schedule (refer to the algorithm type [LSCH-SI](#) for more information).

Billing For Loans And Interest Calculation

A bill segment is produced for a loan when its service agreement's account is next billed.

Factors on a loan's SA type controls when a bill segment is produced for a loan. Typically SA types for loan service agreements are set up to use anniversary [calendar billing](#). For this configuration:

- You must indicate the type of anniversary billing in the calendar billing option. Currently, we only support the **Anniversary Future Billing** (meaning that loans are billed in advance just like a classic home loan is). Refer to [Using The Anniversary Method](#) for more information about how this billing method controls the end date of the loan bill segment.
- You must reference [an anniversary billing frequency](#) consistent with the [recurring charge frequency](#) (e.g., monthly, quarterly, etc.).

To set up for a loan that is billed on a monthly basis, you would define the following fields in SA Type - Billing:

- **Use Calendar Billing:** Anniversary Future Billing
- **Anniversary Bill Frequency:** Monthly
- **Total Bill Amount:** Required
- **Recurring Charge:** Required
- **Recurring Charge Frequency:** Monthly
- **Total Amount To Bill Label:** Loan Amount
- **Recurring Chg Amt Label:** Payment Amount

If your type of loan must be billed on an exact date (for example, always on the 15th of the month) or with an exact number of days between each bill (for example, every 14 days), then your loan should be set up to use the calendar billing option of **Use Bill Period**. In order for your loan bills to be created on specific dates, the system makes the following assumptions:

- Your [bill cycle schedule](#) for the loan's account is defined with the dates that you want the loan to bill and is defined with the window start date equal to the window end date.
- You define a [bill period](#) schedule corresponding to your bill cycle schedule. Each bill period schedule's bill date should match your bill cycle window start date and each bill period schedule's bill segment end date should be set to the loan period end date.
- Considerations for the first bill. If the loan SA Type's Initial Start Date Option indicates that the first day of the service agreement should be billed, then the loan SA's start date should match the window start date of the next bill cycle schedule for the account. If the loan SA Type's Initial Start Date Option indicates that first day of the service agreement should not be included, then the loan SA's start date should be one day prior to the window start date of the next bill cycle schedule for the account.
- Define your [recurring charge frequency](#) to match the frequency of your bill periods.

How the bill segment affects the customer's balance, and how it affects the general ledger are controlled by several algorithms defined on the loan service agreement's [SA type](#) and [bill segment type](#):

- The SA type's loan schedule algorithm controls how the [loan amortization schedule](#) is calculated.
- The SA type's loan interest charge algorithm controls how interest is calculated.
- The bill segment type's create algorithm controls how the bill lines are constructed.

- The bill segment type's financial algorithm controls how the general ledger is affected by the bill.

The 2nd entry in the following table contains an example of the financial transaction produced by the base package algorithms (note, the first financial transaction in the table was described under [Booking The Principal Amount Using An Adjustment](#)).

Event	GL Accounting	Effect On Current Balance	Effect On Payoff Balance	Current Balance	Payoff Balance
Loan service agreement is activated	Long Term Loan Receivable 10000 Cash <10000>	0	10,000	0	10,000 long-term: 10,000
First bill segment is produced	<i>Interest:</i> Long Term Loan Receivable 41.66 Interest Revenue <41.66> <i>Transfer Long Term To Short Term:</i> Short Term Loan Receivable 438.71 Long Term Loan Receivable <438.71>	438.71	41.66	438.71	10,041.66 short-term: 438.71 long-term: 9,602.95

Several important points are illustrated above:

- When a bill segment is produced for a loan service agreement, the following takes place (if you use a bill segment creation algorithm of **Create Bill Segment for Loans**):
 - The loan SA type's bill segment creation algorithm calls the SA type's loan interest charge algorithm.
 - The base package loan interest charge algorithm calculates simple interest based on: 1) unbilled principal (i.e., the service agreement's payoff balance minus the current balance), 2) the number of billing periods covered by the bill, and 3) the interest rate. Refer to the algorithm type LINT-SI for more information about the base package interest calculation algorithm.

No interest on interest and no interest on past due amounts. Just like a classic home loan, the base package algorithms do not calculate interest on interest, nor do they calculate interest on past due amounts. If you want to levy a late payment charge, use the SA type's late payment processing. If your organization calculates interest differently, you must develop your own algorithm(s).

- The SA type's bill segment creation algorithm uses the calculated interest to format the bill segment's bill lines. It creates one bill line to show the amount of interest in the payment, and another bill line to show the amount of principal. The principal amount is equal to the service agreement's periodic payment amount minus the amount of calculated interest.
- The financial transaction illustrated above is created if you use a bill segment financial algorithm of **Payoff Amt = Interest / Current Amt = Bill Amount** on the loan's bill segment type. The following explains how this algorithm works:

- The SA's current balance increases by the amount of the loan's periodic payment amount (i.e., its recurring charge amount). In other words, the amount the customer thinks they owe increases by 438.71.
- The SA's payoff balance increases by the amount of interest. In other words, if the customer wanted to payoff the loan, they'd owe 10,041.66.
- The Interest portion of the GL accounting is straightforward (if you're an accountant). It simply takes the amount of interest and debits it to the SA type's receivable distribution code (long-term receivables) and credits it to the distribution code defined on the interest rate's bill factor (typically interest revenue).
- The Transfer portion of the GL accounting transfers moneys from long-term receivables (i.e., the unbilled principal) to short term receivables (the billed portion of the debt). The amount transferred is equal to the FT's effect on the service agreement's current balance, allowing the general ledger to differentiate between unbilled loan receivables (long term) and billed loan receivables (short term). Remember that the payoff balance is the net of the short-term and long-term receivables.

Paying What Is Owed

When a payment is made for a loan:

- The service agreement's payoff amount is reduced by the payment amount.
- The service agreement's current amount is reduced by the payment amount.

The events that happen when a customer makes a payment against a loan is controlled by the FT algorithm plugged in on the loan service agreement's payment segment type. We'll use an example to help explain how this algorithm works. The 3rd entry in the following table illustrates the financial transaction produced when a payment is made (note, the first two financial transactions were described above).

Event	GL Accounting	Effect On Current Balance	Effect On Payoff Balance	Current Balance	Payoff Balance
Loan service agreement is activated	Long Term Loan Receivable 10000 Cash <10000>	0	10,000	0	10,000 long-term: 10,000
First bill segment is produced	<i>Interest:</i> Long Term Loan Receivable 41.66 Interest Revenue <41.66> <i>Transfer Long Term To Short Term:</i> Short Term Loan Receivable 438.71 Long Term Loan Receivable <438.71>	438.71	41.66	438.71	10,041.66 short-term: 438.71 long-term: 9,602.95
Payment is made	<i>Affect Cash</i> Cash 438.71 Long Term Loan Receivable <438.71> <i>Transfer Long Term To Short Term:</i>	-438.71	-438.71	0	9,602.95 short-term: 0 long-term: 9,602.95

	Long Term Loan Receivable 438.71			
	Short Term Loan Receivable <438.71>			

The financial transaction illustrated above is created if you use a payment segment FT creation algorithm of ***Payoff Amt = Current Amt = Pay Amt*** on the loan's payment segment type. The following explains how this algorithm works:

- The SA's current balance decreases by the amount of the payment amount. In other words, the customer thinks they owe 0 after the payment. Note refer to [Overpayments](#) for information about how an overpayment affects the SA's balances and the general ledger.
- The SA's payoff balance decreases by the amount of the payment. In other words, if the customer wanted to payoff their loan, they'd owe 9,602.95.
- The Cash portion of the GL accounting is straightforward (if you're an accountant). It simply takes the amount of the payment and debits it to the payment segment type's distribution code (typically cash) and credits it to the SA type's receivable distribution code (long-term receivable).
- The Transfer portion of the GL accounting effectively reduces short-term receivables by the FT's effect on the customer's current balance. This reduction is handled by an offsetting increase to long-term receivables (to make up for reduction made when the cash was applied). Again, this differentiation between short-term and long-term receivables allows the general ledger to differentiate between unbilled loan receivables (long term) and billed loan receivables (short term).

Overpayments On Loans

You can determine whether you want to accept loan overpayments. Overpayments reduce the principal amount (the amount owed on the loan), which follows the philosophy adopted by a typical home loan.

When the payment is made, any overpayments are distributed according to the overpayment distribution algorithm defined for the customer class. Any customer classes for which you want to allow loan overpayments should use an overpayment distribution algorithm that keeps the overpayment on the loan SA.

When the payment transaction is frozen, the system checks to see if there is a credit amount on the loan SA's current balance. If a credit exists, the customer has made an overpayment and an adjustment is created to zero out the current balance and transfer the amount of the credit from the SA's short-term receivable to long-term receivable. The adjustment may appear on the customer's next bill to show the additional amount paid against the principal.

The algorithm that controls this adjustment to remove the credit on current balance is plugged in on the loan service agreement's SA type and is applied on the ***SA Type – Payment Freeze*** system event.

The 3rd and 4th entries in the following table illustrate an overpayment (note, the first two financial transactions were described above).

Event	GL Accounting	Effect On Current Balance	Effect On Payoff Balance	Current Balance	Payoff Balance
Loan service agreement is activated	Long Term Loan Receivable 10000 Cash <10000>	0	10,000	0	10,000 long-term: 10,000
First bill segment is produced	<i>Interest:</i> Long Term Loan Receivable 41.66 Interest Revenue <41.66> <i>Transfer Long Term To Short Term:</i> Short Term Loan Receivable 438.71 Long Term Loan Receivable <438.71>	438.71	41.66	438.71	10,041.66 short-term: 438.71 long-term: 9,602.95
Payment is made (with an overpayment of 200.00)	<i>Affect Cash</i> Cash 638.71 Long Term Loan Receivable <638.71> <i>Transfer Long Term To Short Term:</i> Long Term Loan Receivable 638.71 Short Term Loan Receivable <638.71>	-638.71	-638.71	-200.00	9,402.95 short-term: <200.00> long-term: 9,602.95
Create adjustment to remove SA's credit.	<i>Transfer Short Term Credit to Long Term:</i> Short Term Loan Receivable 200.00 Long Term Loan Receivable <200.00>	200.00	0	0	9,402.95 short-term: 0 long-term: 9,402.95

In the 3rd financial transaction illustrated above, the billed amount of the payment works essentially the same as that illustrated under [Paying What Is Owed](#). If the **Keep Overpayment on Loan SA** algorithm is plugged in on the overpayment distribution event on the customer class, the overpayment amount is applied to the loan SA, creating a credit on the SA's current balance. The following explains how this algorithm works:

- If the account has an loan SA and there is an excess credit, the credit is applied to the loan SA (as long as this does not cause the loan SA to have a credit payoff balance).
- If there is not a loan SA to which the credit can be applied, the algorithm checks to see if there is an open excess credit SA for the account.
 - If so, the excess credit amount is applied to the excess credit SA.
 - If not, the algorithm creates an excess credit SA and applies the amount to this SA.

The 4th financial transaction illustrated above is created if the **Create Adjustment to Remove SA's Credit** algorithm is plugged in on the loan's SA type's payment freeze event. The following explains how this algorithm works:

- If the SA's current balance is less than zero, the algorithm creates a frozen adjustment that removes the credit by transferring the credit from the short-term receivable to the long-term receivable. This adjustment ID is captured on the pay segment so the adjustment can be canceled if the payment is later canceled. Note that the adjustment cancel reason used by the system in this case is specified on the [Financial Transaction Options Feature Configuration](#) using the [Adjustment Cancel Reason For Payment Linked To Adjustment](#) option type.
- The SA's current balance increases by the amount of the credit transfer. In other words, the customer thinks they owe 0 after the transfer.
- The SA's payoff balance doesn't change because the payoff balance is always the net of the short-term and long-term receivables. In other words, if the customer wanted to payoff their loan, they'd still owe 9,402.95.

Overpayments and interest. The base package interest calculation algorithm (plugged in on the loan's SA type) does not take into consideration the exact date that the overpayment is made when calculating the interest for the period. It only takes into consideration the outstanding principal amount (payoff balance - current balance) at the time of the interest calculation.

Adjusting Loan Amounts

You would issue ad hoc adjustments if you need to change a loan's payoff and/or current balance outside of the normal billing / payment functions.

An adjustment can:

- Reduce a loan's payoff balance.
- Reduce a loan's current balance (i.e., how much the customer thinks they currently owe).
- Reduce both the loan's payoff and current balance.

For adjustments that affect payoff amount only:

- These adjustments are used to change the principal owed, e.g., if an additional amount is loaned.
- The adjustment's adjustment type should reference the ***Payoff Amt = Adj / Current Amt = 0*** FT algorithm.
- GL lines will be generated to reflect the change to principal.

For adjustments that reduce current amount only:

- These adjustments can be used to change the amount that the customer must pay as part of the next bill.
- The adjustment's adjustment type should reference the ***Payoff Amt = 0 / Current Amt = Adj Amount (no GL)*** FT algorithm.
- Typically, GL lines are not generated for FTs that only affect the customer's current balance. However, moneys must be transferred from long to short in the amount of the adjustment (as described above under [Payoff Balance and Current Balance for Loans](#)).

For adjustments that reduce both payoff and current amount:

- These adjustment types can be used to levy additional charges, such as late fees, or to correct interest calculations.
- The adjustment's adjustment type should reference the ***Payoff Amt = Adj / Current Amt = Adj*** FT algorithm.
- GL lines are generated to reflect the change to principal. In addition, GL lines must be generated to transfer money from long to short in the amount of the adjustment (as described above under [Payoff Balance and Current Balance for Loans](#)).

Note. Adjustments that affect the principal balance (payoff balance - current balance) affect the term of the loan because interest is based on the principal balance.

Adjustments can cause credit balances to exist. If you credit the loan SA, it is possible for the current balance to become negative. You may need to create additional adjustments that affect the current amount, depending on whether the customer needs to pay this amount as part of the next payment.

Writing Off Loans

Loans are written off using the standard write-off processing.

Refer to [The Big Picture Of Write Off Processing](#) for background information.

We illustrate the classic financial transactions that transpire to financially write-off a loan to help illustrate important points (these are the 4th and 5th entries in the following table):

Event	GL Accounting	Effect On Current Balance	Effect On Payoff Balance	Current Balance	Payoff Balance
Loan service agreement is activated	Long Term Loan Receivable 1000 Cash <1000>	0	10,000	0	10,000 long-term: 10,000
First bill segment is produced	<i>Interest:</i> Long Term Loan Receivable 41.66 Interest Revenue <41.66> <i>Transfer Long Term To Short Term:</i> Short Term Loan Receivable 438.71 Long Term Loan Receivable <438.71>	438.71	41.66	438.71	10,041.66 short-term: 438.71 long-term: 9,602.95
Payment is made (with an overpayment of 200.00)	<i>Affect Cash</i> Cash 638.71 Long Term Loan Receivable <638.71> <i>Transfer Long Term To Short Term:</i> Long Term Loan Receivable 638.71	-638.71	-638.71	-200.00	9,402.95 short-term: <200.00> long-term: 9,602.95

	Short Term Loan Receivable <638.71>				
Create adjustment to remove SA's credit.	<i>Transfer Short Term Credit to Long Term:</i> Short Term Loan Receivable 200.00 Long Term Loan Receivable <200.00>	200.00	0	0	9,402.95 short-term: 0 long-term: 9,402.95
Sync up current with payoff balance	<i>Transfer Long Term To Short Term:</i> Short Term Loan Receivable 9,402.95 Long Term Loan Receivable <9,402.95>	9,402.95	0	9,402.95	9,402.95 short-term: 9,402.95 long-term: 0
Transfer balance to a write-off SA	<i>Transfer From Loan SA:</i> Write Off Xfer Clearing 9,402.95 Long Term Loan Receivable <9,402.95> Long Term Loan Receivable 9,402.95 Short Term Loan Receivable <9,402.95> <i>Transfer To Write Off SA:</i> Bad Loans Expense 9,402.95 Write Off Xfer Clearing <9,402.95> Note, these will cause a balance of 9,402.55 to exist on the write-off service agreement.	-9,402.95	-9,402.95	0	0

The only unusual portion of the last two financial transactions is the impact on short and long term receivables. Please see the examples above under [Billing For Loans And Interest Calculation](#) and [Paying What Is Owed](#) to understand how any impact to a loan's current balance causes this type of financial transfer to occur.

Distribution Codes for Loans

As explained above under [Payoff Balance and Current Balance for Loans](#), loans have two accounts receivable distribution codes: long term and short term. These two codes allow the general ledger to differentiate between unbilled loan receivables (long term) and billed loan receivables (short term). Both receivables distribution codes are defined on the loan SA type.

In addition, loans have a distribution code used to book interest revenue. The interest revenue distribution code is defined on the loan's bill factor value for a revenue class (defined on the loan's SA type). For example, on the bill factor you can use one distribution code to book interest revenue from the residential revenue class and another distribution code to book interest revenue from the commercial revenue class. In this example, you create two loan SA types, one for residential revenue and the other for commercial revenue.

Loans may also have a bad loan debt (expense) distribution code that is used when writing off a loan. The bad loan debt distribution code is defined on the loan's write-off service agreement type. Refer to [Defining Credit & Collections Options](#) for more information.

Setting Up The System To Enable Loans

The above topics provided background information about how loans are supported in the system. The following discussion summarizes the various setup tasks alluded to above.

Contents

- [Distribution Code](#)
- [Adjustment Types](#)
- [Adjustment Type Profile](#)
- [Algorithms](#)
- [Bill Factor](#)
- [Customer Class](#)
- [Bill Segment Type](#)
- [Frequency](#)
- [Bill Period](#)
- [Bill Cycle Schedule](#)
- [Collection, Severance and Write Off Processes](#)
- [SA Type](#)
- [Start Options](#)

Distribution Code

You must set up the following [distribution codes](#):

- Long term receivables
- Short term receivables
- Interest revenue
- Bad loan debt

Adjustment Types

The following adjustment types are needed:

- Activate a loan. This should reference a distribution code associated with cash or the cash equivalent and an FT algorithm of ***Payoff Amt = Adj / Current Amt = 0***.

Creating Checks for Loan Amounts. If you want the system to initiate a check to the customer for the loan amount, the loan activation adjustment type should indicate an **A/P Request Type Code**. Refer to [Controls The Interface To A/P & 1099 Reporting](#) for more information.

- Remove Credit (Overpayment). This adjustment should reference an FT algorithm of ***Payoff Amt = 0 / Current Amt = Adj Amount (no GL)***. Even though this algorithm indicates that there is no effect on the GL, there is a special exception built in for loan SAs. The special exception creates GL details to transfer the current balance (short-term receivable) to the long-term receivable. Refer to [Overpayments](#) for more information.

Printing Overpayments on Bills. If you want the overpayment adjustment to appear on customers' bills, turn on **Print by Default** and enter a **Description on Bill** (e.g. "Additional Principal"). For more information, refer to [Controls Information Printed On The Bill](#).

- Adjustment types to perform any of the adjustments described under [Adjusting Loan Amounts](#).

Adjustment Type Profile

Create an adjustment type profile that references the adjustment types used on a loan. Besides the above adjustment types, it should also reference adjustment types to levy late payment charges (if applicable), levy non-sufficient funds charges (if applicable), refund overpayments (if applicable), sync current balance with payoff balance at write-off time, transfer balances to a write-off service agreement, and write-down small balances.

Algorithms

Add the following [algorithms](#):

- Overpayment Distribution. This algorithm is later plugged in on the customer class for the **Overpayment Distribution** system event. Refer to the algorithm type [OVPY-LO-CSA](#) for more information about the base package algorithm.
- Bill Segment Creation for Loans. This algorithm is later plugged in on the loan's bill segment type. Refer to the algorithm type [BSBS-LO](#) for more information about the base package algorithm.
- Bill Segment Financial Transaction Creation for Loans. This algorithm is later plugged in on the loan's bill segment type. Refer to the algorithm type [BSBF-LO](#) for more information about the base package algorithm.
- Amortization Schedule. This algorithm is later plugged in on the SA type for the **Loan Schedule** system event. Refer to the algorithm type [LSCH-SI](#) for more information about the base package algorithm.
- Interest Calculation. This algorithm is later plugged in on the SA type for the **Loan Interest Charge** system event. Refer to the [LINT-SI](#) algorithm type for more information about the base package algorithm.
- Payment Periods/Payment Amount Calculation. This algorithm is later plugged in on the SA type for the **Loan Periods and Amount** system event. Refer to the algorithm type [LPDA-SI](#) for more information about the base package algorithm.
- Loan SA Payment Freeze. This algorithm has a parameter that must reference the Remove Credit adjustment type defined above. This algorithm is later plugged in on the SA type for the **Payment Freeze** system event. Refer to the algorithm type [STPZ-RMVCR](#) for more information about the base package algorithm.

Bill Factor

You must set up a [bill factor](#) that defines the interest rate. If you have different interest rates for different types of loans, you can create a separate bill factor for each or you can use start options to override the interest rate.

On the bill factor's [bill factor value](#), make sure to reference the GL distribution code used to book interest revenue for the revenue class specified on the loan's SA type.

Customer Class

Any [customer class](#) on which you want to allow overpayments for a loan must use the overpayment distribution algorithm defined above. The overpayment distribution algorithm keeps the overpayment on the loan SA rather than transferring it to an excess credit SA, allowing a subsequent adjustment to apply the overpayment to the principal balance.

Bill Segment Type

Create a bill segment type that references the bill segment creation algorithm defined above and the FT creation algorithm defined above.

Frequency

Create [frequency](#) codes to correspond to the frequency of your loans. When setting up your SA types, you must indicate a recurring charge frequency and, if you use the **Anniversary Billing Option**, you must indicate an anniversary frequency.

Bill Period

If you use the **Use Bill Period** option, set up a bill period with an appropriate schedule of dates for billing your loan.

Bill Cycle Schedule

If you use the **Use Bill Period** option, the bill cycle schedule for these types of loans should be defined with an appropriate schedule of dates for billing your loan.

Collection, Severance and Write Off Processes

You should set up the appropriate credit and collections information. Refer to [Defining Credit & Collections Options](#) for more information.

SA Type

You must set up a [SA type](#) for your loan service agreements (you may need multiple SA types if you have different business rules for different types of loans). The following points describe the minimal requirements for a loan SA type.

Contents

[SA Type - Main](#)

[SA Type - Detail](#)
[SA Type - Billing](#)
[SA Type - Rate](#)
[SA Type - SP Type](#)
[SA Type - Adjustment Profile](#)
[SA Type - Credit and Collections](#)
[SA Type - Algorithms](#)

SA Type - Main

Define the following options:

- **Distribution Code** should be a long-term receivable code.
- **Service Type** should reference something like "Miscellaneous Service".
- Specify the **Revenue Class** that, together with the interest bill factor, determines the distribution code used to book the loan interest revenue.
- **Start Option** should be turned on.
- The **Payment Segment Type** should reference the **Normal Payment**. The base package payment segment financial algorithm (**Payoff Amt = Current Amt = Pay Amt**) used for Normal Payment pay segment types creates the additional GL details to transfer the credit from long-term receivables to short-term receivables if the SA's special role is Loan.
- Turn on **Late Payment Charge** if applicable.
- Define an appropriate **Adjustment Type (Synch Current)** that will cause current balance to be synchronized with payoff balance (if the loan is [written off](#)).

SA Type - Detail

Define the following options:

- **Special Role** is **Loan**.
- Specify the **Interest Bill Factor** set up above. You can use start options to override.
- Use **Override Interest Flag** to indicate whether the interest rate defined on the interest bill factor may be overridden at the SA level. If you select **Allowed**, the interest rate may be overridden by a contract value on a start option or the SA.
- Use the short-term receivable account defined above as the **Loan A/R Distribution Code**. If you do not want the system to differentiate between short-term receivables and long-term receivables, make the loan A/R distribution code the same as the distribution code (above).

SA Type - Billing

For an overview of some of these options, be sure to refer to [Billing For Loans And Interest Calculation](#) for more information.

Define the following options:

- The [Bill Segment Type](#) should reference the value created above.
- **Characteristic Premise Required** should not be checked. (A loan SA is not a premise-based service. SA types that have this box checked are filtered out of the SA type search when the start method is Start a SA, so users will never be able to start a loan SA that requires a characteristic premise.)
- **Use Calendar Billing** must equal ***Anniversary Future Billing*** or ***Use Bill Period***.
- **Bill Period** is required for the ***Use Bill Period*** option.
- **Anniversary Billing Frequency** is required for the ***Anniversary Future Billing*** option and must equal the periodicity of the bills (monthly, weekly, etc.).
- **Total Bill Amount** is required (it holds the principal).
- **Total Amount To Bill Label** should reference something like “Loan Amount”.
- **Recurring Charge Amount** is required (it holds the periodic payment amount).
- **Recurring Chg Amt Label** should reference something like “Payment Amount”.
- **Recurring Charge Frequency** is required (it defines the periodicity associated with the recurring charge amount) and must be the same as the bill period frequency or the anniversary billing frequency (based on your **Use Calendar Billing** option).

SA Type - Rate

Turn off the **Rate Required** switch as loans do not use rates.

SA Type - SP Type

Turn off the **Service Points Required** switch as loans do not have service points.

SA Type - Adjustment Profile

Adjustment Type Profile should reference the profile set up above.

SA Type - Credit and Collections

The credit and collections information should reference a **Severance Process Template** that simply expires the loan. The **Debt Class** and **Write Off Debt Class** should reference an appropriate value consistent with your credit and collections rules. Refer to [Defining Credit & Collections Options](#) for more information.

SA Type - Algorithms

The ***Loan Schedule***, ***Loan Interest Charge***, ***Loan Periods and Amounts*** and ***Payment Freeze algorithms*** defined above must be set up.

Start Options

Loans require a start option to define the adjustment type that is used to create the adjustment to book the initial principal amount when the loan is activated. Loan start options can also specify default values for loan amount, payment amount, and number of periods.

Create at least one start option for each loan SA type. The following information should be defined:

- **Adjustment Type** should reference the adjustment type defined above to book principal.
- **Recurring Charge Amount** (Payment Amount) should only be defined if you have standard loan payments.
- **Total Amount To Bill** (Loan Amount) should only be defined if you have standard loan amounts.
- **Number of Payment Periods** should be the number of payment periods in the loan (if you have a standard loan period).
- If you want to define a set of standard APRs for a loan SA Type, set up a **Contract Value** on the Rate Info page. The **Bill Factor** should match the interest bill factor specified on the [SA Type](#).

Defining Non-billed Budget Options

A non-billed budget (NBB) is a payment plan that allows your customers to pay set amounts at specified intervals (e.g. every two weeks). Non-billed budgets are typically used when your company bills on an infrequent basis and you want to provide your customers with a mechanism to make smaller payments more frequently. As the name suggests, bills are not created for the non-billed budget's scheduled payments; customers must remember to make their payments at the scheduled intervals. The topics in this section describe how to design and set up non-billed budgets.

Non-billed budgets are optional. The system configuration requirements described in this section are only relevant if your organization offers non-billed budgets.

Contents

- [What Is A Non-billed Budget?](#)
- [The Financial Impact Of Non-billed Budgets](#)
- [Financial Transactions For Unmonitored Non-billed Budgets](#)
- [Designing Non-billed Budgets](#)
- [Non-billed Budget Recommendation Rule](#)
- [Setting Up The System To Enable Non-billed Budgets](#)

What Is A Non-billed Budget?

A non-billed budget is a special type of budget or payment plan that encompasses three major elements:

- A set of scheduled payments
- The business rules used to recommend and potentially renew the payment schedule
- The business rules that govern the financial impact on the current and payoff balances of the SAs covered by the payment schedule

Non-billed budgets are managed via a service agreement whose SA type has a special role of **Non-billed Budget**. If an SA type has a special role of non-billed budget it must have one or more recommendation rules, and SAs of that type are allowed to have payment schedules.

The Financial Impact Of Non-billed Budgets

When you set up the non-billed budget SA type, you can indicate whether the non-billed budget is monitored by the [account debt monitor](#) process. The following sections contain examples of financial transactions for non-billed budgets that are monitored by the account debt monitor.

Unmonitored non-billed budgets. SAs that are covered by unmonitored non-billed budgets are subject to different financial treatment than those that are monitored. The [financial transactions relevant for unmonitored non-billed budgets](#) are discussed later.

Contents

[Current Amount For SAs Covered By A Non-billed Budget](#)
[Scheduled And Actual Payments On The Non-billed Budget](#)
[Overpayments for Non-billed Budgets](#)
[Underpayments For Non-billed Budgets](#)
[Billing For SAs Covered By The Non-billed Budget](#)
[Distributing Non-billed Budget Credit](#)
[Stopping an SA Covered By a Non-billed Budget](#)

Current Amount For SAs Covered By A Non-billed Budget

As described under [Current Amount versus Payoff Amount](#), the values of the current balance and payoff balance are the same for most financial transactions. One exception is for SAs that are covered by a non-billed budget in which case the current balance is always zero and the payoff balance is always the amount the customer really owes.

When a non-billed budget is activated or an SA is added to a non-billed budget, the system creates adjustments for all affected SAs to set the current amount equal to zero. The adjustment type that references the **Payoff Amt = 0 / Current Amt = Adj Amount (no GL)** algorithm is taken from the SA's SA type.

Non-billed Budgets and Open Item Accounts. If the non-billed budget is for an open item account, no match event is created for the financial transaction that reduces the current amount to zero. This FT must be manually matched if required.

The following example shows the effect of activating a non-billed budget that covers an electric SA with a current balance of 25. The transactions for the electric SA are illustrated on the right and the transactions for the non-billed budget SA are illustrated on the left.

Event	Effect On Current Balance	Effect On Payoff Balance	Current Balance	Payoff Balance	Effect On Current Balance	Effect On Payoff Balance	Current Balance	Payoff Balance
	Electric SA				Non-billed Budget SA			
Starting balance	0	0	25	25				
Non-billed budget is activated	-25	0	0	25				

By setting the current balance for the covered SAs to zero, the SAs are insulated from the regular debt monitoring process. Depending on the non-billed budget [recommendation algorithm](#), the payoff balance can be ignored, divided evenly between the scheduled payments or added to the first scheduled payment.

Activating a non-billed budget has no affect on its own current or payoff balances.

Scheduled And Actual Payments On The Non-billed Budget

When a scheduled payment is due, an adjustment is created to increase the non-billed budget's current balance by the expected amount. The current balance on the SA can be monitored to ensure that payments are being made on time.

When the payment is made, the non-billed budget's current balance is reduced to zero and the non-billed budget's payoff balance reflects the accumulated credit from the payment. This accumulated credit is transferred to the covered SAs when the next bill for the account is completed.

The following example illustrates scheduled and actual payments for the non-billed budget in the previous example. The amount of the scheduled non-billed budget payments is 10. Note that the first two transactions were described above.

Event	Effect On Current Balance	Effect On Payoff Balance	Current Balance	Payoff Balance	Effect On Current Balance	Effect On Payoff Balance	Current Balance	Payoff Balance
	Electric SA				Non-billed Budget SA			
Starting balance	0	0	25	25				
Non-billed budget is activated	-25	0	0	25				
1 st scheduled payment is due					10	0	10	0
Payment					-10	-10	0	-10

An algorithm (***Process NBB Scheduled Payment***) plugged in on the non-billed budget SA type creates the appropriate financial transactions when the scheduled payment is due. The algorithm is called by the Non-billed Budget Scheduled Payment Processing (***NBBPS***) background process.

The normal payment processing handles the adjustments created when the payment is made.

Non-billed Budget Payment Cancellation. If a payment is canceled, the financial transaction is reversed.

Automatic Payments And Non-billed Budgets. Users may set up a non-billed budget to use [automatic payments](#) by setting up the account's auto pay options. Users may also exclude the non-billed budget from automatic payment if the account is set up for automatic payment. The [NBB Scheduled Payment Automatic Payment Create](#) background process calls the [Auto Pay Creation](#) algorithm to create the auto payment for a non-billed budget.

Overpayments for Non-billed Budgets

Typically, payments in excess of the non-billed budget's current balance are credited to an overpayment (excess credit) SA. When the next adjustment is created for a scheduled payment, the credit on the overpayment SA is used to relieve the non-billed budget current balance.

Overpayment Distribution Algorithm. The overpayment distribution is a function of the overpayment distribution algorithm plugged in on the account's customer class. We strongly recommend that the non-billed budget SA type be set up so that overpayment is not allowed. Any excess payments should go to an overpayment SA. For more information about overpayment distribution, refer to [Overpayment Segmentation](#).

In the example below, the customer pays 20 for a scheduled payment instead of 10. The non-billed budget SA is illustrated on the right and the overpayment SA is illustrated on the left. The electric SA is not illustrated in the example below because scheduled payments, payments and overpayments have no effect on covered SAs. The first four transactions were illustrated above.

Event	Effect On Current Balance	Effect On Payoff Balance	Current Balance	Payoff Balance	Effect On Current Balance	Effect On Payoff Balance	Current Balance	Payoff Balance
	Non-billed Budget SA				Overpayment SA			
Starting balance								
Non-billed budget is activated								
1 st scheduled payment is due	10	0	10	0				
Payment	-10	-10	0	-10				
2 nd scheduled payment is due	10	0	10	-10				
Over-payment	-10	-10	0	-20	-10	-10	-10	-10
3 rd scheduled payment is due	10	0	10	-20				
Transfer Adjustment	-10	-10	0	-30	10	10	0	0

After the **Process NBB Scheduled Payment** algorithm creates the next scheduled payment, it looks for a credit amount on the overpayment SA and creates an adjustment to transfer the credit balance (or the amount of the payment if the credit is more than the scheduled payment amount) from the overpayment SA to the non-billed budget SA. The overpayment transfer adjustment type and the overpayment SA type are specified as parameters to the algorithm.

Underpayments For Non-billed Budgets

An insufficient payment or a canceled payment leaves a current balance on the non-billed budget SA.

In the example below, the customer makes a payment of 7 for the fourth scheduled payment. The example below does not show the overpayment SA (illustrated above). The first eight transactions are discussed above.

Event	Effect On Current Balance	Effect On Payoff Balance	Current Balance	Payoff Balance	Effect On Current Balance	Effect On Payoff Balance	Current Balance	Payoff Balance
-------	---------------------------	--------------------------	-----------------	----------------	---------------------------	--------------------------	-----------------	----------------

	Electric SA				Non-billed Budget SA			
Starting balance	0	0	25	25				
Non-billed budget is activated	-25	0	0	25				
1 st scheduled payment is due					10	0	10	0
Payment					-10	-10	0	-10
2 nd scheduled payment is due					10	0	10	-10
Over-payment					-10	-10	0	-20
3 rd scheduled payment is due					10	0	10	-20
Transfer Adjustment					-10	-10	0	-30
4 th scheduled payment					10	0	10	-30
Underpayment					-7	-7	3	-37

The **Process NBB Scheduled Payment** algorithm creates a trigger to ensure that the current balance is monitored by the account debt monitor. Refer to [Credit And Collections And Non-billed Budgets](#) for more information.

Billing For SAs Covered By The Non-billed Budget

When the next bill for the account is completed, the credit on the non-billed budget is transferred to the covered SAs. The credit is prorated over the covered SAs according to the relative payoff balances on each SA.

In the example below, the electric SA's bill is 33. The first ten transactions are discussed above.

Event	Effect On Current Balance	Effect On Payoff Balance	Current Balance	Payoff Balance	Effect On Current Balance	Effect On Payoff Balance	Current Balance	Payoff Balance
	Electric SA				Non-billed Budget SA			
Starting balance	0	0	25	25				
Non-billed budget is activated	-25	0	0	25				
1 st scheduled payment is due					10	0	10	0
Payment					-10	-10	0	-10
2 nd scheduled payment is due					10	0	10	-10
Over-payment					-10	-10	0	-20

3 rd scheduled payment is due					10	0	10	-20
Transfer Adjustment					-10	-10	0	-30
4 th scheduled payment					10	0	10	-30
Underpayment					-7	-7	3	-37
Bill	0	33	0	58				
Bill completion (transfer adjustment)	0	-37	0	21	0	37	3	0

When a bill segment financial transaction is created, the current amount is set to zero for SAs that are covered by the non-billed budget (if you use a bill segment FT algorithm of **Payoff Amt = Bill Amt / Current Amt = Amt Due**). Though not evident by the name, the **Payoff Amt = Bill Amt / Current Amt = Amt Due** algorithm does set the current amount to zero for monitored non-billed budgets. (For SAs with other roles, the current amount is equal to the amount due or the recurring charge.)

A bill completion algorithm transfers money from the non-billed budget to the covered SAs (if you plug in the **NBB Credit Transfer** bill completion algorithm on the non-billed budget SA type). The algorithm type supplied with the base package distributes the credit using the method described in [Distributing Non-billed Budget Credit](#).

Canceled Bill Segments. No new processing occurs when a bill segment is canceled; any credit balance remains on the covered SA.

Distributing Non-billed Budget Credit

Both the NBB Credit Transfer bill completion algorithm type and the non-billed budget SA stop algorithm type supplied with the base package use the same method of distributing a credit from a non-billed budget SA to the covered SAs. The following points describe how the credit is distributed:

- Covered SAs that are already in credit (due to some other circumstance, such as a cancellation and rebill) are excluded from the distribution.
- The distribution to each covered SA will not exceed its total payoff to ensure that none of the covered SAs have a credit balance.
- The credit is prorated over the covered SAs according to the relative payoff balances on each SA.
- The calculation of the payoff balance is adjusted to exclude the current balance to ensure that the credit is prorated over the debt covered by the budget, not any ad-hoc debt for the SA.

Note. The current balance on covered SAs should always be zero. The only exception occurs if an adjustment has been added that directly affected the SA balance. In this case, the distribution assumes that the balance is outside the non-billed budget and needs to be paid separately.

- Any excess credit remains on the non-billed budget SA until the next distribution takes place or until the [non-billed budget SA is stopped](#).
- The type of adjustments created is determined by the **Adjustment Type (Xfer)** specified on the non-billed budget SA type.

The examples in the table below illustrate the points above.

NBB SA Credit	SA 1 Payoff	SA 1 Current	SA 2 Payoff	SA 2 Current	SA 1 Credit Xfer	SA 2 Credit Xfer
-100.00	-100.00	0.00	-200.00	0.00	0.00	0.00
-100.00	150.00	0.00	-50.00	0.00	-100.00	0.00
-300.00	150.00	0.00	50.00	0.00	-150.00	-50.00
-100.00	150.00	0.00	250.00	0.00	-37.50	-62.50
-100.00	150.00	0.00	250.00	50.00	-42.86	-57.14

The first example above shows that covered SAs with a credit balance are excluded from the distribution. Any excess credit remains on the non-billed budget.

The second example shows that one covered SA has a credit balance, so the entire credit is distributed to the remaining SA.

The third example shows the amount distributed to a covered SA does not exceed its payoff balance. Again, any excess credit remains on the non-billed budget SA.

The fourth example illustrates how the credit is prorated based on the payoff balance. The fifth example illustrates the same prorating but with a current balance on one of the SAs (SA 2). (Remember that the prorating excludes any current balance.)

The prorated amount is calculated by subtracting the current balance from the payoff balance then multiplying the result by the distribution amount and dividing by the total payoff owing of all covered SAs.

Stopping an SA Covered By a Non-billed Budget

If a service agreement covered by a non-billed budget is stopped, the system must bring the current balance and payoff balance of the covered SA back in synch. The system creates an adjustment using the Synch Current adjustment type from the SA's SA type.

In the example below, the electric SA's payoff balance is 21. The first twelve transactions are discussed above.

Event	Effect On Current Balance	Effect On Payoff Balance	Current Balance	Payoff Balance	Effect On Current Balance	Effect On Payoff Balance	Current Balance	Payoff Balance
	Electric SA				Non-billed Budget SA			
Starting balance	0	0	25	25				

Non-billed budget is activated	-25	0	0	25				
1 st scheduled payment is due					10	0	10	0
Payment					-10	-10	0	-10
2 nd scheduled payment is due					10	0	10	-10
Over-payment					-10	-10	0	-20
3 rd scheduled payment is due					10	0	10	-20
Transfer Adjustment					-10	-10	0	-30
4 th scheduled payment					10	0	10	-30
Underpayment					-7	-7	3	-37
Bill	0	33	0	58				
Bill completion (transfer adjustment)	0	-37	0	21	0	37	3	0
SA is stopped	21	0	21	21				

Note. In addition to synching the current and payoff balance, the SA being stopped is removed from the covered SAs collection. When the last covered SA is removed from the collection, the non-billed budget is also [stopped](#).

Financial Transactions For Unmonitored Non-billed Budgets

Some companies have the concept of non-billed budgets where the payments made by the customer are optional. This functionality is implemented as an unmonitored non-billed budget. Unmonitored non-billed budgets allow a customer to make optional prepayments towards a bill.

As explained previously, unmonitored non-billed budgets receive different financial treatment than monitored non-billed budgets. The financial transaction algorithms use the **Non-billed Budget Monitor** flag on the non-billed budget's SA type to create the appropriate financial transactions. The following table describes the differences in the financial treatment of monitored and unmonitored non-billed budgets.

System Event	Monitored Non-billed Budgets	Unmonitored Non-billed Budgets
When a non-billed budget is activated or an SA is added to a non-billed budget...	The system creates adjustments for all affected SAs to set the current balance equal to zero.	The current balances for covered SAs are not zeroed. The activate SA and non-billed budget maintenance processing do not create any adjustments if the non-billed budget is unmonitored.
When a scheduled payment is due...	An adjustment is created to increase the non-billed budget's current balance by the expected amount.	There is no change to the non-billed budget's current balance; it is always zero. The scheduled payment processing background process does not execute the NBB process scheduled payment algorithm .
When the payment is made...	The non-billed budget's current balance is reduced to zero and the non-billed budget's payoff balance reflects the accumulated credit from the payment.	Typically the payments are distributed to an excess credit (overpayment) SA. Refer to Distributing Payments for Unmonitored Non-billed Budgets for more information.
When a bill segment financial transaction is created...	The current amount is set to zero for SAs that are covered by the non-billed budget.	The covered SAs' current amounts are equal to their payoff amounts.
At bill completion...	Money from the non-billed budget is transferred to the covered SAs. Customers may receive a bill for information purposes, but they are not required to pay it.	Money from the overpayment SA is transferred to the account's SAs. The customer is still liable to pay the outstanding balance.
When the non-billed budget is stopped or an SA is removed from a non-billed budget...	The system creates adjustments for all affected SAs to synchronize their current balances with their payoff balances, thus removing the zero current balance and replacing it with the actual current balance.	There is no change to the current balances for covered SAs as the balances already reflect the actual current balance.

Unmonitored non-billed budgets also receive different credit and collections treatment. Refer to [Credit and Collections and Unmonitored Non-billed Budgets](#) for more information.

Contents

[Distributing Payments for Unmonitored Non-billed Budgets](#)
[Transferring Credit from Unmonitored Non-billed Budgets](#)

Distributing Payments for Unmonitored Non-billed Budgets

For non-billed budgets, payments are distributed according to the payment and overpayment algorithms on the customer class. The base package payment distribution algorithm applies a payment first towards any SAs that have overdue or current balances (refer to [Distributing A Payment](#) for more information). Since the unmonitored non-billed budget SA doesn't have a current balance, it is not considered by the payment distribution algorithm. If there aren't any SAs with a current balance, the overpayment distribution algorithm handles the remaining credit. You can elect to:

- Apply the overpayment to an excess credit SA. This is the method that we strongly recommend because all financial transactions are then a function of the normal payment, overpayment and billing processes.
- Apply the overpayment to the highest priority SA that is eligible for overpayment (as specified on the SA type). You can use this method to apply the overpayment to the unmonitored non-billed budget SA. If you use this method, you must also set up the system to [transfer the credit from the unmonitored non-billed budget](#).

Use An Excess Credit SA. We strongly recommend that payments for unmonitored non-billed budgets are distributed to an excess credit SA. In this case, the non-billed budget is just a shell to hold the covered SAs and recommend a payment schedule; all financial transactions are a function of the normal payment, overpayment and billing processes.

Refer to [Overpayment Segmentation](#) for a detailed discussion of overpayment distribution options.

Transferring Credit from Unmonitored Non-billed Budgets

If you distribute an overpayment to an unmonitored non-billed budget SA (i.e. the unmonitored non-billed budget maintains a credit balance instead of an overpayment SA), you must plug-in a bill completion algorithm on the SA type to transfer the credit balance to the covered SAs at bill completion time. The bill completion algorithm type ([BCMP-NB](#)) supplied with the base package transfers the credit balance to the covered service agreements when the bill is completed. Additionally, the **Adjustment Type (Xfer)** on unmonitored non-billed budget SA types should reference a FT algorithm of **$Payoff\ Amt = Adj / Current\ Amt = Adj$** to ensure that the credit is removed from both the current and payoff balances.

Warning! You must create your own SA stop algorithm type for correctly stopping an unmonitored non-billed budget that maintains a credit balance. The SA stop algorithm that is supplied with the base package does NOT transfer remaining credit from the unmonitored non-billed budget SA. (The base package SA stop algorithm transfers the remaining credit using the overpayment distribution algorithm on the customer type, which you have set up to transfer to back to the unmonitored non-billed budget.)

Designing Non-billed Budgets

The topics in this section describe functionality that you must consider when designing non-billed budgets.

Contents

- [Making SAs Eligible For Non-billed Budgets](#)
- [Designing Recommendation Rules](#)
- [Activating Non-billed Budgets](#)
- [Renewing Non-billed Budgets](#)
- [Expiring Non-billed Budgets](#)
- [Stopping Non-billed Budgets](#)
- [Automatic Payment and Non-billed Budgets](#)
- [Credit and Collections and Non-billed Budgets](#)
- [Credit and Collections and Unmonitored Non-billed Budgets](#)
- [Non-billed Budget Status](#)
- [Alerts For Non-billed Budgets](#)

Making SAs Eligible For Non-billed Budgets

A billable SA may be covered by a non-billed budget when its SA type is flagged as ***Eligible for Non-billed Budget***.

All SAs that are eligible for non-billed budgets should reference a bill segment type that uses the ***Payoff Amt = Bill Amt / Current Amt = Amt Due*** bill segment FT algorithm. This algorithm sets the SA's current amount to zero if it is covered by a monitored non-billed budget.

A list of the SAs covered by a non-billed budget is maintained with the non-billed budget SA. This list is used at bill completion to determine the financial transactions that should occur.

Designing Recommendation Rules

Users (and the renewal process) ask the system to recommend the scheduled payments for a non-billed budget. In general, this recommendation process must establish:

- The amount to be paid
- The dates on which the payments are due

We envision many different types of recommendation rules. For example:

- Recommend 26 scheduled payments to be made on a fortnightly basis that are due on Tuesdays.
- Recommend monthly payments that are due on the nearest workday after the 10th of the month.
- Recommend scheduled payments where the customer pays their annual charges in 10 out of 12 months where the payments are not due in November and December.
- Recommend bi-monthly payments where the payments are due on the first workday following the 5th and 20th of the month.

Additionally, the recommendation rules must determine how to handle any outstanding payoff balances for covered SAs. The true-up rule provided with the base package [payment schedule algorithm type](#) can ignore the payoff balance, divide the payoff balance evenly between the scheduled payments, or add the payoff balance to the first scheduled payment.

A recommendation rule comprises three elements:

- An algorithm to calculate an average daily amount (the [NBDA-DA](#) algorithm type provided with the base package uses premise billing history)

Calculating an Amount for Non-utility SAs. The algorithm type supplied with the base package only handles service-point oriented SAs. For example, the average daily amount algorithm calculates an average daily amount. For non-utility SAs, you must develop the appropriate algorithm types.

- Two algorithm types are provided with the base package to calculate a schedule of payments. [NBPS-MON](#) calculates a monthly schedule and [NBPS-PS](#) calculates a scheduled based on a specified number of days.
- A collection of default parameter values for the payment schedule algorithm type

A recommendation rule (based on the algorithm types provided with the base package) is illustrated below.

Rule: Weekly		
Avg Daily Amt Alg: Use premise history		
Pay Schedule Alg Type: Number of days		
Parameter	Value	Override
Number of days in period	7	Y
Number of payments	52	N
True-up rule	Spread	N

Weekly Payments Recommendation Rule

Additional Parameters. Not all of the parameters associated with the weekly payment schedule algorithm type are illustrated. Refer to the [NBPS-PS](#) algorithm type for a detail description of the parameters.

The default parameter values for the payment schedule algorithm type may change over time, so the collection contains an effective date. If default values are changed, these changes do not affect non-billed budgets already in effect. Existing non-billed budgets keep the parameter values that were used when the non-billed budget was started.

A user may override the default parameter values for the payment schedule algorithm type to customize the schedule if an override is allowed for a parameter. Additionally, a user may edit the payment schedule details at any time (provided the payment has not yet been processed).

The parameter values used for the recommendation rule are kept with the non-billed budget SA, so that any customized parameter changes can be re-applied to a renewed non-billed budget. For example, the parameter that determines the payment due day may default to the first of the month. To customize the schedule, this value may be changed to the fifth of the month. This amended value is kept with the non-billed budget SA to ensure that the renewed budget follows the same monthly schedule.

Note. Normally parameter values for an algorithm type are kept with the algorithm. Because the parameters may vary for each non-billed budget, the parameter values are kept with the SA in the case of non-billed budgets.

Refer to [Non-billed Budgets Recommendation Rule - Main](#) for information on creating recommendation rules.

Example Recommendation Rules

The following examples may be helpful in designing and implementing your recommendation rules.

Developing Your Own Payment Schedule Algorithms. The base package comes supplied with a [monthly payment schedule algorithm type](#). You can use this algorithm as an example when creating payment schedule algorithm types for your implementation.

Contents

- [Fortnightly Payments Recommendation Rule Example](#)
- [Monthly Payments Recommendation Rule Example](#)
- [Ten Out of Twelve Months Recommendation Rule Example](#)

Fortnightly Payments Recommendation Rule Example

The following diagram illustrates a recommendation rule where the customers pay every two weeks. The current balance for any covered SAs is added to the first payment.

Rule: Fortnightly		
Avg Daily Amt Alg: Use premise history		
Pay Schedule Alg Type: Number of days		
Parameter	Value	Override
Number of days in period	14	Y
Number of payments	26	N
True-up rule	Spread	N

Fortnightly Payments Recommendation Rule**Monthly Payments Recommendation Rule Example**

The following diagram illustrates a recommendation rule where the customers pay twice monthly on the first of the month. The current balance for any covered SAs is spread out over the scheduled payments.

Rule: Monthly		
Avg Daily Amt Alg: Use premise history		
Pay Schedule Alg Type: Monthly		
Parameter	Value	Override
Day of month	1	Y
Number of payments	12	N
True-up rule	Spread	N

Monthly Payments Recommendation Rule**Ten Out of Twelve Months Recommendation Rule Example**

The following diagram illustrates a recommendation rule where the customers pay one a month except for months during a holiday season (November and December). The current balance for any covered SAs is added to the first payment.

Rule: 10 of 12		
Avg Daily Amt Alg: Use premise history		
Pay Schedule Alg Type: X out of Y months		
Parameter	Value	Override
Total number of months	12	N
Months with no payment	11,12	Y
True-up rule	1 st pay	N

Ten Out of Twelve Months Recommendation Rule

Activating Non-billed Budgets

You can plug in an algorithm (of type [SACR-AT](#)) on the SA Creation system event on the non-billed budget SA type to automatically activate the non-billed budget (i.e. transition it from **pending start** to **active** status). If you don't use a SA Creation algorithm to activate the non-billed budget, it is activated the next time the [SA activation background process](#) runs.

When a non-billed budget is activated, you can perform special processing using an algorithm plugged in on the SA Activation system event on the non-billed budget SA type. The special processing can be developed to do anything that you would like, for example you could:

- Create a customer contact that with an appropriate letter template can generate a letter to inform the customer of their payment amount and payment schedule.
- Initiate the creation of a payment coupon book for a customer.

The system comes supplied with a sample algorithm type (called [SAAT-CC](#)) that simply creates a customer contact to indicate that the non-billed budget is activated.

Renewing Non-billed Budgets

A non-billed budget can be renewed either manually or via a background process. When the non-billed budget SA is created, the expiration date, renewal date and the recommendation rule used to create the initial budget are kept with the SA. A renewal flag on the non-billed budget SA type controls if a renewal is required, optional or not allowed. If renewal is required, a user must specify a renewal date when creating the service agreement. The renewal date is defaulted on to an SA based on the value of the **Days Before Expiration for Renewal** field on the SA type.

An algorithm on the SA type can customize the processing required to renew an SA.

The [SA renewal background process](#):

- Executes the SA renewal algorithm (specified on the SA type) when the renewal date is reached (i.e., it is on or before the process date). The base package comes with an algorithm type ([SARN-NB](#)) that determines the current recommendation rule for a non-billed budget and executes the associated payment schedule algorithm using the non-billed budget SA-specific parameter values to generate a new schedule. It returns new expiration and renewal dates.
- If the renewal algorithm is successful, the renewal and expiration date fields on the SA are updated with the new values.
- If the renewal process is not successful, a To Do list entry (of type [TD-SARN](#)) is created for the account and SA.

The new payment schedule that is returned from the renewal process for a non-billed budget is appended to the current schedule.

A user can manually launch the renewal process for a non-billed budget SA by clicking the **Renew NBB** button on the [non-billed budget maintenance page](#).

Expiring Non-billed Budgets

Non-billed budget service agreements may specify an expiration date. The [SA expiration background process](#) initiates the stop process for all pending start or active SAs where the expiration date is reached (before or on the process date).

Stopping Non-billed Budgets

When a non-billed budget stop is initiated, either on request or because it has expired and is not being renewed, the non-billed budget is transitioned to **pending stop** status. You can plug in an algorithm (of type [SAIS-ST](#)) on the SA Stop Initiation system event on the non-billed budget SA type to automatically finalize and stop the SA (i.e. transition it to **stopped** status). If you don't use a SA Stop Initiation algorithm, the non-billed budget is stopped the next time the [SA activation background process](#) runs.

To finalize a pending stop SA, the system first calls the stop SA algorithm plugged-in on the SA Stop system event on the SA type. The stop SA algorithm type ([SAST-NB](#)) supplied with the base package:

- Distributes any credit on the non-billed budget to the covered SAs (using the method described in [Distributing Non-billed Budget Credit](#))
- Distributes any excess credit remaining on the non-billed budget using the [overpayment distribution](#) algorithm for the account's customer class and the overpayment transfer adjustment type (specified as a parameter to the algorithm)
- Creates a trigger to cause the account to be reviewed by the account debt monitor
- Creates a customer contact (if the customer contact class and customer contact type parameters are populated)

Warning! If you do not plug in an SA stop algorithm that transfers the credit balance from the non-billed budget to its covered SAs (or an excess credit SA), the stopped non-billed budget may have a credit balance. You must then manually distribute this credit.

After the SA stop algorithm is finished, the SA stop processing performs the following steps if the SA type has a special role of non-billed budget:

- If the non-billed budget is monitored, create adjustments to synchronize the current and payoff balance of covered SAs using the **Adj. Type (Synch Current)** adjustment type from the covered SAs' SA types
- Remove the covered SAs from the non-billed budget
- Remove all the scheduled payments from the non-billed budget
- Create an adjustment to synchronize current and payoff on the non-billed budget SA using the **Adj. Type (Synch Current)** adjustment type from the non-billed budget's SA type

Note. Synchronizing current and payoff effectively sets the current amount to zero on the non-billed budget SA, as the payoff amount should have been reduced to zero by the distribution and overpayment processing in the algorithm for SA Stop.

Refer to [The Lifecycle Of A Service Agreement](#) for more information about how a *pending stop* SA is **stopped** and **closed**.

Automatic Payment and Non-billed Budgets

If a customer wants to pay their non-billed budget scheduled payments automatically, the account must be set up for automatic payment. In addition, the non-billed budget must indicate that automatic payment is being used.

Refer to [How To Set Up Automatic Payment For A Non-billed Budget](#) for more information.

When this is done, a background process referred to as **NBBAPAY** creates automatic payments on the scheduled payment date by calling the automatic payment creation algorithm plugged in on the installation record.

Note. You must also ensure that your auto pay creation algorithm supports non-billed budget scheduled payments. Note that the [APAY-CREATE](#) algorithm type supplied with the base package supports non-billed budget scheduled payments.

Credit and Collections and Non-billed Budgets

Unless the non-billed budget is [unmonitored](#), the [account debt monitor](#) (ADM) monitors a non-billed budget SA's current amount just as it does for any other SA. The [scheduled payment algorithm](#) creates a trigger to ensure that the account debt monitor reviews the account the next time it runs. The review date on the trigger record is set to the process business date.

A separate [debt class](#) is needed for non-billed budget SA types, thus allowing you to define collection class controls, debt criteria and collection process templates specifically for non-billed budgets. The debt criteria should be set up to trigger a collection process when the arrears amount exceeds \$0.01 for more than n payment periods plus the number of grace days that you want to allow.

The collection process template can perform any of the events in standard collection processes, such as sending letters to customers and creating severance processes. At a minimum, the collection process template should be set up to start a severance process for all service agreements in the debt class. (Since the debt class is specifically for non-billed budgets, the non-billed budget is the only SA that will be subject to a severance process.)

The non-billed budget severance process template should include the following event types:

- Populate a characteristic on the SA to indicate that the SA is broken. The base package comes supplied with a severance event algorithm type ([SVEV-NB](#)) that sets an SA characteristic to indicate that it was “severed”.
- [Expire Severance Agreement](#) to move the SA to the **pending stop** state.

When the system subsequently stops the non-billed budget, the system removes the covered SAs from the non-billed budget and synchronizes their current balances with their payoff balances. Since the SAs have current balances again, they are subject to the account debt monitor, which can start subsequent collection processes for any of the SAs that meet the debt criteria for their debt class.

Refer to [Stopping Non-billed Budgets](#) for a complete description of the events that occur when a non-billed budget is stopped.

Customers can catch up on their payments and avoid having their non-billed budget broken as long as their current balance doesn't violate the debt criteria for the non-billed budget's collection process.

Credit and Collections and Unmonitored Non-billed Budgets

If the **Non-billed Budget Monitor** flag on the non-billed budget's SA type is set to **Unmonitored**, the [NBB Scheduled Payment Processing](#) background process does not create a trigger for the account debt monitor. Additionally, the non-billed budget's current amount is always equal to zero, so it never violates any debt collection criteria. We recommend using a debt class that has the **Eligible for Collection** flag turned off, such as the N/A debt class.

For unmonitored non-billed budgets, the current balance is kept on the covered SAs so they are subject to the account debt monitor and any debt criteria for their SA types' debt classes. For more information, refer to [Financial Transactions for Unmonitored Non-billed Budgets](#).

Non-billed Budget Status

A non-billed budget SA's status is just like any other SA's status. In addition, you can use a characteristic to keep an explicit status relevant to the non-billed budget:

- A base package severance event algorithm type ([SVEV-NB](#)) creates a characteristic value to indicate if a non-billed budget is “broken” (i.e. stopped as a result of a severance process).
- A base package break non-billed budget algorithm type ([NBBR-BRK](#)) creates a characteristic value to indicate if a non-billed budget is “canceled” (i.e. manually stopped by a user).

An active non-billed budget implicitly has a “kept” status (i.e. all scheduled payments have been made).

Alerts For Non-billed Budgets

The system provides [alerts](#) to highlight the existence of non-billed budgets. These alerts are important to assist the customer service representatives:

- An alert is displayed if the account has a non-billed budget that is not stopped (e.g., **pending start**, **active** or **pending stop**).
- When a user denies a non-billed budget (for whatever reason), the user should create a [customer contact](#) with a given [customer contact type](#). This type of alert prevents the customer from shopping around. An existing alert algorithm type ([CC BY TYPCL](#)) can highlight these customer contacts.
- For customers who are permanently forbidden from having a non-billed budget, the user should put a permanent [alert on the account](#).
- Use an algorithm to highlight cancelled or severed non-billed budgets with an entry in the alert zone. The algorithm type to do this is not provided. Use [PP BY STATUS](#) and [CCAL-WF](#) as examples of how to create this type of algorithm.

For more information about introducing alert conditions on Control Central, refer to [Installation Options - Algorithms](#).

Non-billed Budget Recommendation Rule

Recommendation rules are used to recommend scheduled payments for non-billed budgets. For information about designing recommendation rules, refer to [Designing Recommendation Rules](#).

To define recommendation rules, navigate to **Admin Menu, NBB Recommendation Rule**.

Description of Page

Enter an easily recognizable **Recommendation Rule** code and **Description** for each recommendation rule.

Specify the **Average Daily Amount Algorithm** used to calculate the average daily amount for this recommendation rule.

Specify the **Payment Schedule Algorithm Type** used to create the recommended payment scheduled for non-billed budgets that use this recommendation rule. The Payment Schedule Algorithm Type cannot be modified if a non-billed budget SA that is not stopped or cancelled is using this recommendation rule.

Payment Schedule Parameters enables you to define collections of default parameter values for the payment schedule algorithm type that are effective dated. For each collection:

- **Effective Date** defines the date on which the collection of parameter values becomes effective.
- **NBB Rule Parameter Value** specifies the default value of each parameter supplied to the algorithm. Note that the [payment schedule algorithm type](#) controls the number and type of parameters.
- **Override Flag** indicates whether the user can override the default value for the parameter.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference [CI_NB_RULE](#).

Setting Up The System To Enable Non-billed Budgets

The above topics provided background information about how non-billed budgets are supported in the system. The topics in this section describe how to set up the system to enable non-billed budget functionality.

Example Setup. This section describes typical non-billed budget configurations. Your set up and configuration may differ depending on your business needs. This section is provided for guidance only. Read the descriptions of non-billed budget functionality above to understand the implications of the described setup.

Contents

- [NBB Distribution Codes](#)
- [NBB Adjustment Types](#)
- [NBB Adjustment Type Profiles](#)
- [NBB Characteristic Types](#)
- [NBB Customer Contact Class And Types](#)
- [NBB Algorithms](#)
- [NBB Debt Class And Collection Process](#)
- [SA Types for SAs Covered by NBBs](#)
- [NBB Recommendation Rules](#)
- [Non-billed Budget SA Types](#)
- [NBB Background Processes](#)

NBB Distribution Codes

You must set up a Non-billed Budget Clearance [distribution code](#) that is used on the non-billed budget SA type to credit non-billed payments.

NBB Adjustment Types

Non-billed Budget Financial Transaction Algorithms. The non-billed budget adjustment types use the standard FT algorithm types that are provided with the base package. If you have not yet defined algorithms for these types in your system, do so before creating the non-billed budget adjustment types.

The following adjustment types are needed:

- Add SA To Non-billed Budget. This adjustment type should reference an FT algorithm of ***Payoff Amt = 0 / Current Amt = Adj Amount (no GL)***. Because the adjustment affects the current balance only, there is no entry in the GL. This adjustment type is referenced on **Adjustment Type (Current=0)** on SA types that are eligible for non-billed budgets. Note that this adjustment type is never used if the SA is added to an unmonitored non-billed budget.

- Bill Complete for Monitored Non-billed Budget. This adjustment type should reference a distribution code used for balance transfer clearing. It should reference a FT algorithm of **$\text{Payoff Amt} = \text{Adj} / \text{Current Amt} = 0$** . This adjustment type is referenced on **Adjustment Type (Xfer)** on SA types that are monitored non-billed budgets.
- Synch Balance for Non-billed Budget. This adjustment type should be set up for **Sync. Current Amount** and should reference a FT algorithm of **$\text{Payoff Amt} = 0 / \text{Current Amt} = \text{Adj Amount (no GL)}$** . This adjustment type is referenced on the **Adj. Type (Synch Current)** on non-billed budget SA types. It is used when a non-billed budget SA is stopped to synch the current balance with the payoff balance.
- Scheduled Payment. This adjustment type should reference an FT algorithm of **$\text{Payoff Amt} = 0 / \text{Current Amt} = \text{Adj Amount (no GL)}$** . Because the adjustment affects the current balance only, there is no entry in the GL. This adjustment type is referenced on the non-billed budget process scheduled payment algorithm.
- Overpayment Transfer. This adjustment type should reference a transfer distribution code and a FT algorithm of **$\text{Payoff Amt} = \text{Adj} / \text{Current Amt} = \text{Adj}$** . This adjustment is referenced on the non-billed budget process scheduled payment algorithm and the stop non-billed budget algorithm. Note that if there is already a transfer adjustment type created in your system, you do not need to create a new one.

If you are setting up an unmonitored non-billed budget that maintains a credit balance (as opposed to maintaining the credit balance on an overpayment SA), you need to create an adjustment type for Bill Complete for Unmonitored Non-billed Budget. (Refer to [Transferring Credit from Unmonitored Non-billed Budgets](#) for more information.) The adjustment type should reference a transfer distribution code and a FT algorithm of **$\text{Payoff Amt} = \text{Adj} / \text{Current Amt} = \text{Adj}$** . This adjustment type is referenced on **Adjustment Type (Xfer)** on SA types that are unmonitored non-billed budgets.

NBB Adjustment Type Profiles

Create an adjustment type profile for non-billed budgets that references the following adjustment types:

- Bill complete for monitored non-billed budget
- Bill complete for unmonitored non-billed budget (if used)
- Synch balance for non-billed budget
- Scheduled payment
- Overpayment transfer

Create an adjustment type profile for eligible SAs that references the following adjustment types:

- Add SAs to monitored non-billed budgets
- Bill complete for monitored non-billed budget
- Bill complete for unmonitored non-billed budget (if used)

Bill Complete Adjustment Types. Because the bill complete adjustment types transfer amounts between two SAs, they must be in profiles for both non-billed budget and eligible SA types.

Overpayment Transfer Adjustment Type. The overpayment transfer adjustment type created above is used to transfer funds from an excess credit SA to a non-billed budget when the scheduled payment is processed. It is also used to transfer excess funds from a non-billed budget that is being closed to an excess credit SA. The transfer adjustment should therefore be added to an adjustment type profile that is referenced on the excess credit SA type.

NBB Characteristic Types

Create a non-billed budget status characteristic type that specifies **Service Agreement** as its characteristic entity. The characteristic type should include the following predefined values:

- Non-billed Budget Canceled
- Non-billed Budget Severed

NBB Customer Contact Class And Types

Create a non-billed budget customer contact class. The contact class may include the following customer contact types:

- Non-billed Budget Activate
- Non-billed Budget Renewal
- Non-billed Budget Stop

Customer Contact Letters. If you want to send letters to your customers when a contact of any of these types is created, you must create an appropriate [letter template](#) and attach it to the contact type.

NBB Algorithms

You must define the following [algorithms](#):

- On [NBB recommendation rule](#), the Non-billed Budget Daily Amount Calculation. Refer to the [NBDA-DA](#) algorithm type for more information about the base package algorithm.
- On [SA type](#):
 - Non-billed Budget Process Scheduled Payment. This algorithm has parameters that must reference the Scheduled Payment and Overpayment Transfer adjustment types defined above. Another parameter references an overpayment SA type (that you may need to create if there is not already one in your system). Refer to the [NBPA-PS](#) algorithm type for more information about the base package algorithm.
 - Non-billed Budget SA Renewal. Refer to the [SARN-NB](#) algorithm type for more information about the base package algorithm.
 - Break Non-billed Budget SA. Refer to the [NBBR-BRK](#) algorithm type for more information about the base package algorithm.

- SA Activation - automatically activate SA (if you want to automatically activate non-billed budgets when they are created). Refer to the [SACR-AT](#) algorithm type for more information about the base package algorithm. Note that this same algorithm may be used on many SA types.
- SA Activation - create customer contact (if you want the system to create a customer contact when NBB SAs are activated). Refer to the [SAAT-CC](#) algorithm type for more information about the base package algorithm.
- SA Stop - automatically stop SA (if you want to automatically transition non-billed budgets from pending stop to stop when their stop is initiated). Refer to the [SAIS-ST](#) algorithm type for more information about the base package algorithm. Note that this same algorithm may be used on many SA types.
- SA Stop - Stop Non-billed Budget. Refer to the [SAST-NB](#) algorithm type for more information about the base package algorithm.
- Bill Completion - Non-billed Budget Credit Transfer. Refer to the [BCMP-NB](#) algorithm type for more information about the base package algorithm.
- On [bill segment type](#), the Bill FT Algorithm (for all SAs that are eligible for NBB). Refer to the [BSBF-BA](#) algorithm type for more information about the base package algorithm. Note this is the standard bill FT algorithm type used for common bill transactions. It supports bill FT for non-billed budgets and other SA types. You only need to create it if it doesn't already exist in your system.
- On [severance event type](#), an algorithm for Non-billed Budget Severance. Refer to the [SVEV-NB](#) algorithm type for more information about the base package algorithm.

Payment Schedule Algorithm Types. For non-billed budget payment schedule algorithm types, you need to define the algorithm types (if you add your own algorithm types). You do not need to define algorithms because the parameter values for the algorithm are defaulted on the recommendation rule and stored with the non-billed budget SA. (Normally the algorithm holds the parameter values.) Refer to the [NBPS-MON](#) algorithm type for more information about the base package payment schedule algorithm.

NBB Debt Class And Collection Process

Set up a separate debt class, collection class control, collection process template and severance process template for non-billed budgets according to the information in [Credit and Collections and Non-billed Budgets](#).

SA Types for SAs Covered by NBBs

You must modify the SA type for any SAs that you want to allow to be covered by a non-billed budget.

- Verify that the specified **Bill Segment Type** (on the Billing page) references a financial transaction algorithm to set the current amount to zero for monitored non-billed budgets. The FT creation algorithm type [BSBF-BA](#) (i.e., **Payoff Amt = Bill Amt / Current Amt = Amt Due**) supplied with the base package sets the current amount to zero for SAs that are covered by monitored non-billed budgets, though this is not evident by the name. (For SAs that are not covered by a non-billed budget, the current amount is equal to the amount due or the recurring charge.) Refer to [Billing For SAs Covered By The Non-billed Budget](#) for more information.
- Set the **Eligible for Non Billed Budget** flag (on the Billing page) to **Eligible for Non-billed Budget**.
- Populate the **Adjustment Type (Current = 0)** (on the Detail page) to indicate the adjustment to be used to zero out the current amount on the covered SAs when the non-billed budget SA is activated. Use the Add SA To Non-billed Budget [adjustment](#) created above. This adjustment type is only called for SAs that are covered by a monitored non-billed budget.
- Reference an **Adjustment Type Profile** (on the Adjustment Profile page) that includes the Add SA To Non-billed Budget adjustment type referenced in the **Adjustment Type (Current = 0)** field above.
- If not already specified (for write off), an **Adj. Type (Synch Current)** (on the Main page) is also required. It is used to synchronize (make equal) the current amount with the payoff amount when the SA is removed from (i.e. no longer covered by) a non-billed budget.

NBB Recommendation Rules

Set up any [NBB Recommendation Rules](#) that you want to be available for your non-billed budget SAs. Use any Non-billed Budget Daily Amount Calculation algorithms defined above. Also use the Non-billed Budget Monthly Payment Schedule ([NBPS-MON](#)) algorithm types and specify the default parameters.

Non-billed Budget SA Types

You must set up a [SA types](#) for your non-billed budget service agreements. You may need multiple non-billed budget SA types if you have different business rules for different types of non-billed budgets, for example, if you have both monitored and unmonitored non-billed budgets or if you support non-billed budgets with different renewal requirements. The following points provide guidelines for creating a non-billed budget SA type.

Contents

- SA Type - Main (NBB)
- SA Type - Detail (NBB)
- SA Type - Billing (NBB)
- SA Type - Rate (NBB)
- SA Type - SP Type (NBB)
- SA Type - Adjustment Profile (NBB)
- SA Type - Credit and Collections (NBB)
- SA Type - Algorithms (NBB)
- SA Type - NBB Recommendation Rule (NBB)

SA Type - Main (NBB)

Service Type should reference something like "Miscellaneous Service".

Distribution Code should be the one you set up to book credits for non-billed budgets.

Revenue Class should be set to *N/A*. (Revenue classes are not applicable because non-billed budgets do not apply a rate and revenue classes are only relevant for SA types that use a rate.)

The **Payment Segment Type** should reference the *Normal Payment*.

Do Not Overpay should be on. Any excess payments should go to the overpayment SA, not the non-billed budget SA.

Late Payment Charge is not applicable and should not be turned on because non-billed budgets are not billed.

Adj. Type (Synch Current) should reference the Synch Balance for Non-billed Budget adjustment type created above.

SA Type - Detail (NBB)

- **Special Role** is *Non-billed Budget*.
- **Adjustment Type (Xfer)** must be populated to indicate the adjustment to be used for transferring accumulated credit from the non-billed budget SA to the covered SAs. This field is not used for unmonitored non-billed budgets.
- **Renewal** may be optional, not allowed, or required depending on your business processes.
- If Renewal is required, specify the **Days Before Expiration for Renewal**.
- **Non-billed Budget Monitoring** must indicate whether the non-billed budget is monitored by the account debt monitor.

SA Type - Billing (NBB)

Non-billed budget SAs do not get billed, so the **Eligible for Billing** flag should be off.

Additionally, the **Characteristic Premise Required** should not be checked for non-billed budgets.

SA Type - Rate (NBB)

Non-billed budget SAs do not use rates, so the **Rate Required** flag should be off.

SA Type - SP Type (NBB)

Non-billed budget SAs do not have service points, so the **Service Points Required** flag should be off.

SA Type - Adjustment Profile (NBB)

Adjustment Type Profile should reference the [adjustment type profile](#) for non-billed budgets (set up above).

SA Type - Credit and Collections (NBB)

The credit and collections information should reference a **Severance Process Template** that calls the Non-billed Budget Severance algorithm created above and expires the non-billed budget SA. The **Debt Class** should reference an appropriate value consistent with your credit and collections rules. Typically, monitored non-billed budgets should be in their own debt class. Unmonitored non-billed budgets should have a **Debt Class** that is not *Eligible for Collection*.

The **Write Off Debt Class** is not applicable because the non-billed budget contains no debt to be written off. Reference a debt class such as N/A. Refer to [Defining Credit & Collections Options](#) for more information.

SA Type - Algorithms (NBB)

The SA type [algorithms](#) defined above must be set up:

- The Non-billed Budget Credit Transfer algorithm created above should be specified for the Bill Completion system event (not used on unmonitored non-billed budgets).
- The Break Non-billed Budget SA algorithm created above should be specified for the Break NBB SA system event.
- The Non-billed Budget Process Scheduled Payment algorithm created above should be specified for the Process NBB Scheduled Payment system event (not used on unmonitored non-billed budgets).
- The Non-billed Budget SA Renewal algorithm created above should be specified for the SA Renewal system event.
- If you created the Automatic SA Activation algorithm above, it should be specified for the SA Creation system event.
- The Non-billed Budget SA Activation algorithm created above should be specified for the SA Activation system event.
- If you created the Automatic SA Stop algorithm above, it should be specified for the SA Stop Initiation system event.
- The Stop Non-billed Budget algorithm created above should be specified for the SA Stop system event.

SA Type - NBB Recommendation Rule (NBB)

Add the recommendation rules defined above that are valid for SAs of this type. Also, indicate which recommendation rule should be used as the default.

NBB Background Processes

Ensure that the following background processes are scheduled:

- Non-billed Budget Scheduled Payment Processing ([NBBPS](#))
- Non-billed Budget Scheduled Payment Automatic Payment Create ([NBBAPAY](#))
- Service Agreement Renewal ([SARENEW](#))
- Stop Expired Service Agreements ([SAEXPIRE](#))

Defining Quotation Options

This section describes tables that must be set up before quotations can be created.

For more information, refer to [The Big Picture of Quotations](#).

Contents

- [Setting Up SA Types For Quotes](#)
- [Setting Up Quote Route Types](#)
- [Setting Up Terms and Conditions](#)
- [Setting Up Decline Reasons](#)
- [Setting Up Customer Classes For Quotes](#)

Setting Up SA Types For Quotes

The topics in this section describe additional setup responsibilities required on SA types that can have proposal service agreements.

Assumption. We have assumed that you've already designed your SA types for the services that you sell. If you haven't done this, refer to [Defining Service Agreement Types](#).

Contents

- [Enabling The Automatic Generation Of Billing Scenarios](#)
- [Enabling The Generation Of Simulated Bill Segments](#)
- [Enabling The Creation Of A Real SA When A Quote Detail Is Accepted](#)

Enabling The Automatic Generation Of Billing Scenarios

As described under [Proposal SAs Contain Billing Scenarios And Template Consumption](#), a proposal SA requires billing scenarios before a quote detail can be generated. The system will automatically create billing scenarios when a proposal SA is created if you plug-in the appropriate **Proposal SA Creation** algorithm on each such [SA type](#). Refer to [PSAC-CBS](#) for an example algorithm (note, this algorithm can be used to create billing scenarios for both interval and non-interval service agreements).

Enabling The Generation Of Simulated Bill Segments

As described under [Creating Quotes And Quote Details](#), in order for the system to generate simulated bill segments for a proposal SA, you must plug-in a **Proposal SA Bill Segment Generation** algorithm on the proposal SA's [SA type](#). Refer to [CBSP-AR](#) for an example algorithm that creates a simulated bill segments for each billing scenario linked to the proposal SA by calling rate application.

Interval pricing service agreements. If you have proposal SAs that require the derivation of interval profiles from other interval profiles, you must also plug-in another **Proposal SA Bill Segment Generation** on the interval [SA type](#). Refer to [CBSP-IDDRV](#) for an example algorithm that performs derivation. When you plug-in this type of algorithm, don't forget to use a sequence number less than the one that generates the simulated bill segments (see above); otherwise, the derived interval profiles won't exist when rate application is called.

Enabling The Creation Of A Real SA When A Quote Detail Is Accepted

As described under [Accepting / Declining Quote Details](#), in order for the system to create a real SA when a proposal SA is accepted, you must plug-in a **Proposal SA Acceptance** algorithm on the proposal SA's [SA type](#). Refer to [PSAA-PS](#) for an example algorithm that creates a real SA by copying the proposal SA.

Interval pricing service agreements. If the service agreement has SA-specific interval profiles, time-of-use maps, or contract options, it's important that you setup your **Proposal SA Acceptance** algorithm to create fresh interval profiles, time-of-use maps and contract options when the real SA is created (there's a parameter on [PSAA-PS](#) that will do this). Refer to [Proposal SAs And Interval Consumption](#) for more information.

Setting Up Quote Route Types

Quote route types control how quotes are [routed](#) to customers and prospects. To define a quote route type, open **Admin Menu, Quote Route Type**.

Refer to [Printing Quotes](#) for more information about how quotes are routed to customers and prospects.

Description of Page

Enter a unique **Quote Route Type** and **Description** for every quote route type.

Quote Routing Method controls the type of information that may be defined when the respective **Quote Route Type** is selected on [Account - Person Information](#). The following options are available:

- **Postal.** Use this method if the routing is via the postal service.
- **Fax.** Use this method if the routing is via fax.
- **Email.** Use this method if the routing is via email.

Note. The values for **Quote Routing Method** are customizable using the [Lookup](#) table. This field name is **QTE_RTG_METH_FLG**.

- The next two fields control how quotes that are routed using this method are [printed](#) (both in batch and online).
 - Use **Batch Control** to define the process that creates the flat file that is passed to your quote printing software. Refer to [Technical Implementation of Printing Quotes In Batch](#) for more information about these processes.
 - Use **Extract Algorithm** to define the plug-in that constructs the “flat file records” that contain the information merged onto quotes routed using this method. Refer to [Printing Quotes](#) for more information.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference [CI_QTE RTE TYPE](#).

Setting Up Terms and Conditions

As described under [Legal Terms and Conditions May Be Specified On SAs](#), your [SA type start options](#) can reference terms and conditions (T&C's) that should be defaulted onto new service agreements (both real and proposal). Each T&C is identified with a terms and condition code. To define a terms and conditions code, open **Admin Menu, Terms and Conditions**.

T&C print order. The value of the T&C code controls the order in which the T&C appears on the printed quote. This means you should assign these codes in some type of structured format (e.g., 01...) if you would like them to appear in a certain order.

Description of Page

Enter a unique **Terms and Condition** code and **Description**. Use **Text** to describe the exact terms.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference [CI_TC](#).

Setting Up Decline Reasons

A proposal SA decline reason must be supplied when a proposal SA is declined. Open **Admin Menu, Proposal SA Decline Reason** to define your reason codes.

Description of Page

Enter an easily recognizable **Decline Reason** and **Description** for each proposal SA decline reason.

Where Used

Decline reasons are used when a [quote detail is declined](#).

Setting Up Customer Classes For Quotes

An optional plug-in spot exists on [customer class](#) where you can introduce additional logic to be executed when a quote is completed for an account that belongs to this customer class. The base package comes supplied with a sample algorithm that creates a workflow process when a quote is completed (refer to the algorithm [QTEC-WP](#) for more information).

Defining Case Management Options

Case management functionality is a highly configurable tool your organization can use to manage many situations, including (but certainly not limited to) the following:

- a high-bill complaint,
- a bankruptcy,
- an inspection of a premise,
- a customer's request for literature,
- an application for new service,
- a contractor's request to extend a line,
- a customer's rejection of a quote,
- a customer's request to change service providers on a future date,
- the processing of a market message in a deregulated environment,
- ... (the list is only limited by your time and imagination)

Obviously the steps involved in the resolution of the above cases are very different. The topics in this section describe how to configure the system to manage your cases as per your organization's desires.

Separate module. Please note that the Case Management functionality is associated with a separate module. If the **Case Management** module is not applicable to your business you may turn it off. Refer to [Turn Off A Function Module](#) for more information.

Refer to [Case Management](#) for a description of how end-users use cases.

Contents

[The Big Picture Of Cases](#)
[Setting Up Case Management Options](#)

The Big Picture Of Cases

The topics in this section provide background information about how to configure the system to support your case management requirements.

Contents

[Case Type Controls Everything](#)
[Scripts and Cases](#)
[To Do's and Cases](#)
[Examples of Case Types](#)

Case Type Controls Everything

Whenever a user creates a case, they must specify the type of case (e.g., high-bill complaint, literature request, etc.). The case type controls how the case is handled.

Case types hold the business rules that control cases. Since these business rules can sometimes be quite complicated, setting up case types requires planning and foresight. The topics in this section describe the type of business rules that can be configured on your case types.

Contents

- [Person / Account / Premise Applicability](#)
- [Contact Information Applicability](#)
- [Business Object Association](#)
- [Additional Information](#)
- [Access Rights](#)
- [Lifecycle](#)
- [Status-Specific Business Rules](#)
- [Responsible User Applicability](#)

Person / Account / Premise Applicability

Some types of cases may be person-oriented, others may be premise-oriented, and still others may be account-oriented. For example:

- Cases used to keep track of a literature request would reference the person who requested the literature.
- Cases used to keep track of the inspection of a property would reference the premise being inspected.
- Cases used to keep track of a high-bill complaint would reference the account associated with this bill(s) being disputed.

When you set up a case type, you define if its cases must reference a person, account, and/or premise. Note, any combination of these objects is permitted on a case.

Contact Information Applicability

When a case is created, you may want to keep track of how to contact its originator. For example, you may want to record the originator's email address or phone number. When you set up a case type, you define if contact information is required, optional or not allowed on its cases.

Business Object Association

A case type may reference a Business Object, which serves as a link between cases of that type and the options that are associated with the business object.

Additional Information

Some of your cases may require additional information (in the form of [characteristics](#)). For example, a high-bill complaint may require at least one bill. When you set up a case type, you can define the additional fields that are required. In addition, you can define default values for these fields.

The case functionality also allows you to require characteristics when a case enters a given state. Refer to [Required Fields Before A Case Enters A State](#) for the details.

Requiring supporting documents. Because any [type of characteristic](#) can be referenced on a case, you can require references to supporting documents by requiring a **file location** characteristic.

Access Rights

You can take advantage of the system's [security](#) to restrict cases of a given type to certain users. For example, you can restrict high-bill complaints to specific user groups.

The following points describe how to implement this type of security:

- Create an [application service](#) for each type of case you need to secure
- Define the access modes **Add**, **Inquire** and **Change** for each application service
- Define the applicable application service on each case type
- Link the appropriate [user groups](#) to each application service
 - For user groups that are allowed to add cases of a given type, define **Add** as a valid access mode.
 - For user groups that are allowed to view cases of a given type, define **Inquire** as a valid access mode
 - For user groups that are allowed to change cases of a given type, define **Change** as a valid access mode

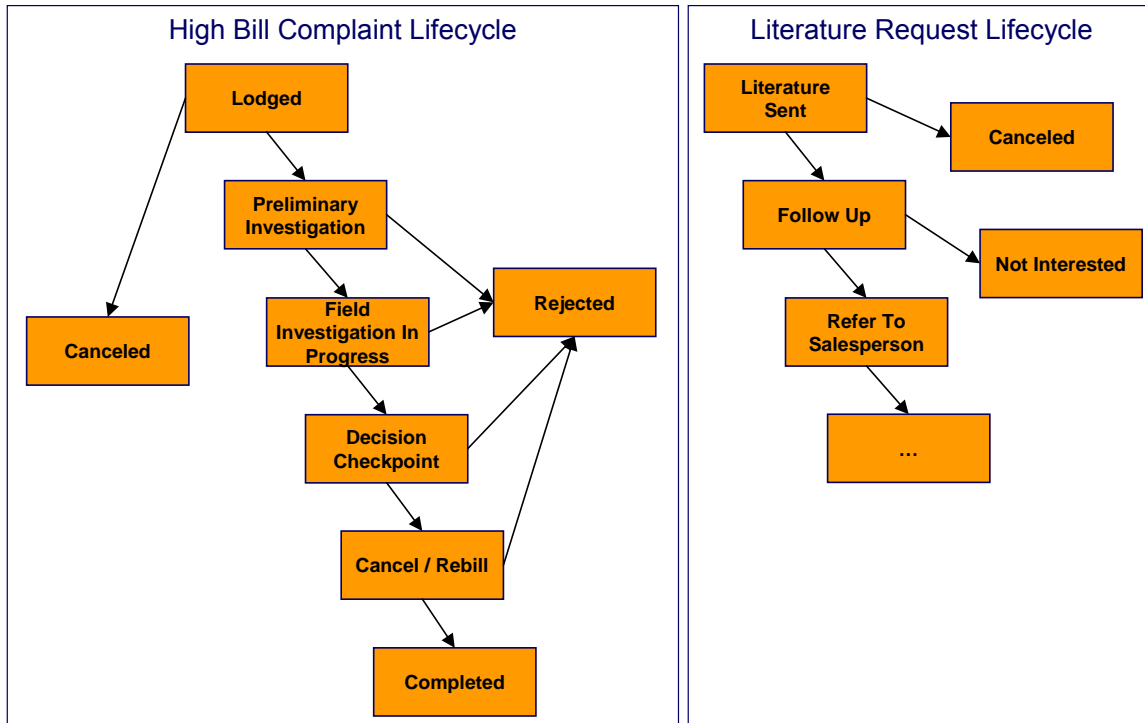
If you restrict access to a case type's cases, you can further restrict which users can work on cases given the status of the case. Refer to [Which Users Can Transition A Case](#) for more information.

Restricting access to cases is optional. If you don't specify an application service on a case type, all users (who have access to the case transaction) may access its cases.

Lifecycle

Many objects in the system have predefined lifecycles whose rules are governed by the base-package and cannot be changed. For example, a service agreement starts out in the **Pending Start** state and eventually becomes **Closed** when it's been final billed (and paid). You can't change the system to allow a service agreement to start its life in the **Closed** state.

The lifecycle of cases is not governed by the base package. Rather, you define the lifecycle of your cases when you set up their case types. Examine the following diagrams; the one on the left shows the potential lifecycle of a case that manages a high-bill complaint, the one on the right shows the potential lifecycle of a case that manages a customer's literature request.



Potential Lifecycles Of Two Types Of Cases

Just examples. The above lifecycles are just examples. When you set up your case types, you must define the valid states for your case type.

The topics that follow describe important concepts that are illustrated in the above diagrams.

Contents

- Valid States versus State Transition Rules
- Transitory States
- One Initial State and Multiple Final States
- Allowing A Case To Be Reopened
- Make Sure To Have A Canceled State
- Buttons Are Used To Transition A Case From Status To Status
- State Transitions Are Audited
- Reports and Analytics Highlight Productivity

Valid States versus State Transition Rules

The orange boxes in the above diagram show the potential valid states a given case can have. The lines between the boxes indicate the state transition rules. These rules govern the states a case can move to while it's in a given state. For example, the above diagram indicates a high bill complaint that's in the **Lodged** state can be either **Canceled** or moved into the **Preliminary Investigation** state.

When you set up a case type, you define both its valid states and the state transition rules.

Transitory States

You can define a state in a case type as **Transitory** if you do not wish the case to exist in a particular state. For example, let's assume that an algorithm is associated with the **Decision Checkpoint** state (Enter Processing) that would automatically determine the next state for the case (i.e. **Cancel/Rebill** or **Reject**) and also contains logic to transition the case accordingly. In this scenario, you may not ever want the case to exist in the **Decision Checkpoint** state, so that a user won't ever see a high bill complaint in that state. If the other states were marked as **non-transitory**, and an error were to occur during the transition from **Decision Checkpoint** to **Cancel/Rebill**, the case would roll back any changes to data made in the **Cancel/Rebill** (Enter Processing) state along with the changes made in the **Decision Checkpoint** state, and would end up in the **Field Investigation In Progress** state - the last non-transitory state prior to **Decision Checkpoint**.

One Initial State and Multiple Final States

When you set up a case type's states, you must pick one as the initial state. The initial state is the state assigned to new cases of a given type. For example, high-bill complaint cases have an initial state of **Lodged**, whereas literature request cases have an initial state of **Literature Sent**.

You must also define which statuses are considered to be "final". When a case enters a "final" state, it is complete and no further action is necessary. You might want to think of the "final" states as the potential outcomes of a case. For example, a high-bill complaint has potential outcomes of **Completed**, **Rejected**, and **Canceled**.

The "final" states are used by the system to differentiate between open and closed cases. For example, an alert highlights when the person / account / premise in context has open cases (this alert only exists if you've plugged-in the appropriate installation [alert](#)).

Allowing A Case To Be Reopened

You can set up your state transition rules to allow a case to be reopened (i.e., to be moved from a final state to a non-final state). Neither of the above examples allows this, but it is possible if you configure a case type accordingly.

Make Sure To Have A Canceled State

The system does not allow you to delete a case. Therefore, if you want to support logical deletion, you should have a status of **Canceled** early in a case type's lifecycle. Doing this allows a user to cancel (i.e., logically delete) a case.

Cancel reason. You might want to consider setting up your case types to require a cancel reason (in the form of a [predefined value characteristic](#)) when a user cancels a case. Refer to [Required Fields Before A Case Enters A State](#) for more information.

Buttons Are Used To Transition A Case From Status To Status

When a case is displayed on [Case - Main](#), a separate button is shown for each state into which the case can be transitioned. For example, a high-bill complaint case that is in the **Lodged** state would show two buttons: **Start Investigation** and **Cancel**. If the user presses the **Start Investigation** button, the case is transitioned to the **Preliminary Investigation** state. If the user presses the **Cancel** button, the case is moved to the **Canceled** state.

You may define the text displayed on the button differently for each state transition. This allows the action description to be varied according to the previous status. For example, the button to transition from **New** to **Active** may be labeled **Activate**, but the button to change from **Closed** to **Active** may be labeled **Reactivate**.

Refer to [Which Users Can Transition A Case](#) for instructions describing how to restrict users to specific actions.

State Transitions Are Audited

The system maintains an audit trail whenever a case transitions from one state to another. This audit is shown in the case's [log](#).

Reports and Analytics Highlight Productivity

When you set up a case type's lifecycle, keep in mind that several reports and analytics highlight how long it took cases to transition into a state. For example, you can use a report to see how long it took high-bill complaints to be completed (or initially actioned or ...). Refer to the [Reports](#) chapter for the details of case reports.

Status-Specific Business Rules

As described in [Lifecycle](#), when you set up a case type, you define the possible states its cases can pass through. The following topics describe business rules that can be configured for each state.

Contents

- [A Script That Helps A User Work Through A Case](#)
- [Required Fields Before A Case Enters A State](#)
- [Validation Before A Case Enters A State](#)
- [Additional Processing When Entering A State](#)
- [Validation Before A Case Exits A State](#)
- [Additional Processing When Exiting A State](#)
- [Automatic Transition Rules](#)
- [Script Launching Option](#)
- [Which Users Can Transition A Case Into A State](#)

A Script That Helps A User Work Through A Case

You can define a [Business Process Assistant script](#) that helps a user work a case while it's in a given state. For example, when you set up the **Preliminary Investigation** state for the high-bill complaint case type, you can define a script. A user can then easily launch this script to help them work through a case in this state.

Please keep the following in mind when you're designing how to integrate BPA scripts with your cases:

- You can have a different script for each state. For example, you could develop a script to help a user work on a case while it's in the **Preliminary Investigation** state and a different script to help them work in a case while it's in the **Decision Checkpoint** state.
- Rather than make a user launch a script by pressing a hyperlink on the [case page](#), you can have the system automatically launch the script while the case is in a given state. Refer to [Script Launching Option](#) for more information.
- You can also have the system automatically launch a script when a user selects a To Do entry. Refer to [Launching Scripts When To Do Entries Are Selected](#) for more information.

Refer to [Scripts and Cases](#) for more information about how to streamline your case processing with scripts.

Required Fields Before A Case Enters A State

You can define additional fields (i.e., characteristics) that are required before a case can enter a given state. For example,

- You can indicate a high-bill complaint must reference at least one bill before it enters the **Preliminary Investigation** state.
- You can indicate a case must reference a cancel reason before it enters the **Canceled** state.

You do this by indicating that [characteristics](#) (that were optional when the case was added) are required when a case enters a given state.

Validation Before A Case Enters A State

You can define validation that executes before a case can enter a given state. For example, you can indicate the case must have been assigned a responsible user before it can enter the **Preliminary Investigation** state. This validation logic is held in algorithms that are plugged in on the case type and therefore you can define any type of validation.

Additional Processing When Entering A State

You can define additional processing that should happen when a case enters a given state. For example, you can have a [letter](#) created when a high-bill complaint case is **Rejected**. Similarly, you can have a [To Do entry](#) created when a high bill complaint enters the **Preliminary Investigation** state. This additional processing is held in algorithms that are plugged in on the case type and therefore you can define any type of additional processing.

You can also incorporate state transition logic within routines that are executed when a case enters a state, so that you do not need to rely upon CASETRAN to transition your cases. For example, when the state entry routines of the **Preliminary Investigation** status for a high-bill complaint are executed, they may be designed to transition the case into either the **Rejected** or **Field Investigation In Progress** state without waiting. Note that your Exit Validation and Exit Processing logic, if configured for the case state, will still be executed as part of the state transition. Auto-Transition logic for this state will be ignored during this transition.

Validation Before A Case Exits A State

You can define validation that executes before a case can exit a given state. For example, you might want to check the account's balance is less than a given value before a case can exit a given state. This validation logic is held in algorithms that are plugged in on the case type and therefore you can define any type of validation.

Additional Processing When Exiting A State

You can define additional processing that should happen when a case exits a given state. For example, you can have a [To Do entry](#) automatically completed when a high bill complaint leaves the **Decision Checkpoint** state. This additional processing is held in algorithms that are plugged in on the case type and therefore you can define any type of additional processing.

Automatic Transition Rules

You can define rules that automatically transition a case into a different state. For example, you can indicate a literature request should be transitioned to the **Follow Up** state 1 week after the literature is sent. Similarly, you can indicate a high-bill complaint should transition to the **Decision Checkpoint** state after the fieldwork is complete. These rules are held in algorithms that are plugged in on the case type and therefore you can define any type of automatic transition rules.

Cases in a state with automatic transition rules are monitored by the **CASETRAN** background process. Each time this program runs, the respective automatic transition plug-in is called for each such case and it transitions the case if the condition applies.

When to execute CASETRAN. Because your automatic transition rules will be dependent on your business requirements, you need to think carefully about when you run the **CASETRAN** background process. For example, if you have automatic transition rules that transition a case to a new state when a related field activity is completed, you would want to schedule this job to run after field activities are uploaded. If you have rules to transition a case after a customer pays a deposit, you'd want to schedule this job to run after payments are uploaded. Bottom line - your business rules will dictate the frequency of execution.

When the user adds a new case or changes the state of a case manually the system attempts to auto-transition the case to subsequent statuses as necessary. If auto-transition rules apply to the new state (and to subsequent ones) they would be executed right away. In other words, you don't need to wait for the auto-transition background process to be executed. An indication that the case was auto-transitioned online is displayed right below the action buttons section.

Auto-Transition Errors. Online auto-transition is performed recursively committing each successful state transition to the database. It is performed up to 100 times or until an error is encountered during the process. If this happens, auto-transition stops at the last **non-transitory** state into which a successful transition had occurred. Two case log entries will be generated automatically - one containing the message that a transition error has occurred, and a second containing the actual error message. A To Do entry will also be generated automatically upon rollback. The type of this To Do entry will be taken from 1) the **Case Transition Exception To Do Type** option for the **Business Object** associated with the case type, and if this is not populated, 2) the **Exception To Do Type** indicated on the Case Options Feature Configuration. All of the above error handling is true for both batch and online processing of cases.

Triggering Auto-Transition. If you have a customized process that affects the state of a case and you want the case to be auto-transitioned right away, i.e. not wait for the next scheduled **CASETRAN** background process to execute, you can customize that process to trigger auto-transition for the specific case, or you can put the state transition logic into the routines that execute at state entry time.

Script Launching Option

You can define whether the script associated with a given state is to be automatically launched while the case is in that state. The system supports the following options:

- Launch the script only if no script is currently active.

- Always launch the script unless this specific script is currently active.

Warning! With this option, if a script is currently open in the page's BPA script area then it will be automatically closed and the case script will open.

- Do not automatically launch the script.

You do this by plugging-in a **Script Launching** algorithm for the given state. If no such plug-in is provided the script is not automatically launched.

Which Users Can Transition A Case Into A State

If you have [restricted access](#) to a case type, you can further restrict which user groups are allowed to transition a case into specific states. For example, you can control which user group can transition a high bill complaint into the **Preliminary Investigation** state. The following points describe how this is done:

- Define actions on the [application service](#) defined on the case type. You must define an action for each status that you need to secure.
- Define each status's corresponding action. Note, you only need to link a status to an action if it's secured. Any user with [access](#) to the case type can perform statuses that aren't linked to actions.
- Define the transition role for each status's valid next status. You can assign valid next statuses to be reachable via system (only), or system and user.
- Define which [user groups](#) have access to the actions (i.e., statuses). In addition, these user groups should have access to the **Change** action.

Responsible User Applicability

Some of your cases may require a "responsible user". This is the user who has overall responsibility for the case. When you set up a case type, you define if a responsible user is required, optional or not allowed on its cases.

The following points describe how to set up the system if a responsible user is not required when a case is first created, but is later in its lifecycle:

- Indicate that a responsible user is optional on the case type
- Plug-in either an [exit validation](#) or [entry validation](#) algorithm on one of the case type's states to require a responsible user at some point in a case's lifecycle

Address To Do entries to the responsible user. If you use the [base-package algorithm](#) to create a To Do entry when a case enters a given state, you can indicate that the To Do entry should be addressed to the responsible user on the case.

Scripts and Cases

There are three ways [Business Process Assistant scripts](#) can be used to manage cases:

- You can create a BPA script to help users create a case. For example, a script can help a user create a new high-bill complaint.

Using a script to create a case can save a user a lot of time (and training efforts). This is because the script can automatically populate many fields on the case based on answers to questions.

Refer to [Initiating Scripts](#) for a description of how end-users initiate scripts.

- You can create a script to help users work on a case when it's in a given state. Refer to [A Script That Helps A User Work A Case](#) for more information.
- You can [set up your case types to create To Do entries](#) to notify users when cases exist that require their attention. Users can complete many of these ToDo entries without assistance. However, you can set up the system to automatically launch a script when a user selects a ToDo entry. For example, consider a ToDo entry that highlights a high-bill complaint that requires investigation. You can set up the system to execute a specific script when a user selects this ToDo entry. This script might guide the user through the investigation process (and help them update the case). Refer to [Executing A Script When A To Do Entry Is Selected](#) for more information.

To Do's and Cases

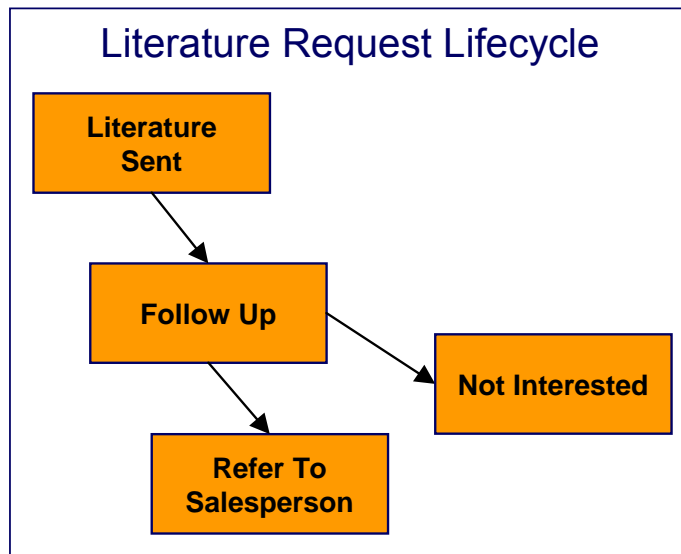
The topics in this section provide background information about how to facilitate case management with [To Do entries](#).

Contents

[Creating and Completing To Do Entries](#)
[Launching Scripts When To Do Entries Are Selected](#)
[All To Do Entries Are Visible](#)

Creating and Completing To Do Entries

You can configure your case types to create and complete [To Do entries](#) when a case enters or exits a state. Let's use the following [lifecycle diagram](#) to illustrate a potential use of To Do's:



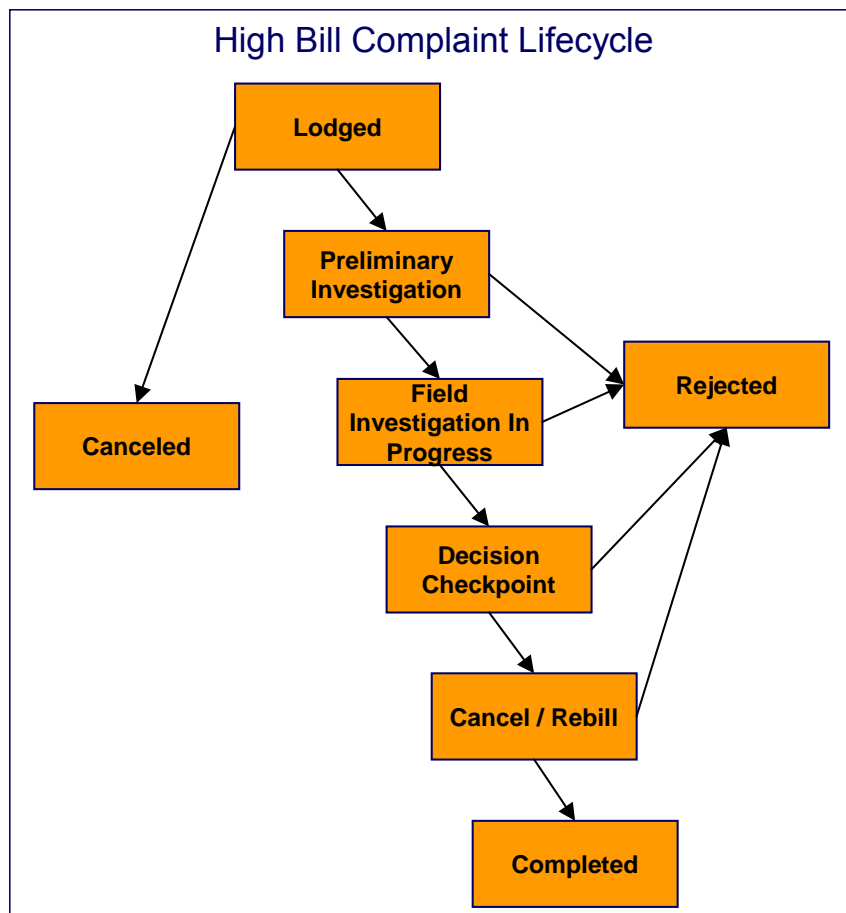
Let's assume the following:

- You want a To Do entry created when a literature request case enters the **Follow Up** state. You want this To Do automatically completed when the case enters either the **Refer To Salesperson** or **Not Interested** states. Note, we refer to this as the "first" To Do entry below.
- You want a different To Do entry created when a case enters the **Refer To Salesperson** state. You do not want the system to automatically complete this entry (the sales person must manually do this). Note, we refer to this as the "second" To Do entry below.

To implement the above, you would set up a case type as follows:

- Plug-in an [entry processing](#) algorithm on the **Follow Up** status to create the first To Do entry.
- Plug-in an [exit processing](#) algorithm on the **Follow Up** status to complete the first To Do entry.
- Plug-in an [entry processing](#) algorithm on the **Refer To Salesperson** status to create the second To Do entry.

While the case type illustrated above had a single To Do entry "active" at any point in time, you can easily configure a case type to have multiple To Do entries active at any point in time. Let's use the following lifecycle diagram to illustrate this point:



Let's assume the following:

- You want a To Do entry created when a high bill complaint is created and you want it completed when the case reaches the **Canceled**, **Rejected** or **Approved** states. This To Do entry could be used by a supervisor to monitor the number of high-bill complaints being worked. Note, we refer to this as the "first" To Do entry below.
- You want a different To Do entry created when the case enters the **Preliminary Investigation** state and you want this entry automatically completed when the case leaves this state. Note, we refer to this as the "second" To Do entry below.
- You want a different To Do entry created when the case enters the **Decision Checkpoint** state and you want this entry automatically completed when the case leaves this state. Note, we refer to this as the "third" To Do entry below.

To implement the above, you would set up the case type as follows:

- Plug-in an [entry processing](#) algorithm on the **Lodged** status to create the first To Do entry. Plug-in an [entry processing](#) algorithm on the **Canceled**, **Rejected** and **Completed** statuses to complete this entry.
- Plug-in an [entry processing](#) algorithm on the **Preliminary Investigation** status to create the second To Do entry. Plug-in an [exit processing](#) algorithm on the **Preliminary Investigation** status to complete this entry. We elected to use an exit processing algorithm because we only have to plug it in on one status. If we'd used an entry processing algorithm, we would need to plug it in on the 2 statuses into which a **Preliminary Investigation** status can transition.
- Plug-in an [entry processing](#) algorithm on the **Decision Checkpoint** status to create the third To Do entry. Plug-in an [exit processing](#) algorithm on the **Decision Checkpoint** status to complete this entry.

Launching Scripts When To Do Entries Are Selected

You can set up your case types to create To Do entries to notify users when cases exist that require their attention. Users can complete many of these To Do entries without assistance. However, you can set up the system to automatically launch a script when a user selects a To Do entry. For example, consider a To Do entry that highlights a high-bill complaint that requires investigation. You can set up the system to execute a specific script when a user selects this type of To Do entry. This script might guide the user through the investigation process. Refer to [Executing A Script When A To Do Entry Is Selected](#) for more information.

All To Do Entries Are Visible

When a case is displayed on [Case Maintenance](#), the system summarizes the number of To Do entries associated with the case (if you've [set up your To Do types](#) appropriately).

Examples of Case Types

The topics that follow provide examples of case types related to several business processes. Use the information in this section to form an intuitive understanding of case types. After attaining this understanding, you'll be ready to design your own case types.

High Bill Complaint

Some organizations will set up a case to manage a high-bill complaint. The following diagram illustrates how such a case type might look:

Case Type – High Bill Complaint

Person / Account / Premise Applicability

Field	Applicability
Person	Required
Account	Required
Premise	Optional

Secured

Yes, all actions

Fields To Be Captured

Field	Required / Optional	Default Value
Bill ID	Optional	“ ”
Cancel reason	Optional	“ ”
Reject reason	Optional	“ ”

Applicability Rules

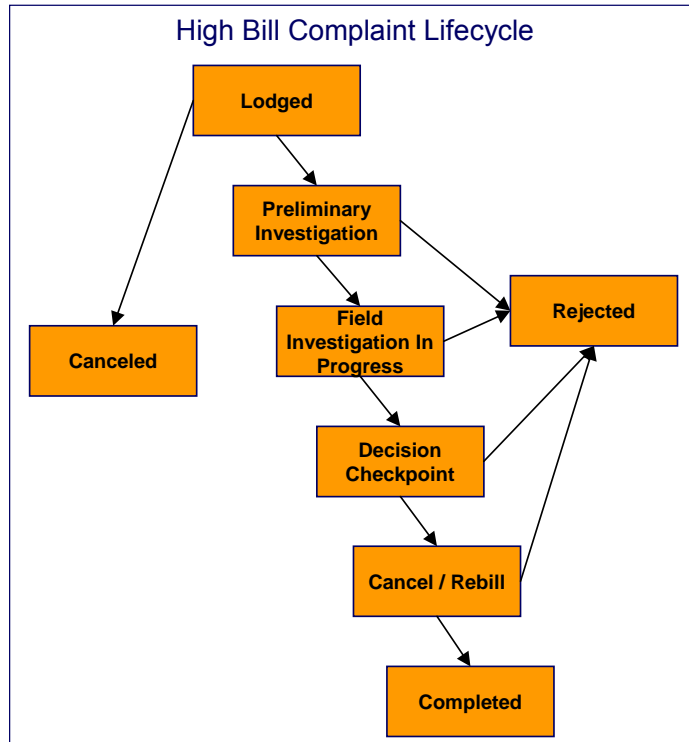
Field	Applicability
Responsible User	Optional
Contact Information	Optional

Note the following about the High Bill Complaint case type:

- Notice that we set up the case type to require a person and an account, but premise is optional. This is because a bill can span multiple premises and knowing the premise isn't so important on cases of this type.
- We need to restrict high-bill complaints to specific user groups. This means we need to [set up a specific application service](#) for this case type (that has a separate "action" for each status).
- Cases of this type have 3 additional fields over their lifetime. Notice the following:
 - The **Bill ID** characteristic is set up to be optional. This is because we've assumed that sometimes a high-bill complaint case can be lodged when you can't find the bill in question and you still want to log the case.
 - Later in this section, you'll see that we've configured the **Preliminary Investigation** status to require a **Bill ID** before a case can enter this state.
 - Both the **High Bill Complaint Cancel Reason** and **Reject Reason** are optional. Later in this section, you'll see that we've configured the Canceled and Rejected statuses to require these fields, respectively.

- Cases of this type do not need a **Responsible User** when first created. Later in this section, you'll see how we've configured the **Preliminary Investigation** status to require a **Responsible User** before a case can enter this state.
- Cases of this type do not need **Contact Information**. This was a subjective decision and depends on your organization's philosophy.

The topics that follow describe each of the statuses in a high-bill complaint's [lifecycle](#). We have assumed the following state transition rules:

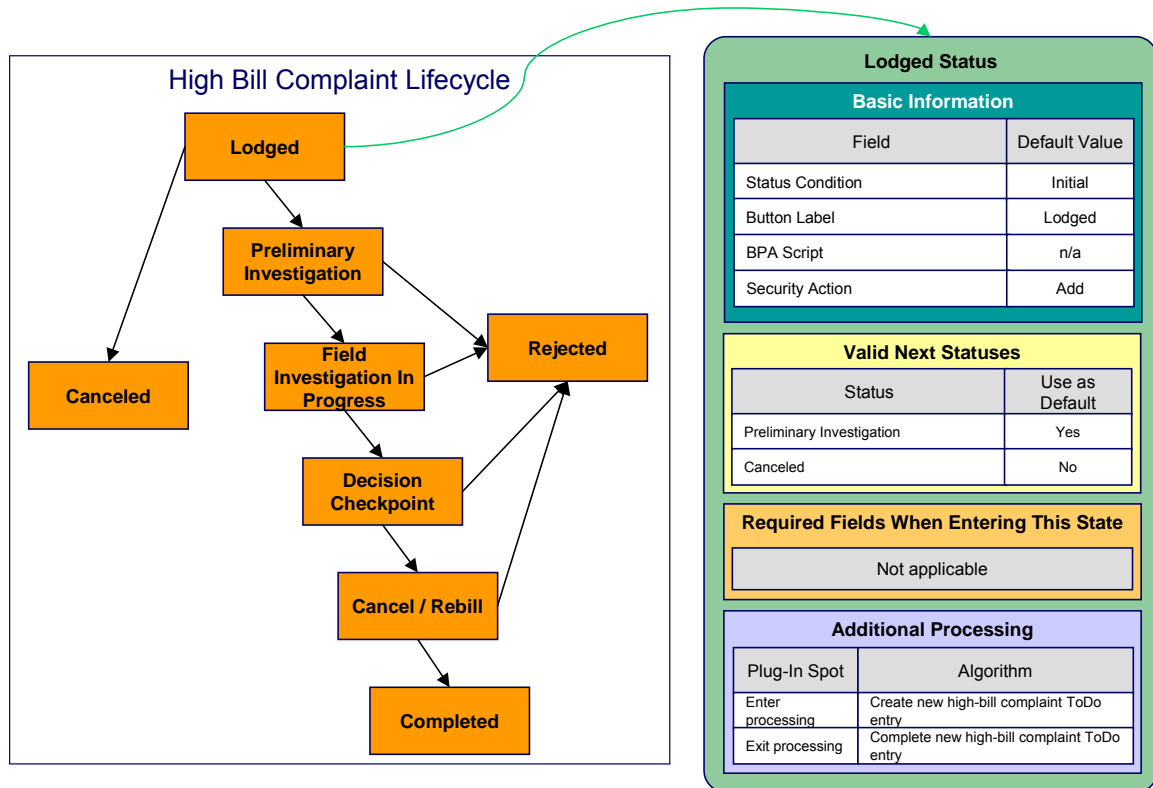


Contents

[Lodged High Bill Complaint](#)
[Preliminary Investigation High Bill Complaint](#)
[Field Investigation High Bill Complaint](#)
[Decision Checkpoint High Bill Complaint](#)
[Cancel Rebill High Bill Complaint](#)
[Completed High Bill Complaint](#)
[Rejected High Bill Complaint](#)
[Canceled High Bill Complaint](#)

Lodged High Bill Complaint

The following is an example of the configuration of the **Lodged** status for high bill complaint cases.

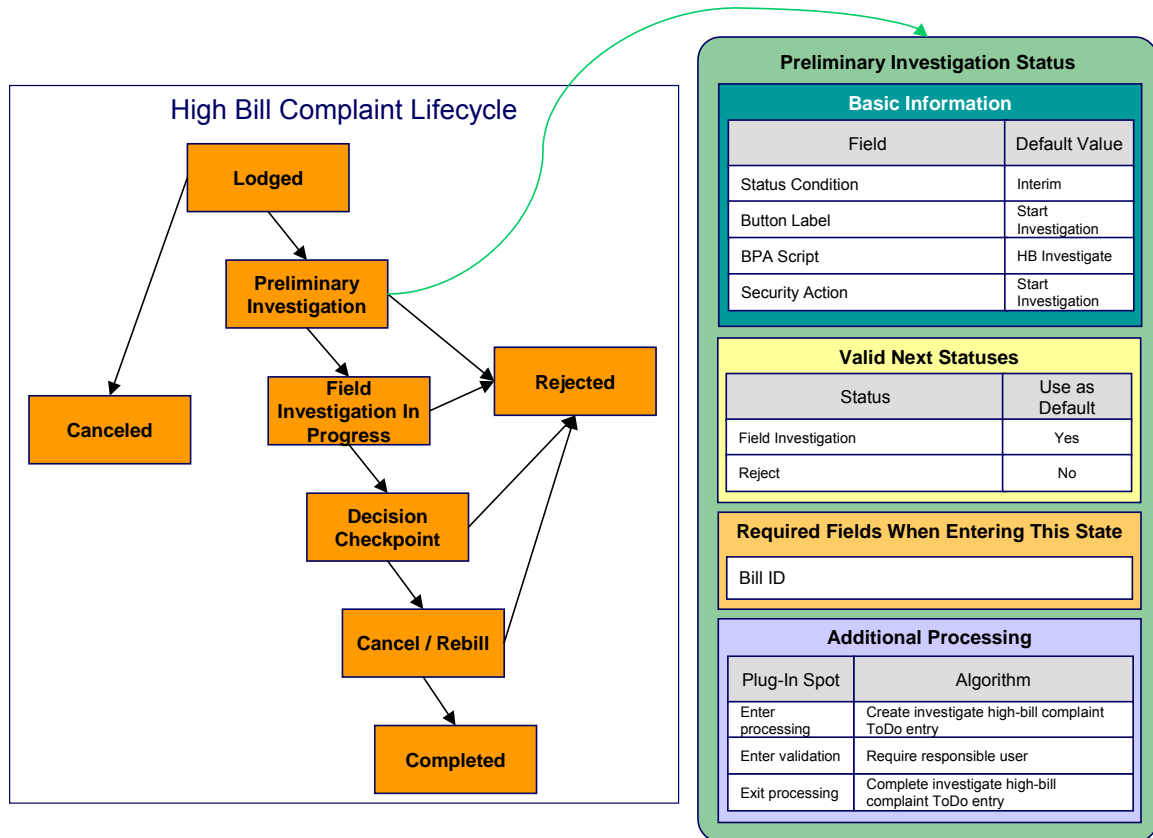


Note the following about this status:

- **Lodged** is the initial state. The initial state is the state populated on new cases of a given type. Remember that only one status can be defined as the initial state.
- It has a button label even though it's the initial state. The above diagram doesn't allow a user to ever transition a case into this state and therefore there will never be a button with such a label. However, it's a required field just in case you change your business rules.
- We have decided not to use a BPA script to help a user work on a high-bill complaint when it's in this state (this is probably not the best decision as BPA scripts can be quite useful).
- We have associated the Add action with this status. This means that only users with rights to the add action for the application service defined on the case type can add cases of this type.
- Notice that Valid Next Statuses are what restricts a case in this state to be transitioned to the **Canceled** and **Preliminary Investigation** states.
- Notice that the Additional Processing plug-ins create and complete a To Do entry when a case enters and exits this state, respectively.

Preliminary Investigation High Bill Complaint

The following is an example of the configuration of the **Preliminary Investigation** status for high bill complaint cases.

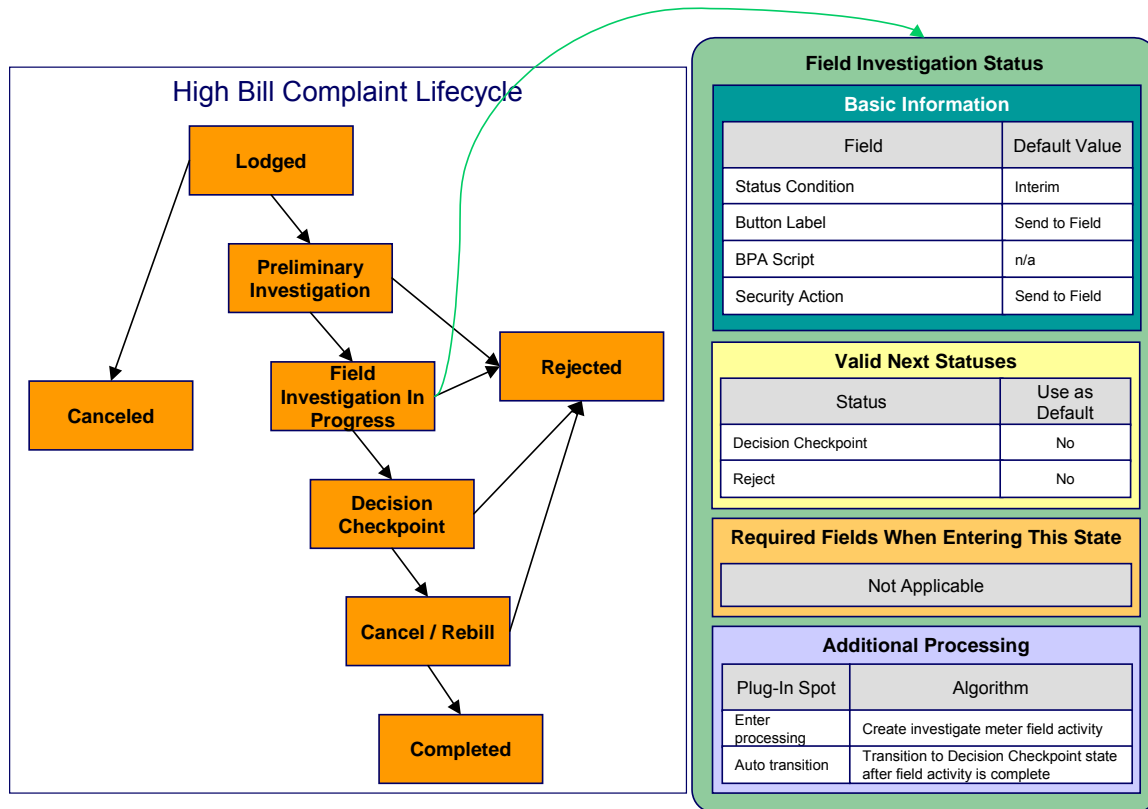


Note the following about this status:

- **Preliminary Investigation** is an interim state (meaning that it's not an initial or final state).
- It has a button label of **Start Investigation**. This is the label on the button that a user presses to move a case into this state. This button will only appear on cases that are in the **Lodged** state as this is the only state that can transition into the **Preliminary Investigation** state.
- We have decided not to specify a BPA script on this status. Rather, we're going to set up the To Do type used to highlight cases in this state to automatically launch an appropriate BPA script when a user selects the To Do entry.
- We have associated the **Start Investigation** action with this status. This means that only users with rights to this action for the application service defined on the case type can move a case into this state.
- Notice that Valid Next Statuses are what restricts a case in this state to be transitioned to the **Field Investigation** and **Rejected** states.
- Notice that a **Bill ID** must be specified on the case before it can be moved into this state.
- Notice that the Additional Processing plug-ins do the following:
 - Create and complete a To Do entry when a case enters and exits this state, respectively
 - Require a responsible user before a case can enter this state

Field Investigation High Bill Complaint

The following is an example of the configuration of the **Field Investigation In Progress** status for high bill complaint cases.



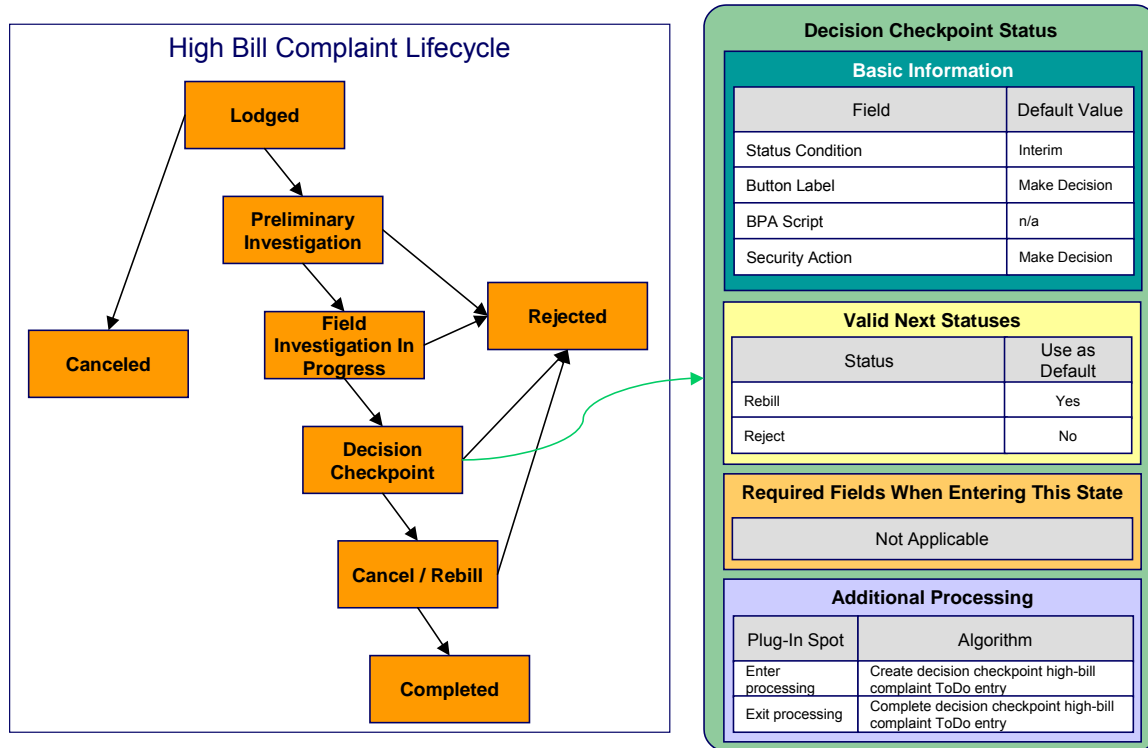
Note the following about this status:

- **Field Investigation In Progress** is an interim state (meaning that it's not an initial or final state).
- It has a button label of **Send to Field**. This is the label on the button that a user presses to move a case into this state. This button will only appear on cases that are in the **Preliminary Investigation** state as this is the only state that can transition into the **Field Investigation In Progress** state.
- We have decided not to specify a BPA script on this status because users don't work cases in this state (see the Additional Processing notes below for why this is the case).
- We have associated the **Send to Field** action with this status. This means that only users with rights to this action for the application service defined on the case type can move a case into this state.
- Notice that Valid Next Statuses are what restricts a case in this state to be transitioned to the **Decision Checkpoint** and **Rejected** states.
- Notice that the Additional Processing plug-ins do the following:
 - Create a field activity when a case enters this state

- Cause the case to automatically transition to the **Decision Checkpoint** state when the field activity is completed

Decision Checkpoint High Bill Complaint

The following is an example of the configuration of the **Decision Checkpoint** status for high bill complaint cases.



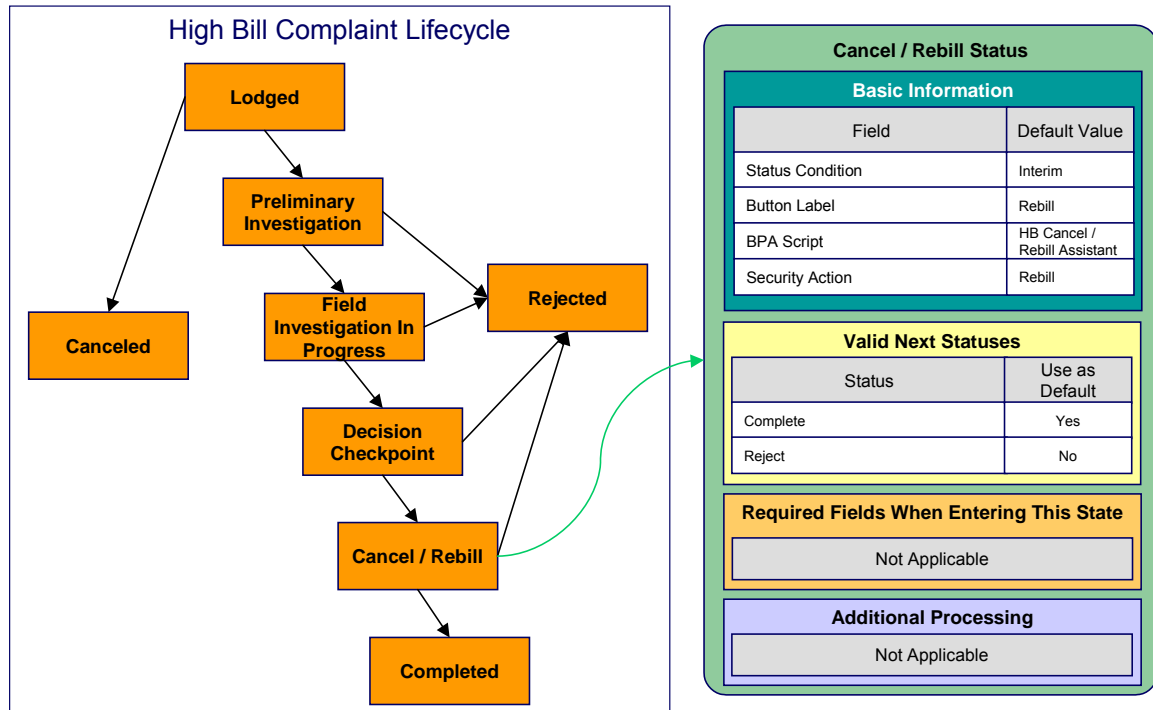
Note the following about this status:

- **Decision Checkpoint** is an interim state (meaning that it's not an initial or final state).
- It has a button label of **Make Decision**. This is the label on the button that a user presses to move a case into this state. This button will only appear on cases that are in the **Field Investigation in Progress** state as this is the only state that can transition into the **Decision Checkpoint** state.
- We have decided not to specify a BPA script on this status. Rather, we're going to set up the To Do type used to highlight cases in this state to automatically launch an appropriate BPA script when a user selects the To Do entry.
- We have associated the **Make Decision** action with this status. This means that only users with rights to this action for the application service defined on the case type can move a case into this state. Because the system will automatically transition cases into this state when the related field activity is complete, users will probably never press this button (and you may wish to prevent users from pressing this button by restricting security rights to the related action).
- Notice that Valid Next Statuses are what restricts a case in this state to be transitioned to the **Cancel / Rebill** and **Rejected** states.

- Notice that the Additional Processing plug-ins create and complete a To Do entry when a case enters and exits this state, respectively.

Cancel Rebill High Bill Complaint

The following is an example of the configuration of the **Cancel / Rebill** status for high bill complaint cases.

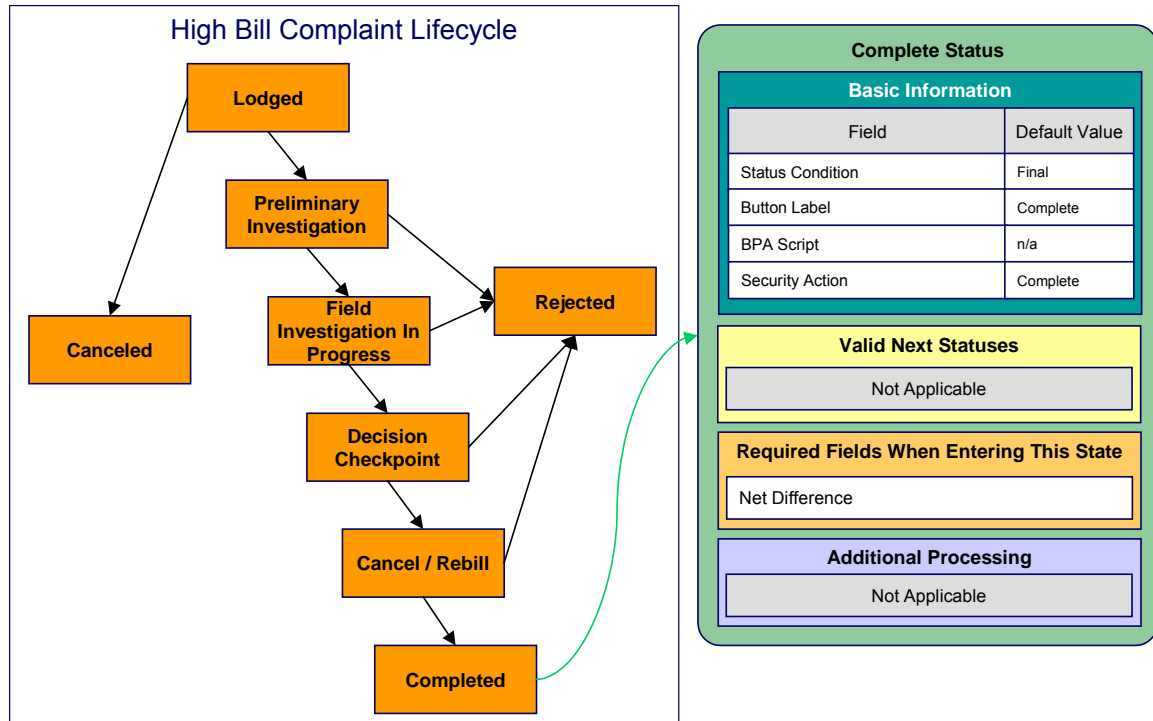


Note the following about this status:

- **Cancel / Rebill** is an interim state (meaning that it's not an initial or final state).
- It has a button label of **Cancel / Rebill**. This is the label on the button that a user presses to move a case into this state. This button will only appear on cases that are in the **Decision Checkpoint** state as this is the only state that can transition into the **Cancel / Rebill** state.
- We have referenced a BPA script that can assist a user in the cancel / rebill efforts.
- We have associated the **Rebill** action with this status. This means that only users with rights to this action for the application service defined on the case type can move a case into this state.
- Notice that Valid Next Statuses are what restricts a case in this state to be transitioned to the **Completed** and **Rejected** states.

Completed High Bill Complaint

The following is an example of the configuration of the **Completed** status for high bill complaint cases.

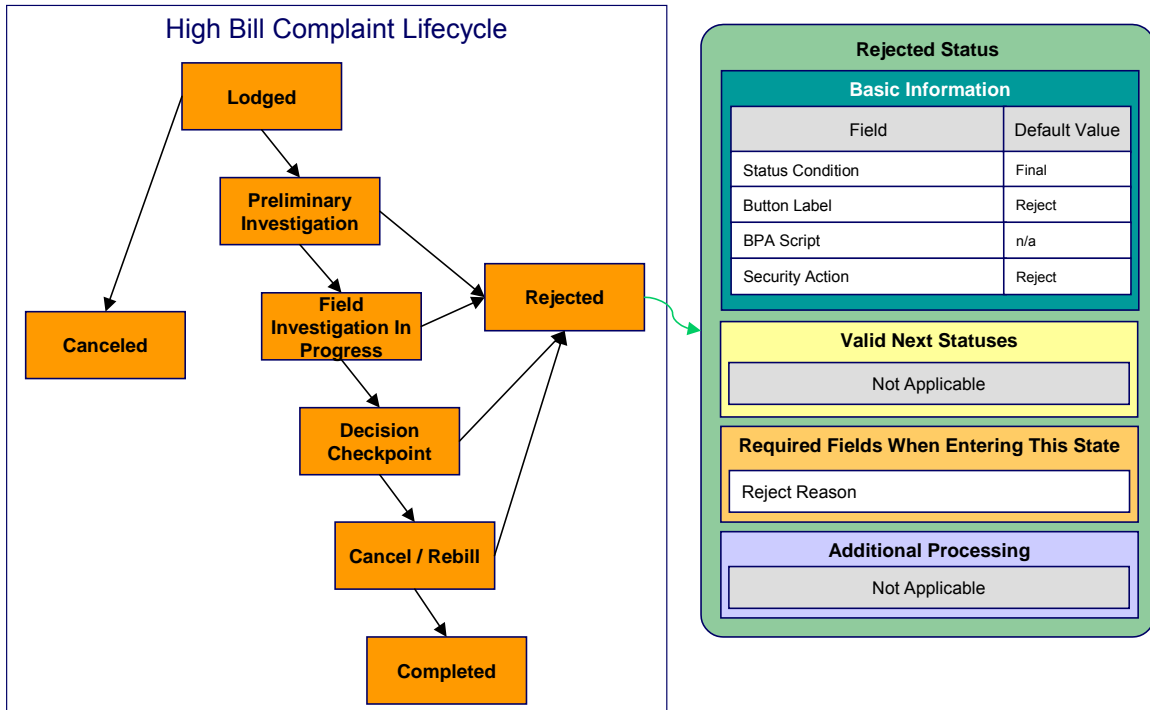


Note the following about this status:

- **Completed** is a final state (meaning that no further action is necessary).
- It has a button label of **Complete**. This is the label on the button that a user presses to move a case into this state. This button will only appear on cases that are in the **Cancel / Rebill** state as this is the only state that can transition into the **Completed** state.
- We have not referenced a BPA script because this is a final state (and no additional user action is necessary).
- We have associated the **Complete** action with this status. This means that only users with rights to this action for the application service defined on the case type can move a case into this state.
- Notice that there are no Valid Next Statuses (because this is a final state). If you wanted to allow a Completed case to be reopened, you would need to define the state into which a Completed case could be transitioned.
- Notice that the **Net Difference** must be specified on the case before it can be moved into this state. This would be the difference to the customer's balance after the cancel rebill took place.

Rejected High Bill Complaint

The following is an example of the configuration of the **Rejected** status for high bill complaint cases.

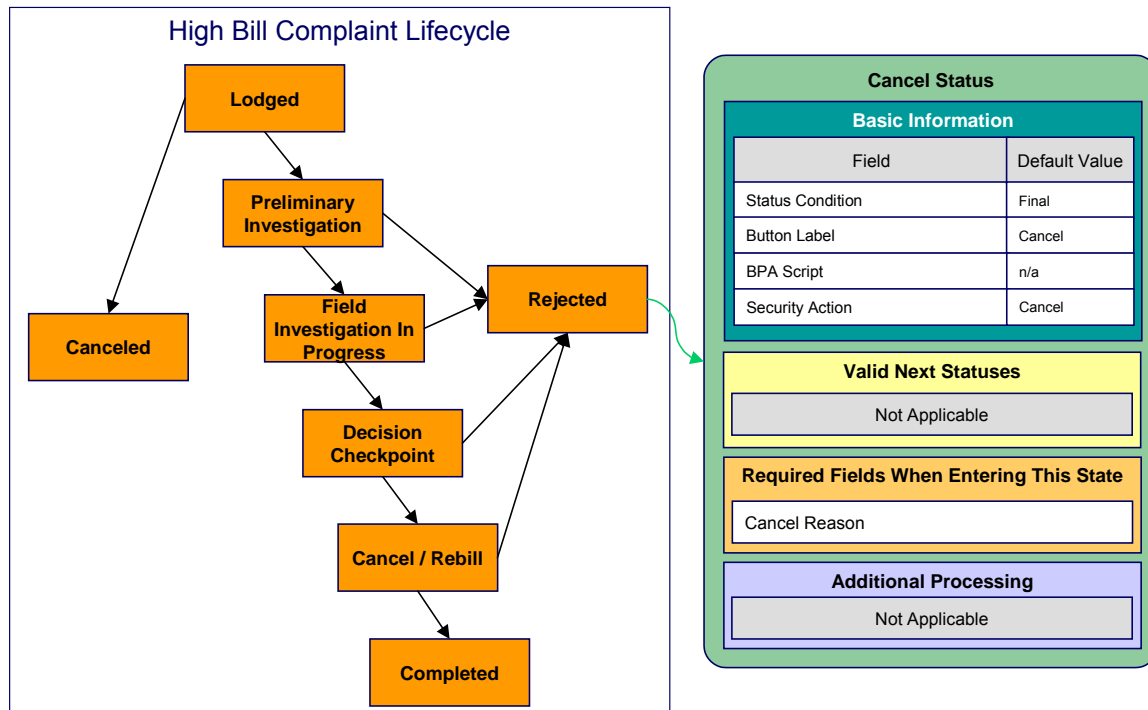


Note the following about this status:

- **Rejected** is a final state (meaning that no further action is necessary).
- It has a button label of **Reject**. This is the label on the button that a user presses to move a case into this state. This button will appear on cases that are in the **Preliminary Investigation**, **Field Investigation in Progress**, **Decision Checkpoint** and **Cancel / Rebill** states.
- We have not referenced a BPA script because this is a final state (and no additional user action is necessary).
- We have associated the **Reject** action with this status. This means that only users with rights to this action for the application service defined on the case type can move a case into this state.
- Notice that there are no Valid Next Statuses (because this is a final state). If you wanted to allow a **Rejected** case to be reopened, you would need to define the state into which a **Rejected** case could be transitioned.
- Notice that a **Reject Reason** must be specified on the case before it can be moved into this state.

Canceled High Bill Complaint

The following is an example of the configuration of the **Canceled** status for high bill complaint cases.



Note the following about this status:

- **Canceled** is a final state (meaning that no further action is necessary).
- It has a button label of **Cancel**. This is the label on the button that a user presses to move a case into this state. This button will only appear on cases that are in the **Lodged** state as this is the only state that can transition into the **Completed** state.
- We have not referenced a BPA script because this is a final state (and no additional user action is necessary).
- We have associated the **Cancel** action with this status. This means that only users with rights to this action for the application service defined on the case type can move a case into this state.
- Notice that there are no Valid Next Statuses (because this is a final state). If you wanted to allow a **Canceled** case to be reopened, you would need to define the state into which a **Canceled** case could be transitioned.
- Notice that a **Cancel Reason** must be specified on the case before it can be moved into this state.

Setting Up Case Management Options

The topics in this section describe how to set up the system to enable case management.

Warning! The following topics assume you thoroughly understand the concepts described under [The Big Picture Of Cases](#).

Contents

[Installation Options](#)
[Setting Up Application Services](#)
[Setting Up Scripts](#)
[Setting Up To Do Types](#)
[Setting Up Characteristic Types](#)
[Setting Up Case Types](#)

Installation Options

Contents

[Case Info May Be Formatted By An Algorithm](#)
[Alert Info Is Controlled By An Installation Algorithm](#)

Case Info May Be Formatted By An Algorithm

An algorithm may be plugged in on the [installation record](#) to format the standard case information that appears throughout the system. Refer to [CSIN-DFLT](#) for an example of this algorithm.

This algorithm may be further overridden by a "Case information" plug-in on the [Case Type](#). Refer to [C1-CT-INFO](#) for an example of this algorithm.

Alert Info Is Controlled By An Installation Algorithm

An algorithm that is plugged in on the [installation record](#) is responsible for formatting the alerts that highlight if the person / account / premise in context has open cases. Refer to [CCAL-CASE](#) for an example of this algorithm.

Setting Up Application Services

As described under [Access Rights](#), you can prevent unauthorized users from accessing cases. The following points describe how to implement this type of security:

- Create an [application service](#) for each case type that needs to be secured
- Create an action on the application service for each status you need to secure
- Link the valid [user groups](#) to the application service and define which actions they can perform
- Define the application service on the [case type](#)
- Define the related action for each status on the [case type / status](#)

Setting Up Scripts

As described under [Scripts and Cases](#), BPA scripts can facilitate the creation and working of cases. Refer to the [Defining Script Options](#) for instructions describing how to set up scripts.

Setting Up To Do Types

As described under [To Do's and Cases](#), To Do entries can be used to highlight cases that require user attention.

The following points provide a high-level description of how to create (and complete) To Do entries for a case type:

- Create a To Do type for each different type of To Do entry used during a case's lifecycle
 - On the To Do type, think carefully about the roles whose users can work on the entries
 - Also consider if you would like a BPA script launched when a user selects the entry
- Specify the To Do type on the appropriate [entry processing](#) or [exit processing](#) algorithm
- If you want the system to automatically complete To Do entries, specify the To Do type on the appropriate [entry processing](#) or [exit processing](#) algorithm

Please be aware that the case maintenance transaction highlights the number of open and being worked To Do entries linked to the case being displayed on the page. However, the system can only do this if the To Do entries reference a [foreign-key characteristic](#) whose foreign key references the case table. If you use the [CSEN-TD](#) algorithm to create To Do entries when a case enters a given state, this algorithm will do this for you if:

- You have set up a [foreign-key characteristic type](#) whose [foreign key](#) references the case table
- In addition, the characteristic type must reference a characteristic entity of **To Do Entry**

Setting Up Characteristic Types

As described under [Additional Information](#), some of your cases may require additional information (in the form of [characteristics](#)). If this is true, you must set up the characteristic types before setting up the case types.

Refer to [Setting Up To Do Types](#) for instructions regarding a characteristic type that must be set up in order for the system to know the To Do entries that are associated with a case.

If you use the [CSEN-CC](#) algorithm to create customer contacts when a case enters a given state, you should set up a [foreign-key characteristic type](#) as follows:

- Its [foreign key](#) must reference the case table
- In addition, the characteristic type must reference a characteristic entity of **Customer Contact**

Setting Up Case Types

The case type maintenance transaction is used to maintain your case types. The topics in this section describe how to use this transaction.

Refer to [The Big Picture Of Cases](#) for more information about how a case type encapsulates the business rules that govern a case.

Contents

- [Case Type - Main](#)
- [Case Type - Case Characteristics](#)
- [Case Type - Lifecycle](#)

Case Type - Main

Use this page to define basic information about a case type.

Open the case type page by selecting **Case Type** from the admin menu.

Recommendation. Before using this transaction, we strongly recommend that you review the [Examples of Case Types](#).

Main Information

Enter a unique **Case Type** code and **Description** for the case type.

Use **Long Description** to provide a more detailed explanation of the purpose of the case type.

Person Usage controls the applicability of a person on cases of this type. Select **Required** if a person must be defined on this type of case. Select **Optional** if a person can optionally be defined on this type of case. Select **Not Allowed** if a person is not allowed on this type of case.

Account Usage controls the applicability of an account on cases of this type. Select **Required** if an account must be defined on this type of case. Select **Optional** if an account can optionally be defined on this type of case. Select **Not Allowed** if an account is not allowed on this type of case.

Premise Usage controls the applicability of a premise on cases of this type. Select **Required** if a premise must be defined on this type of case. Select **Optional** if a premise can optionally be defined on this type of case. Select **Not Allowed** if a premise is not allowed on this type of case.

If you need to restrict access to cases of this type to specific user groups, reference the appropriate **Application Service**. Refer to [Setting Up Application Services](#) for the details of how to secure access to your cases.

If you are configuring a case type to handle the processing of data defined via a **Business Object**, associating the case type with a business object serves to link the properties of the business object (e.g. BO options) with cases of that type. Refer to [Business Objects](#) for further information. In addition, refer to [Automatic Transition Rules](#) for information on the role of BO options in case auto-transition errors.

Responsible User Usage controls the applicability of a responsible user on cases of this type. Select **Required** if a responsible user must be defined on this type of case. Select **Optional** if a responsible user can optionally be defined on this type of case. Select **Not Allowed** if a responsible user is not allowed on this type of case. Refer to [Responsible User Applicability](#) for more information.

Contact Information Fields

There are three contact information fields: **Contact Person & Method Usage**, **Contact Instructions Usage**, and **Callback Phone Usage**. These fields are used to determine whether or not each type of contact information must be entered on case records with this case type. Select **Required** if the contact information must be entered, select **Optional** if the user can choose whether or not to include the contact information on this type of case, or select **Not Allowed** if the contact information cannot be entered on this type of case.

Algorithms

The **Algorithms** grid contains algorithms that control functions for cases of this type. You must define the following for each algorithm:

- Specify the **System Event** with which the algorithm is associated (see the table that follows for a description of all possible events).
- Specify the **Sequence Number** and **Algorithm** for each system event. You can set the **Sequence Number** to 10 unless you have a **System Event** that has multiple **Algorithms**. In this case, you need to tell the system the **Sequence** in which they should execute.

The following table describes each **System Event**.

System Event	Optional / Required	Description
<i>Case Information</i>	Optional	<p>We use the term "Case information" to describe the basic information that appears throughout the system to describe a case. The data that appears in "case information" is constructed using this algorithm.</p> <p>Plug an algorithm into this spot to override the "Case information" algorithm on installation options or the system default "Case information" if no such algorithm is defined on installation options.</p> <p>Click here to see the algorithm types available for this system event.</p>

Case Type Tree

The tree summarizes the case type's lifecycle. You can use the hyperlinks to transfer you to the **Lifecycle** tab with the corresponding status displayed.

Case Type - Case Characteristics

To define characteristics that can be defined for cases of this type, open **Admin Menu, Case Type** and navigate to the **Case Characteristics** tab.

Description of Page

Use **Sequence** to control the order in which characteristics are defaulted. Turn on the **Required** switch if the **Characteristic Type** must be defined on cases of this type. Turn on the **Default** switch to default the **Characteristic Type** when cases of this type are created. Enter a **Characteristic Value** to use as the default for a given **Characteristic Type** when the **Default** box is checked. Refer to [Required Fields Before A Case Enters A State](#) for a description of how you can make option characteristics required at later stages in a case's lifecycle.

Case Type - Lifecycle

Case types that involve multiple users and multiple potential outcomes have complex lifecycles. Before you can design a case type's lifecycle, it's important that you thoroughly understand the concepts described under [Lifecycle](#) and [Status-Specific Business Rules](#). After thoroughly understanding these concepts, we recommend you perform the following design steps:

- Draw a "state transition diagram" as illustrated above under [Lifecycle](#). Keep in mind that if your state transition diagram is complex, your cases will be complex. While some cases warrant complexity, you should always ask yourself if there aren't better ways to achieve the desired results if your first effort results in complexity.
- Determine which characteristics (if any) are required during each stage of a case's lifecycle

- Determine when To Do entries (if any) should be created (and completed) during a case's lifecycle
- Determine additional validation (if any) that should be executed before a case enters and exits each state
- Determine additional processing (if any) that should transpire when a case enters or exits each state
- Determine if scripts are warranted to help users work the cases and, if so, design the scripts for each applicable state

When the above tasks are complete, you will be ready to set up a case type's lifecycle.

Open the Lifecycle page by selecting **Case Type** from the admin menu then the **Lifecycle** tab.

Note. You can navigate to a status by clicking on the respective node in the tree on the Main tab. You can also use the hyperlinks in the Next Statuses grid to display a specific status in the accordion.

Main Information

The **Status** accordion contains an entry for every status in the case type's [lifecycle](#).

Use **Status** to define the unique identifier of the status. This is NOT the status's description, it is simply the unique identifier used by the system.

Use **Description** to define the label that appears on the lifecycle accordion as well as the status displayed on the case.

Use **Script** to reference a BPA script that can assist a user work on a case while it's in this status. Refer to [A Script That Helps A User Work Through A Case](#) for the details.

Use **Access Mode** to define the action associated with this status. This field is disabled if an application service is not specified on the Main page. Refer to [Access Rights](#) for the details of how to use this field to restrict which users can transition a case into this state.

Use **Batch** to specify a batch control that will auto-transition the case. Any case in a status configured with a batch control will be transitioned when the batch job runs (rather than when [CASETRAN](#) is executed). For this purpose, batch process [C1-CSTRS \(Case Scheduled Transition\)](#) is supplied with base package, which will execute all Exit Status logic for the current status, and Enter Status logic for the destination status. You may choose to create a batch process with your own transition logic.

Note. If you wish to defer transitioning a case in a particular status until the batch process on your case type status is executed, you should not populate an Auto-Transition algorithm on that status. Otherwise, CASETRAN will transition the case according to your Auto-Transition logic.

Use **Comment** to describe the status. This is for your internal documentation requirements.

Use **Sequence** to define the relative order of this status in the tree on the Main page.

Use **Status Condition** to define if this status is an *Initial*, *Interim* or *Final* state. Refer to [One Initial State and Multiple Final States](#) for more information about how this field is used.

Use **Transitory State** to indicate whether a case should ever exist in this state. Only **Initial** or **Interim** states can have a transitory state value of **No**.

The **Alert Flag** is used to indicate whether or not an alert should be displayed for customers with cases in the state. (The alert is shown via the base package Installation – Alert algorithm.)

Algorithms

The **Algorithms** grid contains algorithms that control important functions for cases of this type. You must define the following for each algorithm:

- Specify the **System Event** with which the algorithm is associated (see the table that follows for a description of all possible events).
- Specify the **Sequence Number** and **Algorithm** for each system event. You can set the **Sequence Number** to 10 unless you have a **System Event** that has multiple **Algorithms**. In this case, you need to tell the system the **Sequence** in which they should execute.

The following table describes each **System Event** (note, all system event's are optional and you can define an unlimited number of algorithms for each event).

System Event	Description
<i>Auto Transition</i>	This algorithm is executed to determine if a case that's in this state should be transitioned into another state. Refer to Automatic Transition Rules for the details. Click here to see the algorithm types available for this system event.
<i>Enter Processing</i>	This algorithm holds additional processing that is executed when a case is transitioned into this state. You can also specify state transition logic within Enter Processing routines. Refer to Additional Processing When Entering A State for the details. Click here to see the algorithm types available for this system event.
<i>Enter Validation</i>	This algorithm holds validation logic that executes before a case can enter a given state. Refer to Validation Before A Case Enters A State for the details. Click here to see the algorithm types available for this system event.
<i>Exit Processing</i>	This algorithm holds additional processing that is executed when a case is transitioned out of this state. Refer to Additional Processing When Exiting A State for the details. Click here to see the algorithm types available for this system event.
<i>Exit Validation</i>	This algorithm holds validation logic that executes before a case can be transitioned out of a given state. Refer to Validation Before A Case Exits A State for the details. Click here to see the algorithm types available for this system event.
<i>Script Launching</i>	This algorithm sets the script launching option for the script associated with a given state, if any. Refer to Script Launching Option for the details. Click here to see the algorithm types available for this system event.

Next Statuses

Use the **Next Statuses** grid to define the statuses a user can transition a case into while it's in this state. Refer to [Valid States versus State Transition Rules](#) for more information. Please note the following about this grid:

- Use **Action Label** to indicate the verbiage to display on the action button used to transition to this status.

- **Sequence** controls the order of the buttons that appear on [Case - Main](#). Refer to [Buttons Are Used To Transition A Case Into A State](#) for more information.
- **Use as Default** controls which button (if any) is the default button.
- **Transition Condition** may be configured to identify a common transition path for cases of this type in the current state. This transition condition may then be referenced across multiple case types. You'll need to add values to Look Up table field **TR_COND_FLG** that fit the typical transitions for your case types (e.g. **Ok**, **Error**, etc.).

By assigning the transition condition value to a given "next status", you can design your Enter State transition or Auto-Transition logic to utilize those flag values *without specifying a status particular to a given case type*. Thus, similar logic may be used across a range of case types to transition a case into, for example, the next **Ok** state for the case's current status.

- **Transition Role** controls whether only the **System** or both **System and User** have the ability to transition a case into a given "next status".
- You can use the status description hyperlink to open the Status accordion to the respective status.
- When you initially set up a case type, none of the statuses will reside on the database and therefore you can't use the search to define a "next status". We recommend working as follows to facilitate the definition of this information:
 - Leave the Next Statuses grid blank when you initially define a case type's statuses
 - After all statuses have been saved on the database, update each status to define its Next Statuses (this way, you can use the search to select the status).

Required Characteristics

Use the **Required Characteristics** grid to define characteristics that are required when a case enters this state. Only **Optional** characteristics defined on the main page appear in this grid. Refer to [Required Fields Before A Case Enters A State](#) for more information.

Workflow and Notification Options

We use the term “notification” to reference the electronic transactions that you exchange with third parties when:

- They need information about a customer
- They need to change something about a customer

For example, an energy service provider sends a notification to an electric distribution company when a customer elects to use them as their energy provider.

When a notification is received, the system responds by creating a workflow process. The workflow process contains workflow events. These events perform the processing necessary to execute the notification.

Workflow processing is difficult to explain because its flexible design can be used to automate many different types of multi-event processes. For example,

- You can use workflow processing to manage the events associated with the inspection of a new premise.
- You can use workflow processing to manage the events that transpire in a deregulated market when a customer wants to switch energy service providers.

Warning! Setting up the workflow process control tables is as challenging as your organization’s business rules. If you don’t have automated workflow processes, you don’t have to setup anything. If you have sophisticated workflow processing requirements, your setup process will be more challenging.

The topics in this section describe tables that control your automated workflow and notification processing.

Contents

[The Big Picture Of Workflow Processing](#)
[The Big Picture Of Workflow Events](#)
[The Big Picture Of Notification Processing](#)
[Creating Notification and Workflow Procedures](#)

The Big Picture Of Workflow Processing

Refer to [The Lifecycle Of A Workflow Process And Its Events](#) for more information about workflow processes.

The Big Picture Of Workflow Events

This section describes the various types of workflow events and their lifecycle:

Contents

[How Are Workflow Events Created?](#)

[Executing Workflow Events On Their Trigger Date](#)
[Workflow Event Lifecycle](#)
[Workflow Event Dependencies & Trigger Date](#)
[Workflow Events May Be In Error](#)
[Some Workflow Events May Fail](#)
[Errors versus Failure](#)
[Waiting Events And Their Waiting Process](#)
[How Are Workflow Events Canceled?](#)
[Workflow Processes Can Have Multiple Branches](#)

How Are Workflow Events Created?

Workflow events may be created as follows:

- The process that uploads notification requests creates a workflow process to implement the notification request. The workflow process has one or more workflow event(s). The number and type of events is controlled by the workflow process template associated with the workflow process. Refer to [What Type Of Workflow Process Is Created?](#) for more information about how notification requests cause workflow processes to be created.
- Workflow events will be created when an operator creates an ad hoc workflow process. The number and type of workflow events are controlled by the workflow process template associated with the workflow process.
- An ad hoc workflow event may be created and linked to an existing workflow process by an operator at their discretion.
- The system may be customized to create a workflow process when something noteworthy happens in the system. Refer to [System Conditions May Trigger Notification and Workflow](#) for more information.

Bottom line. Most workflow events are created by the system when it creates a workflow process to implement an incoming notification. If you need to create an ad hoc workflow event, you can either create a workflow process using a template that contains the desired event OR link the desired event to an existing workflow process.

For more information about creating ad hoc workflow processes and events, refer to [Workflow Process Maintenance](#).

Executing Workflow Events On Their Trigger Date

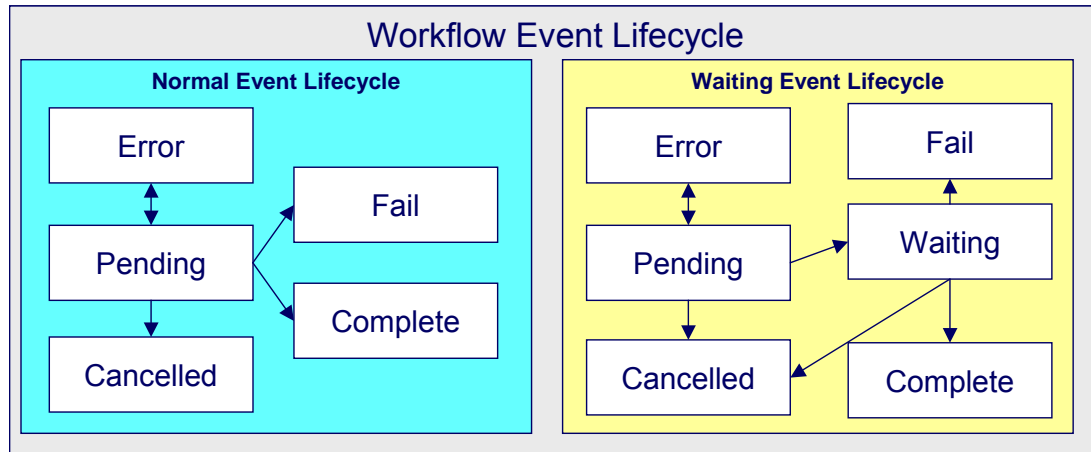
When the background process (referred to as **WFET**) executes a workflow event on its trigger date, it calls the activation algorithm associated with the event's event type. Because you can add and change activation algorithms at will, the variety of workflow events is infinite.

Refer to [Designing Workflow Event Types](#) for more information about activation algorithms.

Workflow Event Lifecycle

The lifecycle of a workflow event is dependent on whether the event has to wait on something before it can complete or fail. For example, an event that creates a field activity has to **wait** until the field activity is performed before it can be considered **complete**.

The following diagram shows the possible lifecycle of a workflow event:



The following points explain the lifecycle of workflow events of the *normal* variety:

- *Normal* events are initially created in the **pending** state.
- On a **pending** event's trigger date, the workflow event activation process (referred to as **WFET**) executes the event's activation algorithm.
- An event's activation algorithm may cause a **pending** event to become in **error**. For example, if an event used to create a field activity can't find a valid service point, the algorithm could change the status of the event to be in **error**. A user should correct the cause of the error and then change the event's status back to **pending**. The **pending** event will be processed the next time the activation program runs.
- An event's activation algorithm may cause a **pending** event to **fail**. For example, if an event used to validate a notification detects invalid information (e.g., an incoming notification is missing the customer's account number), the event will **fail**.
- A **pending** event becomes **complete** when the event's activity is successful. For example, if an event used to validate a notification determines the notification is valid, the event will **complete**. A user may manually change the status to **complete** if the event type indicates that manual completion is allowed.

For more information about a workflow event's trigger date, refer to [Workflow Event Dependencies & Trigger Date](#).

- A **pending** event will be **cancelled** automatically by the system if the workflow process is cancelled (a workflow process will be **cancelled** if an event fails or at the discretion of an operator). A user may **cancel** an event at will. Refer to [How Are Workflow Events Canceled](#) for more information.

The following points explain the lifecycle of workflow events of the *waiting* variety:

- *Waiting* events are initially created in the **pending** state.
- On a **pending** event's trigger date, the system executes the event. For example, if an event is used to create a "special read" field activity, on the event's trigger date, the field activity is created.
- An event's activation algorithm may cause a **pending** event to become in **error**. For example, if an event used to create a field activity can't find a valid service point, the algorithm could change the status of the event to be in **error**. A user should correct the cause of the error and then change the event's status back to **pending**. The **pending** event will be processed the next time the activation program runs.
- If the activation algorithm did not cause the event to become in **error**, the event's status is changed to **waiting** while the system waits for the field activity to be performed.

For more information about a workflow event's trigger date, refer to [Workflow Event Dependencies & Trigger Date](#). For more information about what an event might wait on, refer to [Waiting Events And Their Waiting Process](#).

- A **waiting** event becomes **complete** when the system sees that the thing that it's waiting for is finished.
- A **waiting** event **fails** when the system sees that the thing that it's waiting for didn't complete successfully. For example, an event that sends a confirmation request to a service provider asking if it's OK for a customer to switch suppliers would **fail** if the service provider denies the request.
- A **waiting** event will be **cancelled** automatically by the system if the workflow process is cancelled (a workflow process will be **cancelled** if an event fails or at the discretion of an operator).
- A **pending** event will be **cancelled** automatically by the system if the workflow process is cancelled (a workflow process will be **cancelled** if an event fails or at the discretion of an operator).

Refer to [Workflow Processes Can Have Multiple Branches](#) for more information about event transition in a process with multiple branches.

Workflow Event Dependencies & Trigger Date

The trigger date of most workflow events is blank when they are first created. This is because the trigger date can only be set when ALL of the preceding workflow events on which it depends are complete. An example will help explain why this design is necessary. Consider the following example that shows a simple workflow process and its events:

Event Number	Workflow Event	Dependent On Event(s)	Trigger Date Set To X Calendar Days After Completion Of Preceding Events
10	Validate new customer notification	N/A – first event	0
20	Set up new customer and meter	10	0

30	Send notification confirming new customer	20	0
40	Send welcome letter	20	0

This workflow process is meant to implement the following:

- On the first day, validate the incoming notification (the one that tells us about a new customer).
- If validation is successful, set up the new customer and their meter in the system.
- After the new customer and meter are set up, send a notification to the requester that everything has been set up. Also, send a letter to the customer.

The problem is that you don't want to execute event 20 until event 10 is complete. This is achieved by indicating event 20 is dependent on event 10. The system will only populate event 20's trigger date when event 10 is complete. Similarly, you can't set the trigger dates of events 30 and 40 until the customer has been set up (event 20). So, when you set up a workflow event, you must indicate the dependent events. If you only want the next event to trigger X days after the completion of earlier events, you can indicate such.

Bottom line. The trigger date of a workflow event is set to the current date plus X days where X is the number of calendar days defined on the workflow event. If this date is not a workday for your organization, the trigger date will be set to the next workday. If the resultant date is the current date (because X is zero), the event will be activated immediately.

Refer to [How Are Workflow Events Completed?](#) for information about how these events are executed.

Workflow Events May Be In Error

As explained under [Workflow Event Lifecycle](#), when the background process **WFET** executes an event's activation algorithm, this algorithm may cause a **pending** event to become in **error**. For example, if an event used to create a field activity can't find a valid service point, the algorithm could change the status of the event to be in **error**.

For every workflow event that's in error, a record is written to the [workflow event exception](#) table. To view the errors,

A user should correct the cause of the error and then change the event's status back to **pending**. The **pending** event will be processed the next time the activation process runs.

Refer to [Errors versus Failure](#) for information to help you differentiate between events that have **failed** versus those that are in **error**.

Some Workflow Events May Fail

Some workflow events may fail. For example:

- An event that validates an incoming notification may result in failure if the notification contains invalid information.
- An event that asks for the confirmation from a distribution company may fail if the distribution company rejects the request.

Refer to [Errors versus Failure](#) for information to help you differentiate between events that have **failed** versus those that are in **error**.

The background process that is responsible for activating events (referred to as **WFET**) is the process that can cause an event to fail (failure can happen during the activation algorithm on the workflow event). When an event fails, the system:

- Updates the workflow process with message number and message parameters describing the validation problem.
- Sets the status of the event to **Failed**.
- Cancels the workflow process and its outstanding events.
- Calls the failure algorithm defined on the event's event type.

It's important to note that some types of events can't fail and therefore don't have a failure algorithm. For example, an event that creates a field activity can't fail, neither can an event that creates a welcome letter.

Refer to [Workflow Processes Can Have Multiple Branches](#) for more information about event failure in a process with multiple branches.

Errors versus Failure

As explained under [Workflow Event Lifecycle](#), an event's activation algorithm may cause a **pending** event to become in **error** or to **fail** (amongst other things). You control the exact state when you design your workflow event type activation algorithms.

The main differences between these two states is as follows:

- As described under [Some Workflow Events May Fail](#), a failed event causes the entire workflow process to fail (and it cannot be restarted).
- As described under [Events May Be In Error](#), a user can correct the cause of an error event's error and then change the event's status back to **pending**. The **pending** event will be processed the next time the activation process runs.

You should follow the following guidelines when designing your validation logic in your workflow event activation algorithms:

- If the cause of the problem is correctable by a user, you should set the state of the event to be in **error**.
- If the cause of the problem is not correctable by a user (e.g., you were interfaced information that cannot be corrected by your users), you should set the state of the event to **fail**.

Waiting Events And Their Waiting Process

Some events have to wait until something else happens before they can be **Completed** (or **Fail**). These types of events exist in the **Wait** state until the thing they are waiting for happens. For example, consider an event that creates a field activity – it has to wait until the field activity is complete before it can itself **Complete**.

Every type of event that waits for something else to happen before it completes or fails must have a corresponding background process that monitors the thing on which the event is waiting. We refer to background processes that perform this monitoring as Waiting Processes.

There will be a Waiting Process for every type of event that has to wait for something to happen. The specific background process is defined on the workflow event type. For more information, refer to [Designing Workflow Event Types](#).

The following points describe the responsibilities of a Waiting Process:

- Check on the thing on which the event is waiting. For example,
 - An event that creates a field activity has a Waiting Process that checks on the state of the field activity.
 - An event that creates a request to confirm a customer's request to switch suppliers has a Waiting Process that checks if the confirmation is accepted or rejected.
- Change the state of the event to **Complete** or **Fail** based on what transpired. For example,
 - When the field activity completes, the Waiting event can Complete
 - The acceptance or rejection is received, the Waiting event can Complete or Fail
- Detect that the event has been waiting too long and do something. For example, the Waiting Process could:
 - Create a To Do entry (this might be useful if you need an operator to do something)
 - or, create an outgoing notification informing the sender of the incoming notification that something is wrong
 - or, **Fail / Complete** the event (you may be able to automatically assume success or failure if an event waits longer than a predefined limit)
 - or, execute the event again and reset the base time on the event (this might be useful if the event initiates an outgoing notification to ask permission from some other service provider – if it waits too long, you could simply create another outgoing notification)
 - or, whatever else you can think of developing in the process

How Are Workflow Events Canceled?

The background process that is responsible for activating events (referred to as **WFET**) automatically **cancels** workflow events when an event **fails**.

A **pending** event will be cancelled automatically by the system if the workflow process is cancelled (a workflow process will be cancelled if an event fails or at the discretion of an operator).

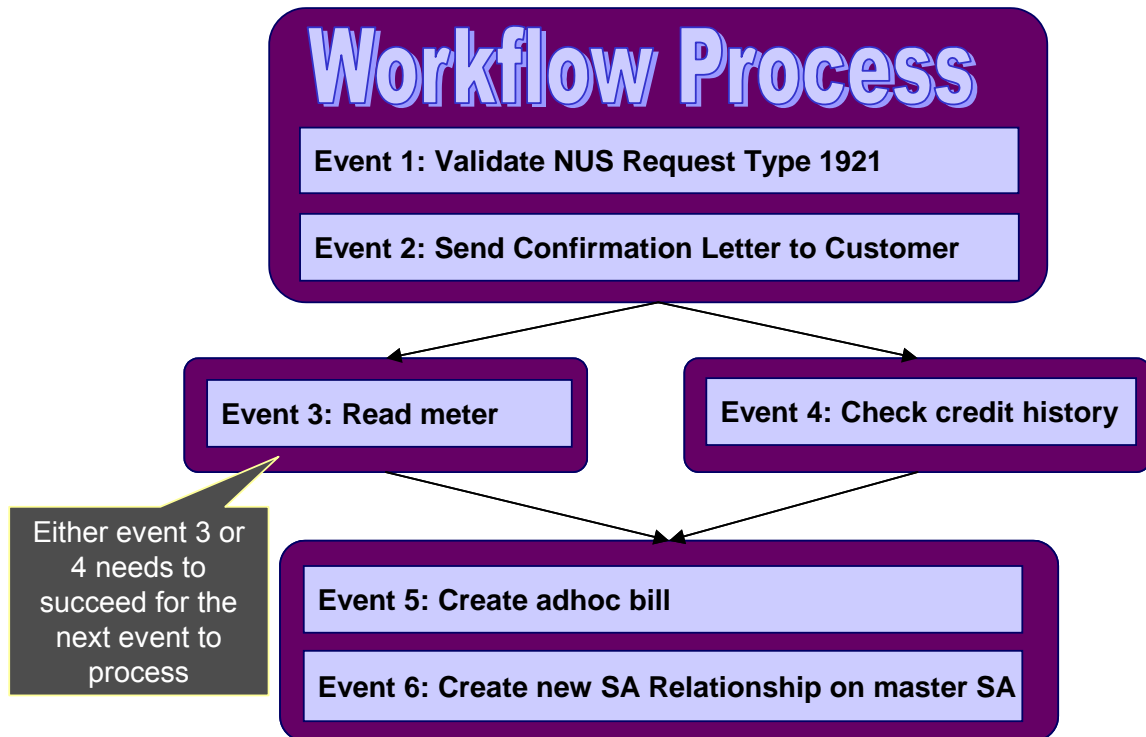
In addition, an operator may cancel a workflow event at will.

An event will be cancelled by the background process that is responsible for activating events (referred to as **WFET**) when it detects that ALL of its earlier, dependent events are cancelled. This is important to understand if your organization has [Workflow Processes With Multiple Branches](#).

Workflow Processes Can Have Multiple Branches

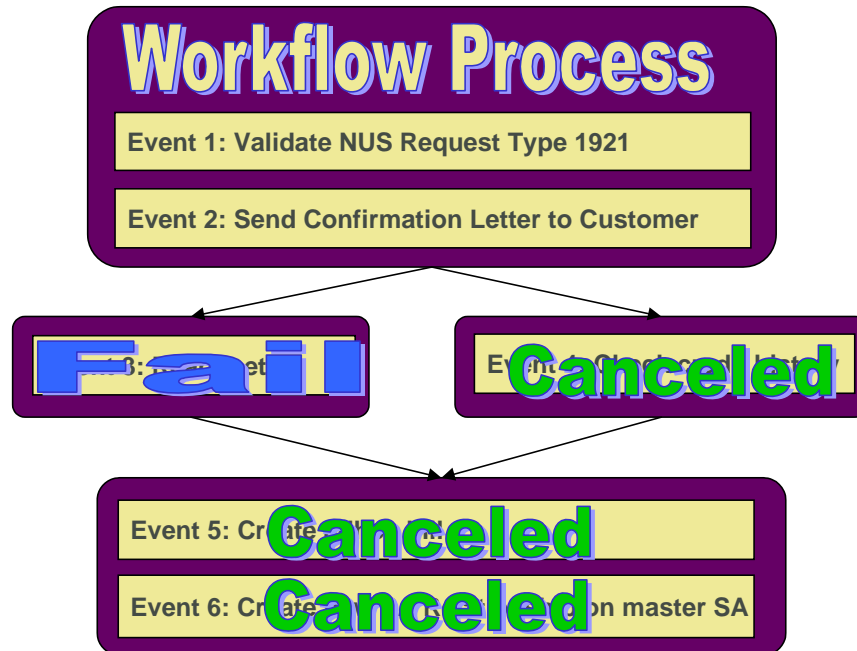
Using the workflow event dependencies, a workflow process may have multiple branches. There are several reasons why you may want to set up a process to contain multiple branches:

- There may be events that can run in parallel. This is useful if the related tasks take time to execute. For example “Read Meter” and “Send Confirmation and Wait for Response”. Both can be executed in parallel and event 5 will execute based on the outcome of events 3 and 4. The following is an example of such a workflow process.

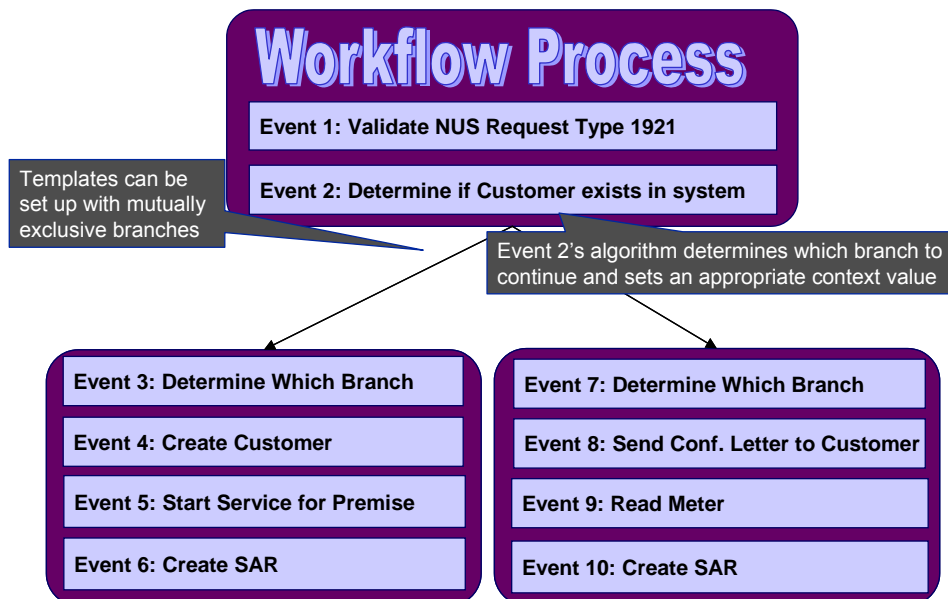


In the example above, if event 3 is canceled and event 4 completes, (or if event 4 is canceled and event 3 completes), event 5 will proceed. However, if both event 3 and event 4 are canceled, all remaining events will be canceled.

It should be noted that if either event 3 or 4 fails, ALL events in the process will be canceled, including events in a different branch.



- You may have a business process that has some common events and some events that are mutually exclusive. Rather than setting up several processes, you can set up one process that branches based on specific criteria. For example, perhaps you get to a certain point in a process that differs based on whether or not a meter is installed. If there is a meter, you follow one branch and if there is no meter, you follow a different branch.

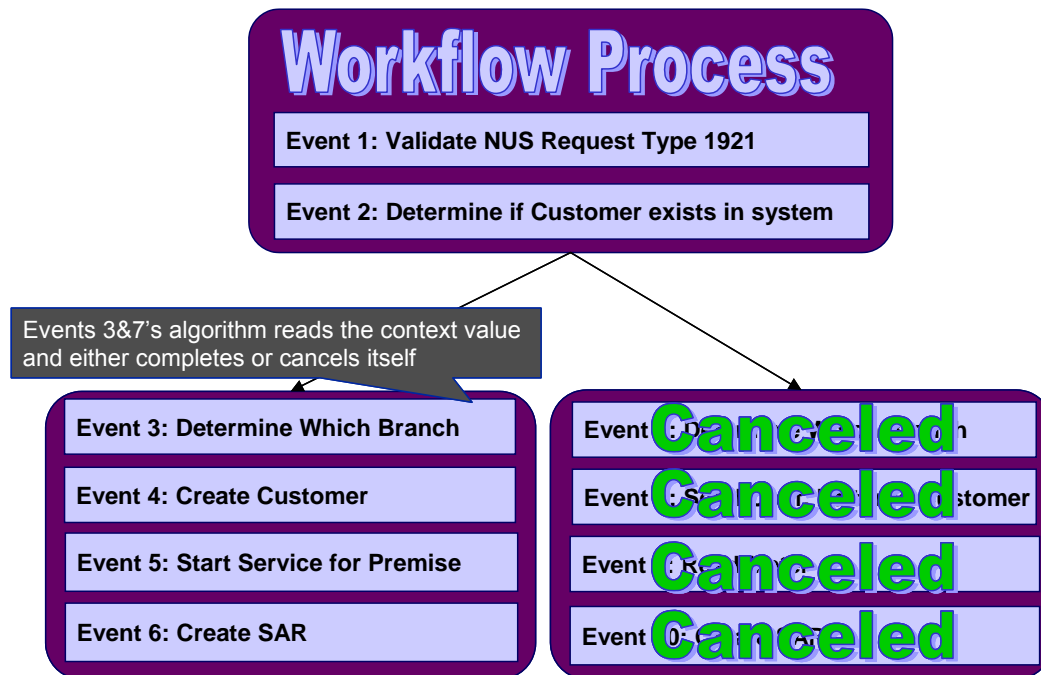


In the above example, event 2 determines if the workflow process is associated with an existing customer or a new customer.

- For a new customer; events 3, 4, 5 and 6 are executed.
- For an existing customer, events 7, 8, 9 and 10 are executed.

Event 2 does this by creating an entry in the Context collection indicating which branch should continue. (For example Context Type / Value of **BRANCH / A**). Events 3 and 7 should be special events whose purpose is to read the BRANCH context type and determine if its branch should continue or be canceled.

In this example, Event 3's algorithm will continue if the context value for **BRANCH** is **A** and Event 7 will only continue if the context value for **BRANCH** is **B**.



Note. The above diagram and description indicates the use of context records to determine which branch to continue. If your implementation chooses to use characteristics instead, the same logic above may be implemented using characteristics as well.

The Big Picture of Notification Processing

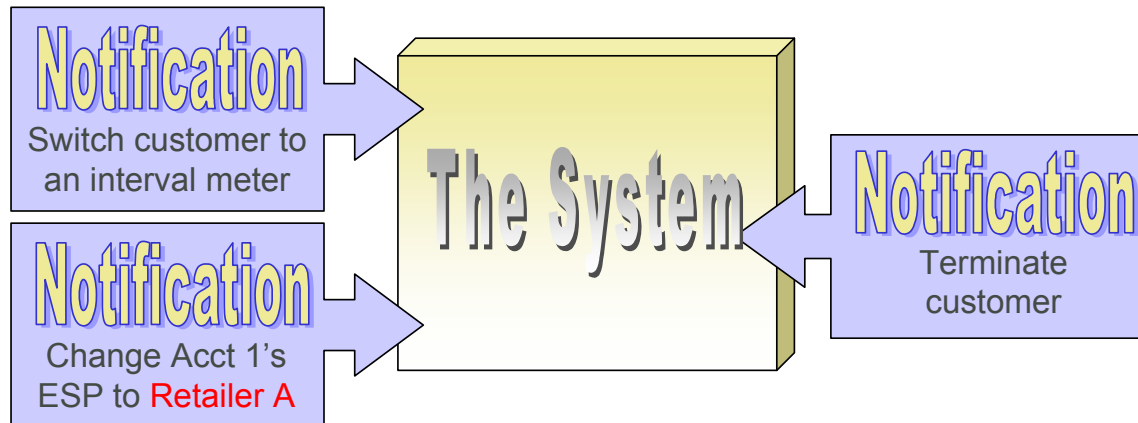
We use the term “notification” to reference the electronic transactions that you exchange with third parties when:

- They need information about a customer.
- They need to change something about a customer.

For example, an electric service provider may send a notification to an electric distribution company when a customer elects to use them as their energy provider.

You may have to support a wide variety of notifications. For example:

- You may receive a notification that tells you to switch a customer's energy service provider to a different provider (if you are a distribution company).
- You may receive a notification that asks for the last 24 months of consumption history for a customer (if you maintain consumption).
- Etc.



When a notification is received, the system responds by creating a workflow process. The related workflow process contains workflow events and it is the events that actually do whatever needs to be done (e.g., change the customer's service provider, create field activities, stop service, etc.).

In addition to incoming notifications, the system creates outgoing notifications when:

- It needs to respond to an incoming notification. For example, if an incoming notification requests a customer's consumption history, the system must send the consumption history by creating an outgoing notification.
- It needs to apprise a third party that something has changed about the customer. For example, if a customer stops service, the system must tell the various service providers of such.

The topics in this section describe how notifications are interfaced into and out of the system.

Contents

- [Uploading Notifications Into The System](#)
- [What Type Of Workflow Process Is Created?](#)
- [How Are Notifications Sent Out Of The System?](#)
- [System Conditions May Trigger Notification and Workflow](#)
- [The Lifecycle of Notification Download Staging](#)

Uploading Notifications Into The System

The following diagram illustrates the steps involved in the uploading of notifications into the system.



When a notification's electronic transaction is received, it must be inserted into the notification upload staging (NUS) table. Why? Because the system periodically looks for pending records in the NUS table and attempts to create a workflow process to implement the notification's request.

It's important to be aware that the events in a workflow process execute the notification request. To help solidify this concept, consider this – the first event in a workflow process typically validates the attributes on the NUS record.

Refer to [Notification Upload Background Process](#) for more information about the upload process.

What Type Of Workflow Process Is Created?

Given that you can receive many different types of notifications (e.g., return consumption, switch supplier, stop service), it should make sense that each type of notification upload staging (NUS) record will probably require a different type of workflow process to implement its request. The system uses the NUS record sender's external system Id and a notification upload type to determine the type of workflow process. This works as follows:

- Every external system references a workflow process profile.
- A workflow process profile defines the type of workflow process template to use to process a given notification upload type.

Different workflow processes for the same type of notification. It's probably obvious why different notification types result in different workflow processes. You may wonder why different workflow processes would be needed by two senders who submit the same type of notification? The reason is that different senders may have different types of computer systems that handle their notification processing (or no computer system at all). For example, you may have to respond by fax to a sender who doesn't have a sophisticated computer system, whereas you may send an electronic transaction to a sender who is technically advanced. The system allows you to create different workflow process profiles for each response mechanism (e.g., sophisticated vs. fax). You then associate the appropriate workflow process with each sender.

How Are Notifications Sent Out Of The System?

In addition to processing incoming notifications from third parties, the system also sends notifications to third parties to provide them with information (and to ask them for information). It is also possible to use notifications to send information to other applications within your company.

A Notification Download Staging (NDS) record is created for every notification that is sent to the outside world. NDS records are created by workflow events (i.e., the event's activation algorithm creates a NDS record). Consider the following examples:

- Many workflow processes exist to process incoming notifications. These processes typically contain workflow events that create NDS records to communicate with service providers. If you look at the workflow processes described under [Designing Workflow Process Templates](#), you will see many such examples.
- When something noteworthy happens, the system may need to tell a third party about it. Or perhaps you have third party software, which contains information from the system and your company needs to keep the information in sync.

Refer to [System Conditions May Trigger Notification and Workflow](#) for more information about triggering notification download staging records from within the system.

Notification and XAI

NDS records may also be processed using the XAI tool. Refer to [Outgoing Messages](#) for an overview of communicating outgoing messages via XAI.

The remaining sections describe the capability to send outgoing messages from the system to another application using the Notification and Workflow engine and XAI.

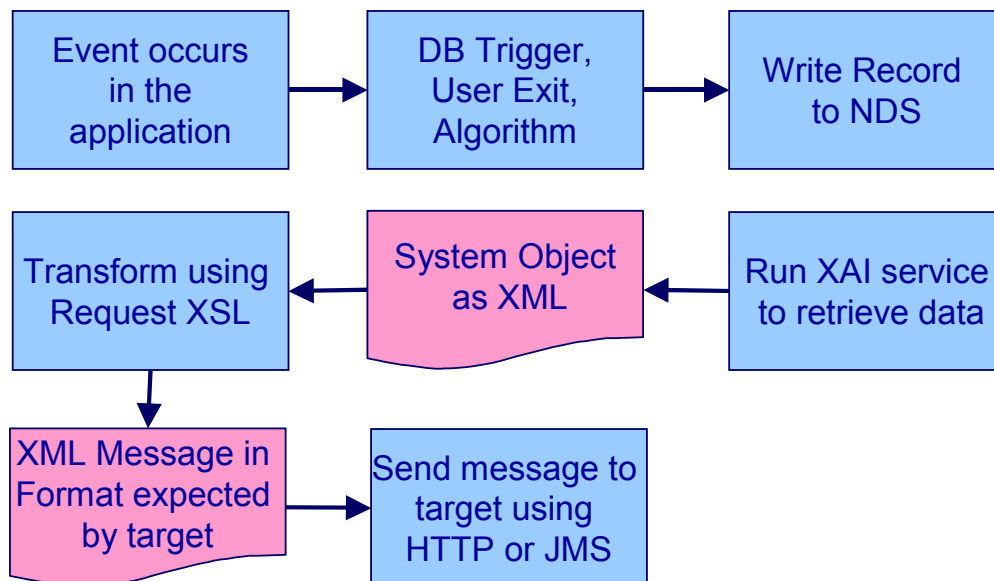
Contents

[Near-Real Time Outgoing Messages](#)

[Real Time Outgoing Messages](#)

Near-Real Time Outgoing Messages

The following diagram illustrates the flow of near-real time messages:



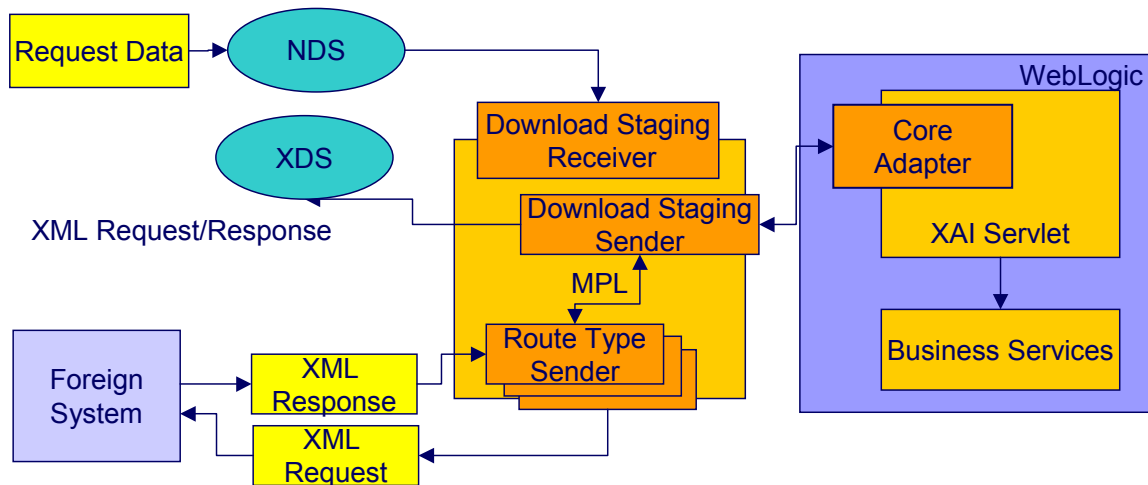
The following points describe the diagram

- When an event occurs in the system, you can initiate an outgoing message by storing a [notification download staging](#) (NDS) record. NDS records for outgoing messages have a special processing method flag of **XAI**.

Refer to [System Conditions May Trigger Notification and Workflow](#) for information about creating notification download staging records.

- Once a message is stored on the NDS table, the download staging receiver reads it and invokes XAI to extract data from your product. An XAI inbound service is used to retrieve the full information for the object related to the message.
- The download staging sender takes the response from the executor (the system object as XML) and continues the processing. It uses a request XSL on the route type to transform the message to a format expected by the target. It then sends the message to the target using the sender on the XAI route type.

The following diagram illustrates the process:



The responsibilities of the download staging receiver and download staging sender are provided in detail below.

Contents

- [Download Receiver](#)
- [Download Staging Sender](#)
- [Asynchronous Responses to NDS Messages](#)

Download Receiver

The download staging [receiver](#) processes records in the NDS table that have a processing method flag equal to **XAI** and a status of either **pending** or **retry**. It also uses the priority flag on the NDS type and the creation date/time on the NDS to process records according to their priority.

The process is referred to as "near real time" because your download staging receiver should be continuously checking for new records in the NDS table.

When it finds a record, it invokes XAI (via the [executor](#)) providing the XAI inbound service.

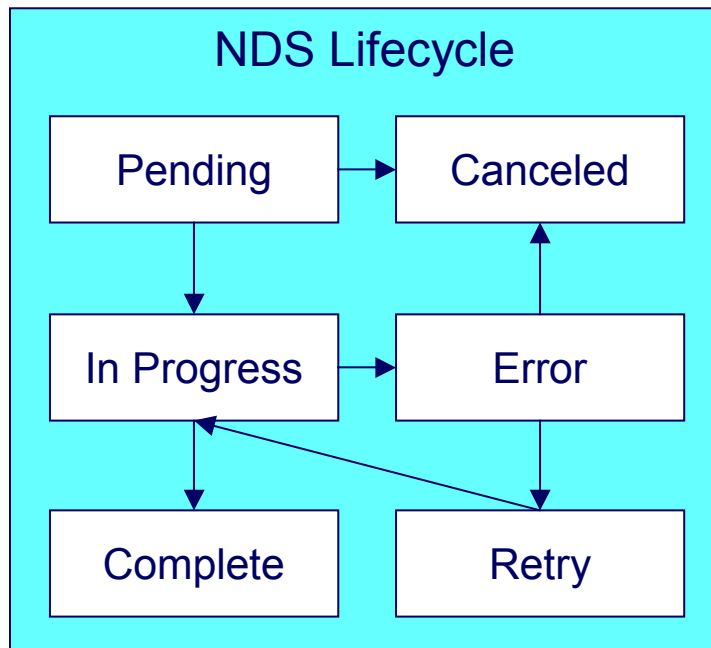
The following section provides more information about the NDS lifecycle.

Contents

- [Lifecycle of Notification Download Staging](#)
- [Building the XML Request](#)

Lifecycle of Notification Download Staging

The download receiver processes NDS records based on their status. The following diagram describes the lifecycle of an **XAI** type NDS.



The following points describe the diagram:

- Records are created in **pending** status. A user may transition the record to the **canceled** state.
- The MPL processes records in **pending** and **retry** status. While the record is being processed, it is changed to **in progress**.
- Based on the results of the processing, records could be transitioned to **error** or **complete**.

Note. An NDS is set to error if there is a problem with the NDS or if any of its related XDS records are in error.

- A user may cancel a record in the **error** state or change the status to **retry** to have the MPL try again.

Building the XML Request

To build the XML request for the outgoing message, the download receiver needs the following information:

- The XML request and response schemas used to retrieve information about the record related to the request. For example, if the outgoing message is to inform an external system that a person's name has changed, the XML request schema must retrieve the appropriate person record.

The **NDS type** related to the NDS defines an XAI Inbound Service that references an appropriate XML request schema.

- The data related to the request. In this example, the NDS record should be stored with the ID of the person record that changed.

The person ID is stored as NDS context, with an appropriate Context Type, like **PERID**.

- XPATH information to tell the system where to plug in the related data into the request schema. In this example, when building the XML request, the person id must be plugged into the appropriate location in the XML request schema prior to executing the request.

The [NDS type](#) related to this NDS defines a collection of context types. For each context type, you define an [XPath](#) used to indicate where to substitute the context data.

Refer to [Configuring the System for NDS Messages](#) for more information.

Once the request schema is build, the receiver invokes XAI (via the executer) to execute the XML request. In our example, it calls the appropriate Person service and the XML response includes the extracted person information.

Download Staging Sender

This piece is somewhat confusing, because the download staging sender behaves differently than other senders. [Senders](#) are typically responsible for

- Routing a request to an appropriate target
- In the case of the upload staging sender and the staging control sender, the sender is responsible for updating the status of the appropriate table.

In the case of the download staging sender, it's responsible for processing the "response" to the XAI executer, whose task was to build the XML request.

- If there is a problem executing the request, the download staging sender updates the NDS record in **error** and creates an [NDS exception](#) record.
- If the executer was successful, the "response" is an XML request in a format understood by your product. The sender must perform the following steps to help process the outgoing message:
 - Transform the request to a format expected by the target system(s)
 - Route the outgoing message to the target
 - Create an XAI download staging record for each message sent out. (There is typically only one, but it is possible for the outgoing message to have [multiple destinations](#).)

If any errors are found during any of these steps, the status of the NDS record is set to **error** and a record is written to the [NDS exception](#) table. If the error is related to routing the message, the XDS is created in **error** and a record is written to the [XDS exception](#) table.

Configuration required. The above explanation assumes that you have correctly configured your download staging receiver to reference the download staging sender. Refer to [Designing Responses for a Receiver](#) for more information.

Routing Optional. It is possible that an [NDS message does not require routing](#) information. In this case the download staging sender simply updates the NDS record's status

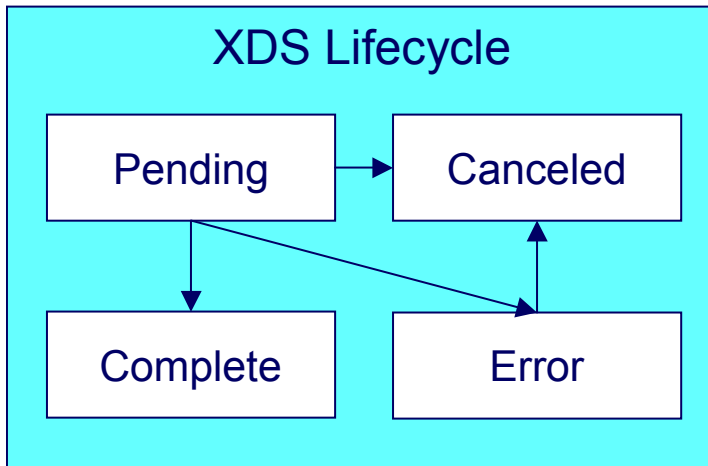
The following sections provide further detail.

Contents

- Lifecycle of XAI Download Staging
- Transform the Request
- Routing the Message
- Multiple Message Destinations
- An NDS Message That Doesn't Require Routing

Lifecycle of XAI Download Staging

The following diagram illustrates the lifecycle of an XAI download staging record.



An XAI download staging record is created in one of the following ways:

- When the download staging sender continues the processing for the outgoing message, it attempts to transform and route a message for each XAI route type indicated on the notification download profile. An XDS is created to record the message sent.
 - If the message was sent successfully, the XDS is created in **complete** status.
 - If an error occurred when sending the message, the XDS is created in **error** status and an [XAI download exception](#) is created.

Automatic Resend. If you have configured the system for [automatic resend](#) and the system detects that the error is due to the sender being unavailable, no XDS record is created in this case.

- When the system attempts to send a message [real time](#) but the external system is unavailable, the system may create a **pending** NDS and a **pending** XDS record.
- A user may **cancel** a **pending** or error XDS record.

If you have resolved the error for an XAI download staging record, change the status of the related NDS record to retry. When MPL processes the NDS again, it deletes all XDS records in error and attempts to send them again.

Transform the Request

The download staging sender takes the XML response with the extracted data and attempts to transform into a format accepted by the target. It needs an appropriate [XSL transformation](#) script.

The XSLT is defined on an [XAI Route Type](#), which is referenced on the [download profile](#) of the service provider referenced by the notification download staging record.

Note. The NDS context information is available to the XSL. Refer to [Substitution Parameter for NDS Messages](#) for more information.

Routing the Message

In addition to defining the appropriate XSL transformation scripts, the XAI route type also references a [sender](#) that tells the system how to route the message, for example it may be routed using a JMS queue.

Inactive Senders. If the message is targeted to an *inactive* sender and you have enabled [automatic resend](#), the message is ignored and an MPL log entry is created indicating that the message was not sent.

If the target supports synchronous communication (for example, an HTTP sender), the [route type](#) may be configured to receive an acknowledgement. This indicates to the download staging sender to wait for a response from the sender on the route type.

The download staging sender creates an appropriate XDS record with the XML request document and with a status of *complete* or *error* based on the results of the communication with the target. If the target has sent a synchronous response, the XML response is also posted on the XDS.

Automatic Resend. If you have configured the system for [automatic resend](#) and the system detects that the error is due to the sender being unavailable, no XDS record is created in this case and the NDS remains in *pending*.

If the target supports synchronous communication, you may indicate that a response to the outgoing message also requires action. For example, perhaps the outgoing message to an external system causes a new record to be created in that system. The response to the message may require an update to our initiating record to post the external system's identifier for the record on their side.

To enable this logic, you must check the Post Response switch on the [route type](#). If this has been checked, the download staging sender creates an XAI upload staging record (and a staging control record) and it links the XDS record as a foreign key. The XUS record is processed along with other uploaded messages, to invoke an appropriate service.

XML Response. If the Post Response switch on the route type is turned on, the XML response that is captured is the XML that results after applying the Response XSL. If the Post Response switch is not turned on (i.e., the response is a simple acknowledgement), XML response is posted without applying a Response XSL. (In fact, for simple acknowledgements, there is no need for a Response XSL.)

Asynchronous Responses. Refer to [Asynchronous Responses to NDS Messages](#) for information about processing responses to messages sent asynchronously.

Refer to [Configuring the System for NDS Messages](#) for more information about setting up these controls.

Multiple Message Destinations

If your message must be sent to more than one external system, a different XAI route type is required for each system that the message must be sent to so that the sender and XSLT for each external system can be defined.

The recommended procedure for sending a message to multiple destinations is to

- Generate an NDS for each destination where each NDS defines the same NDS type but different service providers. In other words, the service provider represents the destination.
- Each service provider references a [notification download profile](#) where the appropriate XAI route type is defined for each NDS type.
- When this NDS record is processed, the sender information defined for the XAI route type is used to route the message appropriately and an XAI download staging (XDS) record is created.

Note that it is possible for the download profile's formatting method to reference more than one XAI route type. This is perhaps another configuration that could be used to send a message to multiple destinations. In this case an XDS record is created for each XAI route type to track the XML request. However, it is not possible to track [asynchronous responses](#) to multiple messages using this mechanism. Defining multiple route types for a download profile formatting method should only be done if you do not expect an asynchronous response.

An NDS Message That Doesn't Require Routing

If an NDS message is being sent to one of your own external systems and this system may be accessed directly, you could design your message such that XAI routing is not necessary. For this scenario, the assumption is that the application service that is invoked by the XAI inbound service completes all the necessary steps to update the external system.

If this is the case, you are not required to define an XAI Route Type on the appropriate notification download profile entry. In addition, no XAI sender is required for this external system.

The download staging sender's sole responsibility for a message of this type is to update the NDS status.

Asynchronous Responses to NDS Messages

If you send an asynchronous NDS message and you expect a response, the response is received as an inbound message. In order to recognize that an inbound message is a response to an NDS message, there must be some handshaking. In other words our system must pass a unique identifier of our message to the external system and the external system must include that identifier in the response.

Contents

- [Populating the Unique Message Identifier](#)
- [Recognizing the Inbound Message Response](#)

Populating the Unique Message Identifier

As described in [Building the XML Request](#), the information that may be included in the XML request we build is information linked to the NDS record as context. Therefore, the unique identifier you want to send to the external system must be stored as a context record. (Let's call this context type Message ID). Because the notification download staging ID is a unique identifier, you may choose to populate Message ID with the NDS ID. However, it is possible that the external system requires an identifier in a different format. For example, the NDS ID is a 12-byte number. Perhaps the external system can only receive a 10-byte number for the Message ID.

It is the responsibility of the mechanism that creates the NDS record (i.e., the algorithm or user exit or database trigger) to create a context entry for Message ID with an appropriate value that is compatible with the target system.

When creating your XML request schema, if you expect a response to your message, you must include the message ID as an attribute in the XML request. Because the message id is not part of the product information that your request schema is extracting, it should be defined as a [private attribute](#). Once the XML is created, it is the responsibility of the XSL to transform the Message ID into the target specific XML element.

Only One Route Type. Because the unique Message ID for an NDS message is linked to the NDS record via context, it is not possible for the system to handle multiple route types (and therefore multiple senders) for a single message IF an asynchronous response is expected. The reason is because all senders receive the same unique Message ID and if each sender responds, the system is not able to determine which response is received for which sender.

Recognizing the Inbound Message Response

When the external system sends an asynchronous response to your NDS message, it is received as an [inbound message](#). This inbound message must include the Message ID that you sent so that you can recognize this as a response. In addition, the XSL transformation must populate the identifier of the external system sending the message.

The following points describe the detail related to receiving an inbound message:

- When a response comes in, the message is translated using an XSLT script. The XSLT must map the message ID, which is the unique identifier of the NDS message that this is a response to, to the XML element **MessageID**. It must create an attribute with the XML element **ReplyToMessageSource** populating the value with an appropriate [external system](#). This value should correspond to the external system code of the service provider related to the NDS message.
- The XML request of the inbound message is processed as normal.
- The existence of the special message elements in the XSL indicates to XAI that XML response processing is required. It looks for a **complete** NDS with a service provider whose external id matches the **ReplyToMessageSource** and with a context type of **Message ID** whose context value matches **XAI Message ID**.
- When the correct NDS is found, the system finds the related XAI download staging record and populates the XML response with the XML request of the incoming message.
- If the XDS request is not associated already with a response or the latest associated response is in **error** status stamp the current response on the XDS request record.

- If the response came into the system as an XAI upload staging (XUS) record, the foreign key to the XDS record (NDS ID and XAI route type) to which it is responding is linked to the XUS.
- If the response did not come in as an XAI upload staging record (for example, it came in via JMS queue), an XAI upload staging record and an XAI staging control record are created in **complete** status as an audit and the foreign key to the XDS record is linked to the XUS.

Errors Linking the XDS. If the system has a problem finding the appropriate XDS record to link to the incoming response, the inbound message is still processed. A log entry is added to the XAI trace file reporting the error in identifying the correct XDS record.

Real Time Outgoing Messages

This section describes the capability to send real time messages from the system to another application. The functionality works in conjunction with the near real time NDS message configuration.

Contents

[Real Time Message Engine](#)

[Sending Real Time Messages in Near Real Time](#)

Real Time Message Engine

The system provides an "engine" to communicate real time messages to an external system. This engine may be invoked by a user exit on any user interface in the system. The real time message engine receives an NDS type and a service provider as input. Using this information, the engine finds the appropriate XAI route type for the service provider's notification download profile. The route type indicates the XSLT and the XAI sender.

HTTP Sender. In the current release of the product, only senders that communicate via HTTP are supported.

Unlike near real time messages, the XAI inbound service referenced on the NDS type is not used to build the XML request. Rather, the user exit that invokes the engine is responsible for extracting the appropriate data and passing to the engine a formatted XML document. The XSLT referenced on the route type should map attributes from XML document into a format expected by the target system.

The following points describe how the real time engine works:

- Once the XML request is built, the message is sent out based on the XAI route type's sender information.
- The system waits for a response for a predefined time out limit defined as a context entry on the [sender](#).
- When the response is received, the engine transforms the message using the response XSL defined on the route type.
- The XML response is then passed back to the user exit that called the engine. The user exit processes the response as needed.

Note, as described in Routing the Message, the route type includes a flag to indicate to the system if a response requires additional action. If so, an XUS is created. The real time message engine does not support this logic. It expects the user exit to process the response as needed.

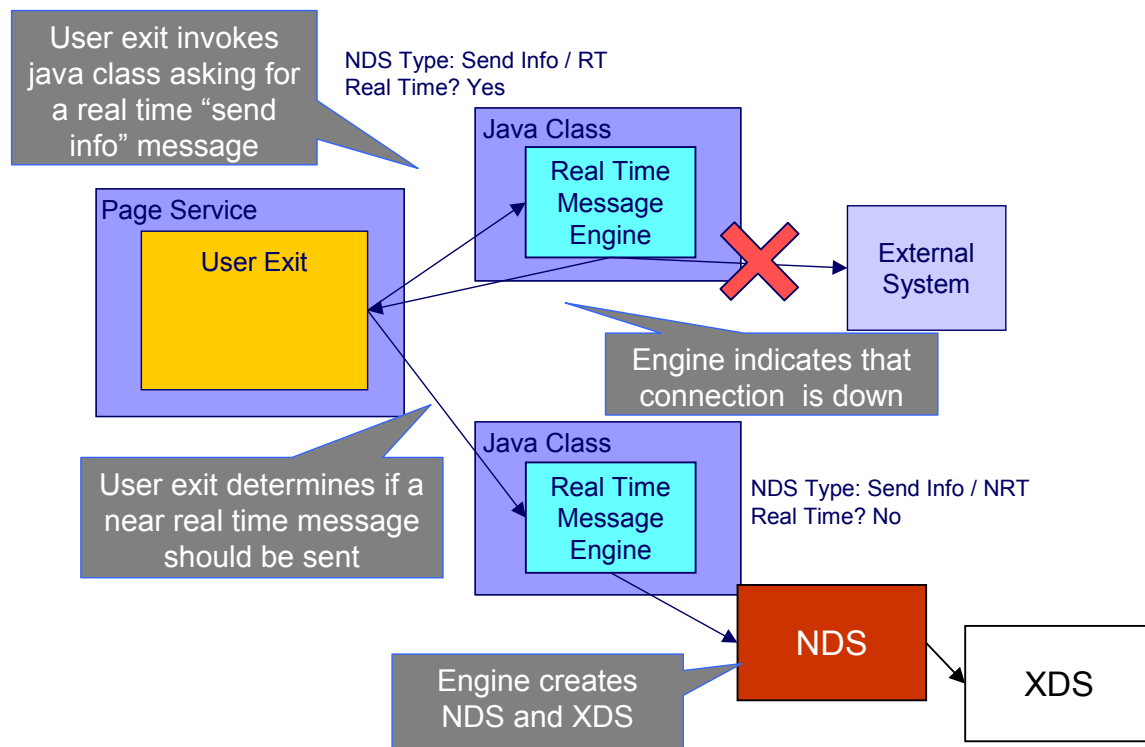
Sending Real Time Messages in Near Real Time

If the connection is not available when the real time engine attempts to communicate with the target, the engine may post the message to the download staging tables to send in "near real time" when the target system is available again.

Messages should only be posted to the staging table if the user does not need to wait for a response in real time. Consider the following examples:

- A real time message is used to request information from an external system to display to a user on a page. If the connection is down, it does not make sense to store a near real time message for this case because the user is not able to wait for the response to the request.
- There is a connection problem when the user attempts to send information to an external system. In this case, perhaps it makes sense to store a near real time message so that the information is sent to the external system when the connection is restored.

Note. The real time message engine does not make the decision to post a message near real time when the communication is unavailable. The responsibility lives solely with the user exit that calls the java class that invokes the engine. The following diagram illustrates a possible interaction between the user exit and the message engine:



The following points describe the diagram:

- When the user exit is informed that the real time message could not be sent, it is the responsibility of the user exit to decide if the engine should be invoked again to create a near real time message.

- The NDS type used for near real time messages is different than the one used for real time messages. The reason is because the XSL transformation script is different for real time vs. near real time.
- The NDS is created and marked *pending*.
- The XDS record is created and marked *pending*. The XML message that is ready to be sent to the sender is posted as the XML request for the XDS.

Because the XML request is already built, the NDS type used for the NDS should reference a special XAI inbound service. This service basically tells the download staging sender that the XDS already exists and doesn't need to be created.

System Conditions May Trigger Notification and Workflow

Outbound Messages. The mechanism for triggering communication to an external system described here does not take advantage of the configurable business object functionality that enables outbound messages to be triggered from a server-based script. Refer to [Near Real Time Outbound Message](#) for more information about communicating with an external system using outbound messages.

There may be many noteworthy occurrences in the system, which should trigger the notification and workflow process.

Some of these occurrences may need to be communicated to an external system. Examples of when this may occur are as follows:

- You use an external software package, which uses information from the system and you need to send a message to this software package when certain attributes of the customer change.
- You need to inform third parties when something changes about a customer. For example, if you are a distribution company, you probably advise a customer's energy supply company when their meter is changed or when the customer stops. Or if a customer, who has been referred to a collection agency, makes a payment, you will need to inform the agency of the payment.

In many scenarios, you may want more than just a notification to be sent out, but you would also require a workflow process. For example, perhaps when a customer's mailing address changes, you want to a) send a confirmation letter to the customer's old address and their new address; b) inform a third party service provider about the change; and c) send the update to an external marketing system. In this case, the act of changing the mailing address could trigger a workflow process, which would perform the above steps.

You may have other scenarios where a noteworthy condition in the system should trigger a workflow process, which does not require a notification to be sent out of the system. For example, perhaps your company follows a sophisticated procedure for starting service, which requires certain events to transpire before other events can occur. You could create a workflow process, which gets triggered upon starting service to handle this sophistication.

In summary, when something occurs in the system, you may need to

- Create a notification download staging (NDS) record. You would do this if you only need to send a message to an external system.
- Create a notification upload staging (NUS) record. You would do this if you need to trigger a workflow process to do one or more steps. One of the workflow events in this process could create a NDS if an external system needs to be informed.

Contents

[Creating a Notification Download Staging Record](#)

[Creating a Notification Upload Staging Record](#)

[Triggering Notification & Workflow from the Application](#)

Creating a Notification Download Staging Record

To create an NDS record directly, you need the following information:

- Service Provider (who will receive the outbound message)
- NDS Type

You also need to populate relevant data using either context or characteristics. For example, if a change to a person's name should trigger an outbound message, the Person ID must be referenced.

The process that creates the NDS must determine the appropriate service provider that should receive this information based on the business rules. As for the NDS type, this is typically "hard-coded" based on what has occurred to trigger the outbound message. For example, if the customer's name changed, then the NDS Type is set to something like "Name Change".

For any NDS records automatically created by the system in sample algorithms or in system processes, the NDS type is not hard-coded. Rather, the system uses a download type condition flag value to look up the appropriate NDS type. As a result, if an implementation chooses to use these sample algorithms, but prefers to use a different NDS type, it is possible. When designing your own NDS types that are referenced by NDS records created based on system conditions, you may choose to create new download type condition flag values as well.

Refer to [Designing Notification Download Types](#) and [Setting Up Notification Download Types](#) for more information.

Creating a Notification Upload Staging Record

To create an NUS record directly, you need the following information:

- External System (to point to a service provider)
- NUS Type

You also need to populate relevant data using either context or characteristics. For example, if a change to a person's name should trigger a letter to the customer and an outbound message, the Person ID must be referenced.

The process that creates the NUS must determine the appropriate external system based on the business rules. If the steps that need to be taken as a result of this system condition are based on an internal practice, the external system could be something like "SYSTEM". That external system would reference an appropriate workflow process profile that defines the appropriate workflow process based on the NUS type.

Determining the NUS type is similar to determining the NDS type as described in the previous section.

Refer to [Designing Notification Upload Types](#) and [Setting Up Notification Upload Types](#) for more information.

Triggering Notification & Workflow from the Application

You and your implementers must determine the noteworthy events that should trigger workflow and/or outgoing notifications based on your business practice. For example, you may communicate all meter exchanges and customer address changes to the customer's various service providers.

Next, you must assess how you are going to create the appropriate records:

- Determine if there is a plug-in spot that exists where you can create an algorithm to trigger the NUS or NDS. For example, if canceling a service agreement should trigger a notification, an SA Cancel algorithm plugged in to the SA type may be used.
- If there is no plug-in spot, determine if there is a user exit at the appropriate place for you to trigger the creation of an NUS or NDS
- You may also decide to use a database trigger to create the NUS or NDS

Once you have determined how and when to trigger the creation of notification records, have your implementers create the CM code to do the work.

The Lifecycle of Notification Download Staging

If a download staging record is routed to the outside world via XAI, a status is assigned to the download staging record.

Refer to [Lifecycle of Notification Download Staging](#) for more information.

Creating Notification and Workflow Procedures

A workflow process is created by copying the events defined on a workflow process template (a workflow process template contains the standard events).

Workflow processes can be used for any multi-event process. While this section describes how to design workflow processes to support incoming and outgoing notifications, workflow processes can be used for other multi-step processes. For example, you can use a workflow process to manage the steps involved with the installation and inspection of backflow devices at a water service point.

The topics in this section describe how to design and setup the tables that control your notification and workflow processing.

For more information about notification and workflow processing, see [The Big Picture Of Workflow Events](#) and [The Big Picture Of Notification Processing](#).

Warning! There are innumerable ways to design your notification and workflow procedures. Some designs will result in easy long-term maintenance, others will result in maintenance headaches. In this section, we provide information to help you understand the ramifications of the various options. Before you set up your production procedures, we encourage you to gain an intuitive understanding of these options by using the system to prototype the alternatives.

Contents

[Designing Notification Upload & Workflow Procedures](#)
[Designing Notification Downloads](#)
[Setting Up Notification and Workflow Procedures](#)

Designing Notification Upload & Workflow Procedures

The topics in this section describe how to design the tables that control your notification upload and workflow processing.

Contents

[Designing Workflow Process Profiles](#)
[Designing Notification Upload Types](#)
[Designing Workflow Process Criteria](#)
[Designing Workflow Processes To Process Incoming Notifications](#)
[Designing Workflow Processes To Deal With Invalid Sender or Notification Upload Type](#)
[Designing Workflow Process Templates](#)
[Designing Workflow Event Types](#)
[Designing External Systems](#)

Designing Workflow Process Profiles

The following sections describe how to design a workflow process profile. A workflow process profile controls the type of workflow process that will be created to process your incoming notifications. If you plan to use workflow processes outside of the notification upload process, you can skip to [Designing Workflow Process Templates](#).

You associate a workflow process profile with one or more external system. Whenever a notification is received from an external system, the system uses the workflow process profile to determine the type of workflow process to create to implement the notification.

The easiest way to design a workflow process profile is to choose a representative external system and design a workflow process profile for it by filling in the matrices below. After you've designed a profile, determine how many other external systems can use it. Then design the next external system's profile and determine who can reuse it. Repeat this process until all your external systems have a profile. Once the profiles are designed, you're ready to set up the workflow control tables.

Refer to [The Big Picture Of Notification Processing](#) for more information about how workflow processes are used to implement incoming notifications.

The topics discussed below will gradually complete the following matrix using a simple case-study. We recommend that you use the following matrix as your guide. When the matrix is complete, you're ready to set up a workflow process profile.

WF Process Criteria		
---------------------	--	--

Notif. Upload Type		

Designing Notification Upload Types

You will need one notification upload type for every type of notification your organization can receive from the various service providers. To “design” your notification upload types, you document the codes used by external systems to identify the transaction types on their notification upload staging records.

We have populated the Notification Upload Type column with a few classic examples that can be received by a distribution company:

WF Process Criteria Notif. Upload Type		
Retrieve a customer's consumption history		
Switch a customer to a different service supplier		
Switch customer to an interval meter		

Note. It is recommended that the unique identifier of your Notification Upload Types match the “transaction types” that are referenced on incoming notifications. If it is not possible to do this, the process that creates the upload staging records must map the external transaction types to the notification upload types that you define in the system.

In addition, as described in [System Conditions May Trigger Notification and Workflow](#), you should evaluate any condition that should cause an NUS to be created (to eventually create a workflow). A new NUS Type should be defined for each condition. For each of these, consider creating a new value for the upload condition flag so that the NUS type is not hard-coded in the system process that creates the NUS.

Navigating to Related NUS Extension

When designing the types of notifications your organization may receive, you must determine where you plan to store the detailed information related to the incoming transaction: as context linked to the NUS, as characteristics linked to the NUS or by using an extension record.

Refer to [Process X - Populate Notification Upload Staging](#) for information about the different options available.

If you choose to capture the detailed information in a new extension record with a new user interface, then in order to be able to drill down to the related extension record, the following steps are required.

- Add a new [lookup](#) value for your new extension record for the field **NT_UP_EXTSN_FLG**. (This step is needed regardless of whether you want to drill down to the related record.)
- Add a [navigation option](#) pointing to the appropriate navigation key for your new transaction. This navigation option must reference a usage type of **notification upload type**.
- Indicate the appropriate lookup value and navigation option on the appropriate notification upload type.

Designing Workflow Process Criteria

The matrix's second dimension is dependent on workflow criteria algorithms. Workflow criteria are confusing. Think of them as optional conditions that, if met, will cause a different type of workflow process to be started when a given notification upload type is received.

You must define a **Default** criterion in case none of the override criteria are met. You MAY have override criteria if different situations result in different types of workflow processes. For example, let's assume some notification upload types have different workflow processes for industrial customers as compared to all other types of customer. This assumption necessitates the introduction of an override workflow process criteria; we'll call it **Industrial Customer**.

WF Process Criteria Notif. Upload Type	Default	Industrial Customer
Retrieve a customer's consumption history		
Switch a customer to a different service supplier		
Switch customer to an interval meter		

The workflow process criteria are limited by your imagination (and business requirements). We have provided the workflow process criteria you see above as an example; we don't expect you'll be able to use the exact conditions we supply. Your conditions will be based on any number of factors.

New workflow process criteria may require programming. See [How To Add A New Algorithm](#) for more information.

Designing Workflow Processes To Process Incoming Notifications

The next step involves populating each cell in the matrix with the workflow events that should be executed when the system receives a notification identified with a given notification upload type. If override criteria aren't relevant for a given notification upload type, we will mark the cell as "N/A".

WF Process Criteria Notif. Upload Type	Default	Industrial Customer
Retrieve a customer's consumption history	Validate notification - consumption history request	N/A (meaning that industrial customers use the Default criteria)

	Confirm requester is a valid service provider for the customer's service. Create notification download – send consumption history	
Switch a customer to a different service supplier	Validate notification – supplier switch Confirm requester is a valid service provider for the customer's service. Check with current supplier if the switch is allowed. Switch suppliers.	N/A (meaning that industrial customers use the Default criteria)
Switch customer to an interval meter	Validate notification – change customer to an interval meter Create notification download – reject request (only industrial customers can have an interval meter)	Validate notification – change customer to an interval meter Confirm requester is a valid service provider for the customer's service. Create field activity to exchange current meter. Change rate on exchange date.

Note. Notice that the first event in each cell typically validates the notification upload staging record.

At this point, the matrix is complete. Before you're ready to design your workflow process templates you must design the processes described in the next section.

Designing Workflow Processes To Deal With Invalid Sender or Notification Upload Type

The system needs to know the type of workflow process to create when a notification is uploaded that does not contain a valid External System or Notification Upload Type (these two fields are the ones that control the type of workflow process that's created to process the uploaded notification). When these conditions are detected by the notification upload process, most utilities create an outgoing notification rejecting the uploaded notification when such conditions transpire. We've shown these in the following table.

Notification Condition	Workflow Process Events
Unknown Notification ID	Create notification download – reject notification, bad external system
Unknown Notification (Upload) Type	Create notification download – reject notification, bad notification upload type

These processes are not in a workflow process profile. The above workflow process templates are not referenced in a workflow process profile. Rather, when you create these workflow process templates you label them with a **Notification Condition** of **Unknown Notification ID** or **Unknown Notification Type**. The notification upload process will then create workflow process when these events transpire.

Designing Workflow Process Templates

The following table shows the workflow process templates referenced in the previous section's matrix. Adjacent to each process is its events and an indication of when they are triggered.

Workflow Process Template	Event Number	Workflow Event Type	Dependent On Event(s)	Trigger Date Set To X Calendar Days After Completion Of Dependent Events
Retrieve a customer's consumption	10	Validate notification – consumption history request	N/A – first event	0
	20	Confirm requester is a valid service provider for the customer's service	10	0
	30	Create notification download – send consumption history	20	0
Switch a customer to a different service supplier	10	Validate notification – supplier switch	N/A – first event	0
	20	Confirm requester is a valid service provider for the customer's service	10	0
	30	Check with current supplier if the switch is allowed	20	0
	40	Switch suppliers	30	0
Switch customer to an interval meter	10	Validate notification – interval meter switch	N/A – first event	0
	20	Confirm requester is a valid service provider for the customer's service	10	0
	30	Create field activity to exchange current meter	20	0
	40	Change rate on exchange date	30	0
Reject interval meter switch	10	Validate notification – interval meter switch	N/A – first event	0
	20	Create notification download – reject	10	0

		request		
Reject bad notification upload type	10	Create notification download – reject request	N/A – first event	0
Reject bad external system	10	Create notification download – reject request	N/A – first event	0

Note. The workflow process for “Reject bad notification upload type” should reference the Notification Condition **Unknown Notification Type**. The workflow process for “Reject bad external system” should reference the Notification Condition **Unknown Notification ID**.

Designing Workflow Event Types

If we extract each unique event type from the above table, we end up with the following:

Workflow Event Type
Validate notification – consumption history request
Confirm requester is a valid service provider for the customer's service
Create notification download – send consumption history
Validate notification – supplier switch
Current supplier confirmation
Switch supplier
Validate notification – interval meter switch
Create field activity to exchange meter
Change rate
Create notification download – reject request

Next, we have to determine the algorithm that will be used when each event is activated on its trigger date. We call this algorithm the *activation algorithm*. An activation algorithm is a stand-alone routine that does whatever you need done when an event is activated. We have populated the following table with brief descriptions of the types of activation algorithms you'd need for the above workflow events.

The activation algorithms are limited by your imagination (and business requirements). We have provided the activation algorithms you see below as an example; we don't expect you'll be able to use the exact algorithms that we supply. Your algorithms will be based on any number of factors. Be aware that new activation algorithms may require programming. See [How To Add A New Algorithm](#) for more information.

Workflow Event Type	Activation Algorithm
Validate notification – consumption history request	Validate consumption history request

Confirm requester is a valid service provider for the customer's service	Confirm service provider is valid requester
Create notification download – send consumption history	Create notification download – send consumption history
Validate notification – supplier switch	Validate supplier switch request
Current supplier confirmation	Create notification download – check if it's OK to switch customer from current supplier
Switch supplier	Switch supplier
Validate notification – interval meter switch	Validate interval meter switch request
Create field activity to exchange meter	Create field activity – exchange meter
Change rate	Change rate
Create notification download – reject request	Create notification download – reject request

Next, we have to determine which types of events can fail. Refer to [Some Workflow Events May Fail](#) for background information failure. For those types of events that can fail, we will indicate their *failure algorithm* in the table. A failure algorithm is a stand-alone routine that does whatever you need none when an event fails. We have populated the following table with brief descriptions of the types of failure algorithms you'd need for the above workflow events.

The failure algorithms are limited by your imagination (and business requirements). We have provided the failure algorithms you see below as an example; we don't expect you'll be able to use the exact algorithms that we supply. Your algorithms will be based on any number of factors. Be aware that new failure algorithms may require programming. See [How To Add A New Algorithm](#) for more information.

Workflow Event Type	Activation Algorithm	Failure Algorithm
Validate notification – consumption history request	Validate consumption history request	Create notification download – invalid request
Confirm requester is a valid service provider for the customer's service	Confirm service provider is valid requester	Create notification download – invalid requester
Create notification download – send consumption history	Create notification download – send consumption history	N/A
Validate notification – supplier switch	Validate supplier switch request	Create notification download – invalid request
Current supplier confirmation	Create notification download – check if it's OK to switch customer from current supplier	Create notification download – reject request due to supplier rejection
Switch supplier	Switch supplier	N/A
Validate notification – interval meter switch	Validate interval meter switch request	Create notification download – invalid request
Create field activity to exchange meter	Create field activity – exchange meter	N/A
Change rate	Change rate	N/A

Create notification download – reject request	Create notification download – reject request	N/A
---	---	-----

And finally, for those events whose activation algorithm puts them into a wait state, we have to determine the waiting process that monitors the waiting events. Refer to [Waiting Events And Their Waiting Process](#) for more information about waiting.

The waiting processes are limited by your imagination (and business requirements). We have provided the waiting processes you see below as an example; you may not be able to use the exact processes that we supply as your processes will be based on any number of factors. Be aware that a new waiting process will require programming.

Workflow Event Type	Activation Algorithm	Failure Algorithm	Waiting Process
Validate notification – consumption history request	Validate consumption history request	Create notification download – invalid request	N/A
Confirm requester is a valid service provider for the customer's service	Confirm service provider is valid requester	Create notification download – invalid requester	N/A
Create notification download – send consumption history	Create notification download – send consumption history	N/A	N/A
Validate notification – supplier switch	Validate supplier switch request	Create notification download – invalid request	N/A
Current supplier confirmation	Create notification download – check if it's OK to switch customer from current supplier	Create notification download – reject request due to supplier rejection	Check if supplier has accepted process
Switch supplier	Switch supplier	N/A	N/A
Validate notification – interval meter switch	Validate interval meter switch request	Create notification download – invalid request	N/A
Create field activity to exchange meter	Create field activity – exchange meter	N/A	Check if field activity is complete
Change rate	Change rate	N/A	N/A
Create notification download – reject request	Create notification download – reject request	N/A	N/A

Designing External Systems

As described under [What Type Of Workflow Process Is Created?](#), every notification upload staging (NUS) record contains an external system. An external system is the unique identifier of the sender of the notification.

To “design” your external systems, document the external id’s (e.g., DUNS number) of your service providers. The following table contains a sample:

External System	Description
102910	Energy Supplier, Inc.
392191	Cheap Power, Inc.

The above example would result in 2 external systems – 102910 and 392191.

Note. It’s obvious, but worth stressing, that the External System should be the exact number referenced on notifications sent by a sender.

Designing Notification Downloads

The system creates outgoing notifications when:

- It needs to respond to an incoming notification. For example, if an incoming notification requests a customer’s consumption history, the system must send the consumption history by creating an outgoing notification.
- It needs to apprise a third party that something has changed about the customer or their meter. For example, if a customer stops service, the system must tell the various service providers of such.
- It needs to exchange information with another application in the company, for example a CRM system.

A Notification Download Staging (NDS) record is created for every notification that is sent to the outside world.

The topics in this section describe how to design the tables that control your notification download processing.

Contents

[Designing Workflow Processes To Support Outgoing Notifications](#)
[Designing Your External System Feature Configuration](#)
[Define the XAI Sender](#)
[Designing Notif. Download Types](#)
[Designing Notif. Download Profiles](#)

Designing Workflow Processes To Support Outgoing Notifications

As described under [How Are Notifications Sent Out Of The System?](#) notifications are sent out via Notification Download Staging (NDS) records. NDS records are created by workflow events (i.e., the event’s activation algorithm creates notification download records). There are two types of workflow processes that contain these types of workflow events:

- Many workflow processes exist to process incoming notifications. These processes typically contain workflow events that create NDS records to communicate with service providers. If you look at the workflow processes described under [Designing Workflow Process Templates](#), you will see many such examples.

- When something noteworthy happens, the system may need to tell a third party about it. As described in [System Conditions May Trigger Notification and Workflow](#), you may decide that the noteworthy condition should cause a workflow process to be created where one or more of the workflow events can create an NDS. The following table shows representative workflow process templates of this type.

Workflow Process Template	Event Number	Workflow Event Type	Dependent On Event(s)	Trigger Date Set To X Days After Completion Of Dependent Events
Stop service	10	Create notification download – inform all parties that customer is stopping service	N/A – first event	0
Meter exchange	10	Create notification download – inform all parties that meter has been changed	N/A – first event	0

After documenting these types of workflow process templates, follow the instructions under [Designing Workflow Event Types](#) to add any new types of events to the list.

Designing Your External System Feature Configuration

For certain types of external systems, your application may define options used to configure the interaction between your product and your external system. The feature configuration table is used to define some settings applicable to the interaction with your external system, for example:

- The Options grid allows you to define information needed to communicate with the external system. For example, if you interface with an external workforce management system for booking appointments, a feature configuration for the WFM system may use an option to define whether or not the user can manually define an appointment.
- The Messages collection allows you to map each message that you may receive from the external system to a corresponding [message](#) in this system.

Recommendation. We recommend creating a new message for every message that is being mapped. This ensures that changes to future base product messages do not affect the integration.

Define the XAI Sender

If any of your messages are routed to an external system via XAI, the message must also be associated with an [XAI Sender](#), which tells the system how to send the message.

Sender Optional. If the NDS message does not require routing, you are not required to define an XAI sender.

Designing Notif. Download Types

You will need one notification download type for every type of notification your organization can send to service providers. To “design” your notification download types, list:

- Every outgoing notification that you documented under [Designing Workflow Event Types](#).
- Every outgoing notification that you documented under [Designing Workflow Processes To Support Outgoing Notifications](#).
- Every outgoing notification that may be triggered from within the system as described in [System Conditions May Trigger Notification and Workflow](#). For each of these, consider creating a new value for the download type condition flag so that the download type is not hard-coded in the system process that creates the NDS.

We have populated the Notification Download Type column with a few classic examples:

Notification Download Type	Download Type Condition Flag
Inform all parties that customer is stopping service	CUSTSTOP
Inform all parties that meter has been changed	METERCH
Send consumption history	
Check if it's OK to switch customer from current supplier	
Reject request	

You should also consider the priority of each NDS type with respect to the other types.

Once you know the different types of download notifications, you have to determine the physical method used to route the notification to the third party. Refer to [Designing Notification Download Profiles](#) for how to do this.

If the outbound notification may be routed via XAI, you must define an [XAI inbound service](#). This is used by the XAI MPL to retrieve information about the record related to the outbound message.

Refer to [Configuring the System for XAI Messages](#) for more information on designing your XAI outgoing messages.

Contents

- [Designing Context Values for the NDS Type](#)
- [Configuring the System for XAI Messages](#)
- [Designing Near Real Time NDS Messages](#)

Designing Context Values for the NDS Type

A notification download staging record must reference some "relevant data" that allows the extract process or the XAI MPL process to retrieve the data needed to send to the external system. For example, if the person's name has changed, the NDS must reference the Person Id. When formatting the data to send to an external system, the appropriate process would take the person ID and retrieve the data needed to send to the external system.

The NDS may use either the context collection or the characteristics collection to enter this "relevant data". If you choose to use context, you may decide to define the context types for the NDS type (although it is not required).

If NDS records of this type are interfaced through XAI, then the context types and corresponding XPATH values are required for every piece of data that is required to call an application service to build the extract. The XPATH indicates the relative path in the XML document where the field value will be placed when building the XML document.

Refer to [Configuring the System for XAI Messages](#) for more information on designing your XAI outbound messages.

Configuring the System for XAI Messages

Contents

- [Designing Your Real Time Messages](#)
- [Designing the Real Time Message XSL](#)
- [Designing the Real Time Message Route Types](#)
- [Designing the Real Time Message NDS Types and Download Profile](#)
- [Designing the Near Real Time Response](#)

Designing Your Real Time Messages

Consider the following integration scenario. A page in your product is configured to send information to an external system real time. If the connection is down, the user is warned and may authorize that the message is sent in near real time. The following sections describe how to design the control tables needed for both scenarios.

Designing the Real Time Message XSL

During implementation, you will determine the information required to send to the external system. The real time message engine is passed all the data available in the page model as an XML document. The request XSL must be designed to map this data into a format expected by the target.

In addition, you should create two response XSLs to transform the message received from the target system.

- One XSL is used for real time responses. The XSL transforms the message into a format understood by the page service user exit. Note that any updates to a system business object as a result of the response is the responsibility of the user exit that called the real time message engine.
- One XSL is used for non-real time responses. In this case, if the response should update a system object, the response must trigger an XUS to update the system object. Therefore, the XSL must map the response into an XML request that reference the appropriate service to update the system object.

Designing the Real Time Message Route Types

For this scenario, you must create two route types.

- Create an [XAI Route Type](#) for the real time message. Indicate the sender, the request XSL and the response XSL used for real time responses.

Note. The sender referenced on the route type should be one that supports synchronous communication (such as an HTTP sender).

- Create an XAI route type for the near real time message. Indicate the sender, the request XSL and the response XSL used for near real time responses. Mark the route type to check the flags Receive Acknowledge and Post Response. These flags indicate to XAI that a response should be received and that additional processing is required.

Designing the Real Time Message NDS Types and Download Profile

For this scenario you must create two NDS types

- One NDS type is used for the real time message. Indicate an appropriate notification download condition. This allows the mechanism that creates the NDS to refer to this condition rather than hard-coding the NDS type.
- One NDS type is used for the near real time message. This NDS type must reference a special XAI inbound service used to indicate to the download staging sender that the XDS does not need to be created.

Download Condition. The real time message engine receives the NDS type as input. It is the user exit's responsibility to determine the appropriate NDS type based on a notification download condition.

You'll need to create an entry in the service provider's [notification download profile](#) for each NDS type. Set the processing method to **XAI** and indicate the appropriate route type (real time or near real time) created above.

Designing the Near Real Time Response

To process a response to the near real time message, you must also design the XAI inbound service to be used to process the response.

- The XSL transformation scripts that are referenced by the inbound service should populate the appropriate XML elements to allow XAI to recognize that the inbound message is a response.
- Otherwise, the inbound service should be designed to process this message as appropriate. For example, if the response should update the status of the record that initiated the outgoing request, the inbound service should reference the appropriate schema to update the appropriate record.

Designing Near Real Time NDS Messages

As described above, near real time NDS messages require you to define:

- A [download staging receiver](#) to process records on the NDS table.
- A download staging [sender](#) to process "responses" to the download staging receiver.

The following sections identify other control table design issues.

Consider the following integration scenario. A change to a certain type of record in your product should trigger a message to an external system to inform that system of the change. This will be a near real time message and we expect an asynchronous response.

Contents

[Designing the XML Request and Inbound Service](#)

[Designing the Near Real Time NDS Type](#)

[Designing the Near Real Time Route Type](#)

[Designing the Response](#)

Designing the XML Request and Inbound Service

You should work with the external system to determine the information they require for this message. If a system service already exists to extract all the required information, you may use that service. However, it's possible that you would need to create a new service to extract the information you need.

Your implementers must create an appropriate service and then use the [schema editor](#) to create request and response schemas based on this service.

Message ID. If you expect an asynchronous response to this message, the request and response schemas must include a private attribute to hold the *MessageID*.

Create an XAI inbound service for extracting the appropriate information and building the XML request.

Designing the Near Real Time NDS Type

Define an appropriate NDS type. The information defined here is required to successfully build the XML request.

- Indicate the XAI Inbound Service created above.
- Indicate an appropriate notification download condition. This allows the mechanism that creates the NDS to refer to this condition rather than hard-coding the NDS type.
- Use the notification download type context to identify the data required to build the request to invoke the XAI Inbound Service for this download type. In our case, context types must be defined for the business object id and the message id. Use the [XPATH](#) to indicate the relative path for this element in the request schema.

Context Type	XPATH
Maintenance Object ID	//ExtractInfoService/ExtractInfoHeader@MaintenanceObjectID
Message ID	//ExtractInfoService/ExtractInfoHeader/MessageID

Designing the Near Real Time Route Type

You must create an [XAI Route Type](#). The route type identifies the sender for routing the message and the XSL transformation script required to transform the request into a format expected by the sender.

If the sender supports synchronous communication, you must check the Receive Acknowledge and Post Response flags to successfully process the response. Our scenario indicated that we expect an asynchronous response so we'll leave these flags unchecked.

Routing Optional. If the message does not require routing, you are not required to define an XAI route type.

Designing the Response

If you expect a response to the message sent, you must also design the XAI inbound service to be used to process the response.

- The XSL transformation scripts that are referenced by the inbound service should populate the appropriate XML elements to allow XAI to recognize that the inbound message is a response.
- Otherwise, the inbound service should be designed to process this message as appropriate. For example, if the response should update the status of the record that initiated the outgoing request, the inbound service should reference the appropriate schema to update the appropriate record.

Designing Notif. Download Profiles

A notification download profile controls how the system routes notifications to third parties. You associate a notification download profile with one or more service providers. Whenever a notification is sent to a service provider, the system uses the notification download profile to determine the interface method and the format of notifications.

A notification download profile corresponds with a protocol used to route notifications to service providers. If you electronically route all notifications using the same protocol (for example, all service providers in a given jurisdiction may conform to the same record formatting and routing method), you will have a single notification download profile. If you have to route notifications using different protocols to different providers, you will need one notification download profile per protocol.

Using the notification download profile, you can define which outgoing messages are sent as an XML document via the XAI tool and which are sent in batch format. When a message is designated as being sent via the XAI tool, then you typically define XAI routing information. If a message is designated as being sent via batch, then you must indicate the formatting algorithm used by the extract program.

XAI Routing Information Optional. You may indicate that the message should be routed via XAI, but enter no XAI Routing information. You may do this when you are interfacing with one of your own external systems and where the system may be accessed directly. In this scenario, you are using the XAI functionality to communicate between systems using XML documents, but you don't need the routing logic within XAI. Refer to [An NDS Message That Doesn't Require Routing](#) for more information.

The easiest way to design a notification download profile is to choose a representative service provider and design a notification download profile for it by filling in the following matrix. After you've designed a profile, determine how many other service providers can use it. Then design the next service provider's profile and determine who can reuse it. Repeat this process until all your service providers have a profile. Once the profiles are designed, you're ready to set up the control tables.

Background Process used for batch messages	Notification Download Type	Processing Method	XAI Route Type	Format Method
File Transfer	Inform all parties that customer is stopping service	Batch		Format Stop
	Inform all parties that meter has been changed	XAI	Meter Change	
	Send consumption history	Batch		Format Consumption
	Check if it's OK to switch customer from current supplier	XAI	Confirm Switch	
	Reject request	XAI	Reject	

Notice that you define the background process at the profile level, but you indicate for each download type whether it will be processed via batch or via XAI. If at least one of the processing methods is batch, then a background process must be entered, otherwise it's not applicable.

Also note that it is possible to link more than one XAI route type to each notification download type formatting method. However, this is not recommended if you expect an [asynchronous response to the NDS message](#).

After you've designed notification download profiles to cover every protocol, you are ready to set up the control tables.

Refer to [Notification Download Background Processes](#).

Setting Up Notification and Workflow Procedures

In the previous sections, [Designing Notification Upload & Workflow Procedures](#) and [Designing Notification Downloads](#), we presented a case study that illustrated a mythical organization's workflow procedures. In this section, we'll explain how to set up the control tables to implement these procedures.

Contents

- [Setting Up Workflow Event Types](#)
- [Setting Up Workflow Process Templates](#)
- [Setting Up Notification Upload Types](#)
- [Setting Up External Systems](#)
- [Setting Up Workflow Process Profiles](#)
- [Setting Up Notif. Download Types](#)
- [XAI Route Type](#)
- [Setting Up Notif. Download Profiles](#)

Setting Up Workflow Event Types

Workflow event types control what is done by a given workflow event. Open **Admin Menu**, **Workflow Event Type** to define your workflow event types.

Refer to [Designing Workflow Event Types](#) for more information.

Description of Page

Enter a unique **Workflow Event Type** and **Description** for the event type.

Turn on **Allow Manual Completion** if an operator is allowed to change the status of workflow events of this type to **Complete**.

The **Event Activation Algorithm** is used by the system when it activates a workflow event of this type. Refer to [Executing Workflow Events On Their Trigger Date](#) and [Designing Workflow Event Types](#) for more information. If you haven't done so already, you must set up this algorithm in the system. To do this:

- Create a new algorithm (refer to [Setting Up Algorithms](#)).
- On this algorithm, reference an Algorithm Type that is associated with workflow event activation. The system comes supplied with several sample algorithm types that should be used as a sample if you have to write a new algorithm type. Click [here](#) to see the algorithm types available for this system event.

The **Event Failure Algorithm** is used by the system when a workflow event of this type fails. This algorithm need only be defined if events of this type can fail. Refer to [Some Workflow Events May Fail](#) and [Designing Workflow Event Types](#) for more information. If you haven't done so already, you must set up this algorithm in the system. To do this:

- Create a new algorithm (refer to [Setting Up Algorithms](#)).
- On this algorithm, reference an Algorithm Type that is associated with workflow event failure. The system comes supplied with several sample algorithm types that should be used as a sample if you have to write a new algorithm type. Click [here](#) to see the algorithm types available for this system event.

The **Wait Process** is the background process responsible for monitoring workflow events of this type that are in the **Waiting** state. Refer to [Waiting Events And Their Waiting Process](#) for more information.

Where Used

Follow this link to view the tables that reference [CI_WF_EVT_TYPE](#) in the data dictionary schema viewer.

Setting Up Workflow Process Templates

A workflow process template defines the workflow events that will be created when a workflow process is created using a template.

Refer to [Designing Workflow Process Templates](#) for more information.

Contents

- [Workflow Process Template - Main](#)
- [Workflow Process Template - Template Tree](#)

Workflow Process Template - Main

Open **Admin Menu, Workflow Process Template** to define your workflow process templates.

Description of Page

Enter a unique **Workflow Process Template** code and **Description** for the workflow process template.

The system needs to know the type of workflow process to create when a notification is uploaded that does not contain a valid External System or Notification Upload Type (these two fields are the ones that control the type of workflow process that's created to process the uploaded notification). If you create workflow process templates and label them with a **Notification Condition** of **Unknown Notification ID** or **Unknown Notification Type**, the notification upload process will create a respective workflow process when either of the above conditions are discovered. Note: most utilities create an outgoing notification rejecting the uploaded notification when such conditions transpire. The notification condition field is also available for use by your plug-in algorithms for any purpose where you need to identify a specific workflow process.

Note. The values for this field are customizable using the Lookup table. This field name is WF_PROC_COND_FLG.

Use **Comments** to describe the workflow process template.

When a workflow process is created, the system links one or more workflow events to it. The information in the **Workflow Responses** scroll defines these events and when they will be triggered. The following fields are required for each event:

Event Sequence	Sequence controls the order in which the workflow events are executed. The sequence number is system-assigned and cannot be changed. If you have to insert a workflow event between two existing events, you'll have to remove the latter events, insert the new event, and then re-specify the removed events.
Workflow Event Type	Specify the type of workflow event to be generated. The event type's description is displayed adjacent.
Dependent on Other Events	Turn this indicator on if the trigger date of the event can only be determined after earlier events are complete. Refer to Workflow Event Dependencies & Trigger Date for more information. If this switch is on, you must define the events on which this response depends in the Dependent on Other Events grid.
Days After Previous Response	Specify the number of calendar days after <u>the completion of the dependent events</u> on which the workflow event will be triggered. If this event is not dependent on the completion of other events, this field contains the number of calendar days after the <u>creation of the workflow process</u> that the related workflow event will be triggered.

When the **Dependent on Other Events** switch is on, a grid appears in which you specify the events on which this event is dependent. The following fields are required for each event:

Sequence	Sequence is system-assigned and cannot be specified or changed.
-----------------	---

Dependent on Sequence	Specify the sequence number of the workflow event type on which the above workflow event depends.
Workflow Event Type	The system displays the ID of dependent workflow event in this column.

For more information about workflow event templates, see [Setting Up Workflow Event Types](#).
For more information about trigger dates, see [Workflow Event Dependencies & Trigger Date](#).

Where Used

A Workflow Process Profile references one or more workflow process templates. Refer to [Setting Up Workflow Process Profiles](#) for more information.

A Workflow Process references a workflow process template. Refer to [Workflow Process – Main](#) for more information.

Workflow Process Template - Template Tree

Open **Admin Menu**, **Workflow Process Template** and navigate to the **Template Tree** tab to view information about your workflow process template.

Description of Page

This page is dedicated to a [tree](#) that shows the events linked to the workflow process and information about the event dependencies. You can use this tree to both view high-level information about these objects and to transfer to the respective page in which an object is maintained.

Setting Up Notification Upload Types

Every notification upload staging record has a notification upload type. This code is one of several fields that control the type of workflow process used to process the incoming notification. Open **Admin Menu**, **Notification Upload Type** to define your notification upload types.

Refer to [Designing Notification Upload Types](#) for more information.

Description of Page

Enter a recognizable **Notification Upload Type** and **Description**.

Enter a value for the **Upload Condition Flag** when a system condition should trigger the creation of a notification record. Refer to [System Conditions May Trigger Notification and Workflow](#) for more information.

Note. The values for this field are customizable using the Lookup table. This field name is NT_UP_TY_COND_FLG.

If NUS records of this type will be associated with an **Extension** record, indicate the extension here along with its **Navigation Option** to allow a user to navigate to the correct extension record when viewing the notification upload staging record. Refer to [Navigating to Related NUS Extension](#) for more information.

Where Used

A [Notification Upload Staging](#) record must reference a notification upload type.

A [Workflow Process Profile](#) references one or more notification upload types.

Setting Up External Systems

Every notification upload staging record references the system that sent the message. The external system is one of several fields that control the type of workflow process used to process the incoming notification. Refer to [External Systems](#) for more information.

Setting Up Workflow Process Profiles

The system uses a workflow process profile to determine the type of workflow process to create for incoming notifications sent by the service provider. A workflow process profile is associated with one or more service providers. Open **Admin Menu, Workflow Process Profile** to define your workflow process profiles.

Refer to [Designing Workflow Process Profiles](#) for more information.

Description of Page

Define a unique ID and **Description** for each workflow **Process Profile**.

The information in the grid defines the type of workflow process that will be created for each **Notification Upload Type**. The type of workflow process may differ depending on special criteria. For example, you may have a different workflow process if the customer is industrial (as compared to commercial and residential). You must define **Default** criteria in case none of the override criteria are met (the **Default** criteria should have the lowest priority). You MAY have override criteria if different situations result in different types of workflow processes.

The following fields are required for each criterion:

Priority	The priority controls the order in which the system determines if the respective workflow process should be used to process notifications of a given type. Higher priorities are checked before lower priorities.
-----------------	---

Note. The values for this field are customizable using the Lookup table.

Criteria Algorithm	Select the algorithm to be used to check if the workflow process should be initiated for notifications of a given type. If a condition is met, a workflow process is created using the associated workflow process template.
---------------------------	--

If you haven't done so already, you must set up this algorithm in the system. To do this:

- Create a new algorithm (refer to [Setting Up Algorithms](#)).

- On this algorithm, reference an Algorithm Type that determines if an incoming notification should be processed using the associated **Workflow Process Template**. The system comes supplied with a sample algorithm type called that should be used as a sample if you have to write a new algorithm type. Click [here](#) to see the algorithm types available for this system event.

Important! You must have at least one entry in this collection otherwise the system will not start a workflow process when an incoming notification of this type is received. This entry should have the lowest priority code and should reference a **Criteria Algorithm** that references the default workflow criteria algorithm type.

Workflow Process Template Specify the workflow process template to use to process incoming notifications identified with the notification upload type.

Refer to [Designing Workflow Process Criteria](#) for more information.

Setting Up Notif. Download Types

Every notification download staging record has a notification download type. This code controls the format of the record that is sent to a service provider. Open **Admin Menu, Notification Download Type** to define your notification download types.

Refer to [Designing Notification Download Types](#) for more information.

Contents

- [Notification Download Type - Main](#)
- [Notification Download Type - Context](#)

Notification Download Type - Main

Every notification download staging record has a notification download type. This code controls the format of the record that is sent to a service provider. Open **Admin Menu, Notification Download Type** to define your notification download types.

Description of Page

Enter a unique **Notification Download Type** and **Description**.

Specify the **XAI In Service Name** that will be called for this NDS type if the download profile formatting method indicates that this download type will be processed by XAI.

Enter a value for the **Download Type Condition Flag** when a system condition should trigger the creation of a notification record. Refer to [System Conditions May Trigger Notification and Workflow](#) for more information.

Note. The values for this field are customizable using the Lookup table. This field name is NT_DWN_TY_COND_FLG.

Indicate the relative **Priority** for processing NDS records of this type with respect to other types. This value defaults to **Priority 90 – Lowest**.

Note. The values for this field are customizable using the Lookup table. This field name is NT_DWN_TY_PRIO_FLG.

Notification Download Type - Context

If the notification download staging record is communicated to the external system through XAI, the NDS context is used to help build the XML document.

Description of Page

The context collection functionality allows you to define a collection of **Context Types** that should be defined for a notification download staging record of this type. When the NDS record is created, with its collection of these Context Types, the Context Values would correspond to system data related to this NDS record. In addition, you may specify a **Context Value** if this is a constant value for NDS records of this type.

The **XPATH** is used by XAI NDS Types. For each context type, use the XPATH to indicate the relative path in the XML document where the field value will be placed when building the XML document.

Where Used

A Notification Download Staging record must reference a notification download type. Refer to [Process X – Populate Notification Upload Staging](#) for more information.

A Notification Download Profile references one or more notification download types. Refer to [Setting Up Notification Download Profiles](#) for more information.

XAI Route Type

Use this page to define information required by the XAI tool to send outgoing messages. Navigate to this page using **Admin Menu, XAI Route Type**.

Description of Page

Enter the **XAI Route Type**, which defines the information required for outgoing messages, which use the XAI tool. Enter a **Description** for this route type.

The **XSL Request** is the schema used to transform information from the format produced by the system to a format understood by the sender, who receives a message of this type.

The **XSL Response** is the schema used to transform information from the format sent to us by the sender, who responds to this message into a format understood by the system.

Use the **XAI Sender** to indicate where messages of this type should be sent.

Refer to [XAI Sender](#) for more information.

Check the **Receive Acknowledge** box if the system expects to receive a synchronous response to outgoing messages of this type.

Check the **Post Response** box if a synchronous response to an outgoing message requires something to occur in the system. If the box is checked, a response to a message of this type causes an [XAI upload staging](#) record to be created. That record is processed along with other uploaded messages, to invoke an appropriate service.

Where Used

All outgoing messages are sent through the notification download staging record. Information related to the formatting of messages is defined on a notification download profile.

Setting Up Notif. Download Profiles

A notification download profile controls how the system routes notifications to service providers. You associate a notification download profile with one or more service providers. Whenever a notification is sent to a service provider, the system uses the notification download profile to determine the interface method and the format of notifications.

Open **Admin Menu, Notification Download Profile** to define your notification download profiles.

Refer to [Designing Notification Download Profiles](#) for more information.

Description of Page

Define a unique ID and **Description** for each **Download Profile**.

Use **NDS Extract Process** to define the background process that creates notification download records and interfaces them out of the system.

The information in the scroll controls how the **Extract Process** formats an interface record for each **Notification Download Type**. In addition to defining a **Description** and **Comments**, you must define the **Processing Method**. The valid values are **XAI** and **Batch**.

If your processing method is **Batch**, you need to define the **Notification Format Algorithm** that actually formats the interface record. If you haven't done so already, you must set up this algorithm in the system. To do this:

- Create a new algorithm (refer to [Setting Up Algorithms](#)).
- On this algorithm, reference an Algorithm Type that is associated with workflow event failure. The system comes supplied with several sample algorithm types that should be used as a sample if you have to write a new algorithm type. Click [here](#) to see the algorithm types available for this system event.

If your processing method is **XAI**, you may define one or more **XAI Route Types**. The [XAI Route Type](#) describes how the message should be formatted and the destination of the message.

Defining Umbrella Agreement Options

Your organization may use umbrella agreement functionality to manage many situations, including (but not limited to) the following:

- Creating a "negotiated contract" with a large customer with many sites.
- Creating a "negotiated contract" with all customers in a common geographic area, for example a specific city or an apartment complex
- Grouping and managing the proposal service agreements created during the quoting process
- Managing the renewal process for special contracts
- Applying override rate terms to one or more service agreements for a given effective period
- more...

Separately module. Please note that the umbrella agreement functionality is part of the **Contract Management** module. If this module is not applicable to your business you may turn it off. Refer to [Turn Off A Function Module](#) for more information.

Contents

[The Big Picture Of Umbrella Agreements](#)
[Setting Up Umbrella Agreement Options](#)
[Umbrella Agreements Configuration Examples](#)

The Big Picture Of Umbrella Agreements

The topics in this section provide background information about how to configure the system to support your umbrella agreement requirements.

Contents

[Renewable Umbrella Agreements](#)
[Terms Of Service Covered Entities](#)
[Overriding Rate Terms](#)

Renewable Umbrella Agreements

When defining your umbrella agreement types, you must determine if umbrella agreements of this type are renewable. If so, you must set the renewal flag on the UA type to **Renewable** and determine the number of days prior to the end date of the umbrella agreement that the renewal process should start.

In addition, you must decide on your renewal procedure and design an appropriate renewal [algorithm](#). You may use one of the sample renewal algorithms provided by the system.

- The sample algorithm [URNW-TD](#) creates a To Do entry. Using this algorithm you can direct the To Do entry to an appropriate role to determine if the umbrella agreement should be renewed. If the umbrella agreement is associated with an account management group, the To Do entry could be assigned to a role defined for that AMG.

- The sample algorithm [URNW-CA](#) creates a [case](#). Using this algorithm you can create an appropriate case to help track the steps required for renewing an umbrella agreement.

If the base algorithms do not provide the logic your organization needs, you may create a new algorithm.

Note. It is the responsibility of the renewal algorithm to change the renewal date as per the business requirements. If renewal requires manual review, the algorithm should reset the date to blank. If the algorithm can determine a new renewal date, the date should be changed to the appropriate value.

Terms Of Service Covered Entities

For each terms of service type you design, consider whether or not a TOS created for this type should reference one or more entities in the system that are “covered” by the terms of service. Following are some examples of covered entities:

- A special contract is created for all the McDonalds franchises in your service area. When creating the terms of service record for the umbrella agreement used for this contract, you want the user to link the person record representing the McDonalds corporate entity as a covered entity for the TOS.
- An umbrella agreement is created for a large customer with several types of services. Perhaps you want to group the services in the contract based on type of service. The service type may be set up as a covered entity.

The covered entities could also be used to denote eligibility. Refer to [Linking To A UA For A Group Of Premises](#) for an example of using the covered entity collection for eligibility.

The covered entities are actually characteristics. You must define the desired characteristic types if they do not already exist. In this case, they would be foreign key characteristic types. The characteristic types that you want to use must reference a characteristic entity of **TOS covered entity**.

A terms of service record may only reference covered entity types that you designate on the TOS type record. When designing the covered entities that are allowed for TOS records of a given type, you may also configure the TOS type to indicate that one or more covered entity types are required.

Refer to [Terms Of Service Type - Main](#) for more information.

Overriding Rate Terms

If your organization has business rules that require a service agreement's rate terms to be overridden, you can accomplish this using umbrella agreements and terms of service records. There are two levels of overriding that are possible:

- Override rate schedule
- Override other terms on the rate, including contract riders, contract terms and tax exemptions

To override any rate terms, the service agreement must be linked to a terms of service record that references a template service agreement. In addition, you must configure settings in the system to indicate whether or not overriding of terms is allowed.

- The [SA type](#) for the customer's service agreement indicates whether or not the rate schedule may be overridden using the rate source flag.
 - If the rate source value is **Check TOS First, then SA**, when applying the rate, the system determines if the SA is linked to a TOS effective at the time of the bill with a template SA. If so, the rate schedule on the template SA is used.
 - If the rate source value is **Check SA Only**, when applying the rate, the always uses the rate schedule on the customer's SA (regardless of whether or not the SA is linked to a TOS effective at the time of the bill with a template SA).
- If any bill factor on one of the rate components indicates that it is eligible for tax exemptions, contract terms or contract riders, the terms of service usage flag on the bill factor indicates where the system should look for the appropriate value. The flag tells the system to find applicable information either on the TOS's template SA only, on the customer's SA only or it should first check the TOS, then check the customer's SA.

The following table illustrates the different scenarios possible when configuring your bill factors and the resulting behavior. The term "standard behavior" in the tables below indicate that the standard behavior as described in [Defining Bill Factors](#) is followed.

	TOS Usage	Template SA	Customer SA	BF Char Source
Contract Rider Applicability is checked	Check TOS First, then SA	Found	Not checked	If source = SA, the template SA's characteristic collection is checked. If source <> SA, the standard behavior is followed.
		Not Found	Found	If source = SA, the customer SA's characteristic collection is checked. If source <> SA, the standard behavior is followed
		Not Found	Not Found	N/A
	Check TOS Only	Found	Not checked	If source = SA, the template SA's characteristic collection is checked. If source <> SA, the standard behavior is followed
		Not Found	Not checked	N/A
	Check SA Only	Not checked	Standard behavior	

	TOS Usage	Template SA	Customer SA	BF Value Source	BF Char Source
	Check TOS First,	Found	Not checked	Template SA	N/A
		Not Found	Found	Customer SA	N/A

Value in Contract Term is checked	<i>then SA</i>	Not Found	Not Found	Bill Factor Value (based on Char Source)	If source = SA, the template SA's characteristic collection is checked. If source <> SA, the standard behavior is followed
	<i>Check TOS Only</i>	Found	Not checked	Template SA	N/A
		Not Found	Not checked	Bill Factor Value (based on Char Source)	If source = SA, the template SA's characteristic collection is checked. If source <> SA, the standard behavior is followed
	<i>Check SA Only</i>	Not checked	Standard behavior		

	TOS Usage	Template SA	Customer SA	BF Value Source
Tax Exemption is checked	<i>Check TOS First, then SA</i>	Found	Not checked	Template SA
		Not Found	Found	Customer SA
		Not Found	Not Found	No value
	<i>Check TOS Only</i>	Found	Not checked	Template SA
		Not Found	Not checked	No value
	<i>Check SA Only</i>	Not checked	Standard behavior	

Note that [proration](#) logic is followed for the above functionality as expected. For example, imagine that a bill factor with Value in Contract Terms is prorable and the TOS usage value is **Check TOS First, then SA**. If the template SA has a contract value that is only in effect for the first 10 days of a bill, that value is used for the first 10 days and the remaining days in the bill take the value from the SA (if applicable) or the bill factor value as per the table rules above.

Note. When designing your terms of service type, you must indicate whether a template service agreement is **optional**, **required** or **not allowed**. When the template SA is required, the user setting up the TOS record for this TOS type must link the correct service agreement as the template SA. If a new template service agreement is required, the new SA must be created first.

Setting Up Umbrella Agreement Options

The topics in this section describe how to set up umbrella agreement options.

Contents

- Configure SA Types for Umbrella Agreements
- Configure Bill Factors for Umbrella Agreements
- Configure A Campaign To Link An SA To A Terms Of Service

[Setting Up Umbrella Agreement Types](#)
[Setting Up Terms Of Service Types](#)
[Setting Up TOS Cancel Reasons](#)

Configure SA Types for Umbrella Agreements

For each type of service agreement that may be linked to a terms of service record, you must configure the Rate Source for the [SA type](#) to indicate if the rate should always be taken from the service agreement (**Check SA Only**) or if the rate should be taken from the terms of service record for the service agreement, if any (**Check TOS First, then SA**).

Refer to [Overriding Rate Terms](#) for more information.

Note. The rate source is only visible on SA type if the **Contract Management** module is not [turned off](#).

Configure Bill Factors for Umbrella Agreements

For each bill factor that has a value in contract term, contract rider applicability or tax exemption applicability, you must indicate where the system should look for a value when a service agreement is linked to a terms of service with a template SA and where this bill factor is referenced by the rate being applied. Using the terms of service usage flag, tell the system to **Check TOS Only**, **Check SA Only** or **Check TOS First, then SA**.

Refer to [Overriding Rate Terms](#) for more information.

Note. The terms of service usage flag is only visible on bill factor if the **Contract Management** module is not [turned off](#).

Configure A Campaign To Link An SA To A Terms Of Service

The system provides a pair of column reference algorithm types that support linking a new service agreement to an existing terms of service record via an [order](#). The pair includes a validation algorithm type [CRVL-SATOS](#) and a posting algorithm type [CRPS-SATOS](#).

In order to take advantage of this functionality, you must:

- Create an algorithm for each of the above algorithm types. Only one algorithm is needed for each algorithm type because there are no parameters for either one.
- Setup a unique [column reference](#) for the terms of service ID column reference code and plug in the new validation and posting algorithms on the record.
- For campaigns whose orders link a new service agreement to an existing umbrella agreement, make sure to define a [miscellaneous field](#) for the terms of service ID. Note, the miscellaneous field must reference the column reference that you setup above.

Based on your business practice, you may want to further configure your campaign / package for linking a service agreement to an existing terms of service. Refer to [Linking A Service Agreement To A TOS Through Orders](#) for more information.

Setting Up Umbrella Agreement Types

The umbrella agreement type transaction is used to maintain your UA types. The topics in this section describe how to use this transaction.

Contents

[Umbrella Agreement Type - Main](#)

[Umbrella Agreement Type - Algorithms](#)

Umbrella Agreement Type - Main

Open **Admin Menu, Umbrella Agreement Type** to define basic information about an umbrella agreement type.

Description of Page

Enter a unique **Umbrella Agreement Type** code and **Description** for the UA type.

Use **Renewal** to indicate whether an umbrella agreement of this type is **Renewable** or **Not Renewable**. For renewable UA types, indicate the **Renewal Days Before Expiration**. The renewal date for umbrella agreements of this type are calculated based on this number of days before the end date.

Use the characteristics collection to define characteristics that can be defined for umbrella agreements of a given type. Use **Sequence** to control the order in which characteristics are defaulted. Turn on the **Required** switch if the **Characteristic Type** must be defined on umbrella agreements of a given type. Turn on the **Default** switch to default the **Characteristic Type** when umbrella agreements of the given type are created. Enter a **Characteristic Value** to use as the default for a given **Characteristic Type** when the **Default** switch is turned on.

Characteristic Types. You can only choose characteristic types defined as permissible on an umbrella agreement record. Refer to [Setting Up Characteristic Types & Their Values](#) for more information.

Enter the valid **Terms of Service Types** that may be referenced for umbrella agreements of this type.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference [CI_UA_TYPE](#).

Umbrella Agreement Type - Algorithms

Open **Admin Menu, Umbrella Agreement Type** and then navigate to the **Algorithms** tab.

Description of Page

The grid contains **Algorithms** that control important functions in the system. You must define the following for each algorithm:

- Specify the **System Event** with which the algorithm is associated (descriptions of all possible events are provided below).
- Specify the **Sequence** and **Algorithm** for each system event. You can set the **Sequence** to 10 unless you have a **System Event** that has multiple **Algorithms**. In this case, you need to tell the system the **Sequence** in which they should execute.

The following table describes each **System Event** for which you can define algorithms.

System Event	Description
<i>UA Renewal</i>	Algorithms of this type are executed by the UARENEW background process when an umbrella agreement has reached its renewal date. Only one renewal algorithm is allowed for a given UA type. Click here to see the algorithm types available for this system event.

Setting Up Terms Of Service Types

The terms of service type transaction is used to maintain your TOS types. The topics in this section describe how to use this transaction.

Contents

- [Terms Of Service Type - Main](#)
- [Terms Of Service Type - UA Types](#)

Terms Of Service Type - Main

Open **Admin Menu, Terms of Service Type** to define basic information about terms of service types.

Description of Page

Enter a unique **Terms of Service Type** code and **Description** for the TOS type.

Use **Template SA Usage** to indicate whether a template SA is **Optional**, **Required** or **Not Allowed**. Refer to [Overriding Rate Terms](#) for more information about using template SAs.

Use the characteristics collection to define characteristics that can be defined for terms of service records of a given type. Use **Sequence** to control the order in which characteristics are defaulted. Turn on the **Required** switch if the **Characteristic Type** must be defined on terms of service records of a given type. Turn on the **Default** switch to default the **Characteristic Type** when terms of service records of the given type are created. Enter a **Characteristic Value** to use as the default for a given **Characteristic Type** when the **Default** switch is turned on.

Characteristic Types. You can only choose characteristic types defined as permissible on a terms of service record. Refer to [Setting Up Characteristic Types & Their Values](#) for more information.

Use the **Covered Entities** collection to define the types of entities that are “covered” by terms of service records of this type. Refer to [Terms Of Service Covered Entities](#) for more information

Characteristic Types. You can only choose characteristic types defined as permissible for TOS covered entity. Refer to [Setting Up Characteristic Types & Their Values](#) for more information.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference [CI_TOS_TYPE](#).

Terms Of Service Type - UA Types

Open **Admin Menu**, **Terms of Service Type** and then navigate to the **UA Types** tab.

Description of Page

Enter the valid **Umbrella Agreement Types** that may reference this terms of service type.

Setting Up TOS Cancel Reasons

Whenever a terms of service record is canceled, a cancel reason must be specified. Open **Admin Menu**, **TOS Cancel Reason** to specify such cancellation reasons.

Description of Page

Enter an easily recognizable **TOS Cancel Reason** and **Description** for the terms of service cancellation reason.

Umbrella Agreements Configuration Examples

This section provides examples of how you would configure the system to support various business scenarios.

Linking A Service Agreement To A TOS Through Orders

As described in [Configure A Campaign To Link An SA To A Terms Of Service](#), the system has provided a pair of column reference algorithms to enable your users to link a new service agreement to an existing terms of service record (and its umbrella agreement) when starting service via the order transaction.

When designing your campaign and packages to use this option, consider the scenarios where you may use this to determine whether or not you require additional configuration. The following topics describe some possible scenarios.

Contents

- [Linking To A UA For A Given Account](#)
- [Linking To A UA For A Group Of Premises](#)

Linking To A UA For A Given Account

Imagine that you would use this logic when an important customer is starting additional service and has an existing umbrella agreement for other services that the new service should be added to. The user creating the order is probably the account manager for that account and would know the correct terms of service record to link the new service to.

For this scenario, you do not need to provide any "help" to the user to indicate that linking to a TOS is an option or to help select the correct TOS. You simply need to ensure that the campaign that would be used by this account manager to start service for the important customer simply includes a question / miscellaneous field to [link the SA to the terms of service](#).

Linking To A UA For A Group Of Premises

Imagine you have created an umbrella agreement for a group of premises that are not actively managed by a specific user. For example, perhaps there are several apartment complexes that have negotiated a special rate with your company. When new customers sign up for service at a premise that belongs to one of these apartments, the user should be informed that there is a special contract for this premise. The user may also need help selecting the correct terms of service record. To accomplish this logic, you must define additional algorithms to help your users.

For example, perhaps your standard campaign for residential electric customers includes a package to be used by premises linked to one of these apartment buildings and should only be eligible to one of those premises. You must create an eligibility algorithm for this package. Its logic could be designed based on the following assumptions:

- The qualifying premises are linked to a parent premise
- The parent premise is linked to the appropriate terms of service record in its covered entity collection

The eligibility algorithm checks to see if the order's premise is linked to a parent premise that is referenced on a terms of service record for an umbrella agreement that is currently in effect. It could also be restricted to search for a specific umbrella agreement type or terms of service type if appropriate.

Note. The campaign for this scenario must include a question / miscellaneous field to [link the SA to the terms of service](#). It should be configured with an applicability of ***only applicable on package***. The package that is used to include the SA into the terms of service for the apartment complex would include this question / miscellaneous field.

Once the user chooses this package, user is prompted to provide the correct terms of service record via the question / miscellaneous field. To help the user, you should create a column reference retrieval algorithm to default the correct terms of service record for this premise. It would use the same logic as the eligibility algorithm described above. In other words, it would find the parent premise for this premise and then find the terms of service record covered entity collection includes this parent premise and whose umbrella agreement is currently in effect.

Reports Addendum

This chapter is an addendum to the general [Defining and Designing Reports](#) chapter. This addendum describes the sample reports that are provided with Oracle Utilities Customer Care and Billing.

Description of Sample Reports

This section provides an overview of each sample report supplied with Oracle Utilities Customer Care and Billing that may be found in the demonstration database. They may be used by your organization as they are or as a starting point for creating a [new report](#).

Account Security. Please note that the sample reports provided with the product do NOT incorporate account security. If a user has been given security to view the report, then all the data in the report is available for viewing.

Contents

- [Active Severance Processes - CI_ACSVPR](#)
- [Bill Print in BI Publisher and Crystal - CI_BILLPR](#)
- [Billed Revenues by Rate - CI_BILREV](#)
- [Case Statistics By Case Type - CI_CSESTS](#)
- [Case Statistics for a Given Status - CI_CSESGS](#)
- [Collection Summary - CI_CLLSUM](#)
- [Customer Contacts by Type - CI_CUSTCN](#)
- [Customers with Life-Support / Sensitive-Load - CI_PMLSSL](#)
- [Field Order Print in BI Publisher and Crystal - CI_FOPRNT](#)
- [GL Accounting Summary - CI_GLACSM](#)
- [Letter Print - BI Publisher and Crystal Sample Welcome Letters - CI_LTRGN_ENG REPORT](#)
- [Meter Reads Performance - CI_MTREAD](#)
- [Open Cases By Type - CI_CSEOPN](#)
- [Payments Balance - CI_PMTBAL](#)
- [Receivables Aging - CI_RCVAGA](#)
- [Tax Payables Analysis - CI_TXPYBL](#)
- [To Do Entries - CI_TDENTR](#)
- [Umbrella Agreement Summary - CI_UASUMM](#)
- [Unbilled Revenues - CI_UBLREV](#)
- [Vacant Premises with Consumption - CI_VACANT](#)

Active Severance Processes - CI_ACSVPR

Parameters

Parameter	Parameter Code	Description
Start Date	P_FROM_DT	Start date to use for reporting severance processes. If not defined, start date is set to 1900-01-01.
End Date	P_TO_DT	End date to use for reporting active severance processes. If not defined to date is set to the current date.

Report Description

This report is used to aid in the monitoring of active severance processes. The report details are used to review how the company is doing as far as the collection of outstanding debt, as well as to monitor the progress of currently active severance processes.

The report selects active severance processes whose creation date is between the input start and end dates.

Bill Print in BI Publisher and Crystal - CI_BILLPR

Note. Bill Print in batch for BI Publisher is not supported yet.

Parameters

Parameter	Parameter Code	Description
CC&B user	P_USER_ID	No default value. Required. Returns an error message if user ID is null, empty or does not exist in database.
Batch Switch	P_BATCH_SW	For a single bill print, Batch Switch is 'No' and for multiple bill print, Batch Switch is 'Yes'.
First Bill	P_FROM_BILL_ID	First Bill Id to print for the Bill Range.
Last Bill	P_TO_BILL_ID	Last Bill Id to print for the Bill Range.
Batch Code	P_BATCH_CD	Passed in by the system the report is submitted in batch. Only bills with a Bill Routing that references this batch code are selected.
Batch Number	P_BATCH_NBR	Passed in by the system the report is submitted in batch. Only bills with a Bill Routing that references this batch number are selected.
Extract Algorithm	P_EXTRACT_ALG	Passed in by the system the report is submitted in batch. Only bills with a Bill Routing whose bill route type references this extract algorithm are selected.

Report Description

This report is used with BI Publisher and Crystal Reports to display all the information that appears on the customer's printed bill for the bill ID or range of bills entered in the input parameter. For a report submitted in batch, the parameters batch code, batch number and extract algorithm are also used to limit the bills selected. Refer to the description of the parameters above. The bill is only displayed if the bill is complete.

See below for details of the information extracted for the report. In addition, the sample report layout provided for the base product includes:

- Payment coupon.
- Form for change of address.
- A graph of consumption for current charges.

The following sections highlight the data that is extracted for this report.

Contents

[Bill Information](#)
[Bill Segment Information](#)
[Account Summary Information](#)
[Current Charge Information](#)
[Premise Information](#)
[Auto Pay Information](#)
[Bill Segment Readings](#)
[Bill Segment Consumption History](#)
[Payment Information](#)
[Adjustment Information](#)
[Bill Message Information](#)

Bill Information

Field Name	Format	Source/Value/Description
SORT_KEY	A12	Primary sort for the report - currently sorted by Bill Id
BILL_ID	A12	CI_BILL
SEQNO	N5	CI_BILL_ROUTING
PER_ID	A10	CI_BILL_ROUTING
NBR_BILL_COPIES	N1	CI_BILL_ROUTING
ENTITY_NAME1	A64	CI_BILL_ROUTING. Mailing Entity Name 1.
ENTITY_NAME2	A64	CI_BILL_ROUTING. Mailing Entity Name 2.
ENTITY_NAME3	A64	CI_BILL_ROUTING. Mailing Entity Name 3.
COUNTRY	A3	CI_BILL_ROUTING
ADDRESS1	A64	CI_BILL_ROUTING
ADDRESS2	A64	CI_BILL_ROUTING
ADDRESS3	A64	CI_BILL_ROUTING
CITY	A30	CI_BILL_ROUTING
COUNTY	A30	CI_BILL_ROUTING
STATE	A6	CI_BILL_ROUTING
POSTAL	A12	CI_BILL_ROUTING
IN_CITY_LIMIT	A1	CI_BILL_ROUTING
BATCH_CD	A8	CI_BILL_ROUTING
BATCH_NBR	N10	CI_BILL_ROUTING
NO_BATCH_PRT_SW	A1	CI_BILL_ROUTING
ACCT_ID	A10	CI_BILL
BILL_STAT_FLG	A2	CI_BILL
BILL_DT	Date	CI_BILL
DUE_DT	Date	CI_BILL
ACCT_ENTITY_NAME	A64	Get Main Person from CI_PER_NAME. Account Entity Name
LANGUAGE_CD	A3	Person's Language from CI_PER

CURRENCY_CD	A3	CI_ACCT
CUR_SYMBOL	A4	CI_CURRENCY_CD
DECIMAL_POSITIONS	N1	CI_CURRENCY_CD
CUR_POS_FLG	A2	CI_CURRENCY_CD
OPEN_ITEM_SW	A1	CI_CUST_CL
BILL_COPY	A1	Bill copy number

Bill Segment Information

Field Name	Format	Source/Value/Description
BSEG_ID	A12	CI_BSEG
SA_ID	A10	CI_BSEG
START_DT	Date	CI_BSEG
END_DT	Date	CI_BSEG
BILL_CYC_CD	A4	CI_BSEG
WIN_START_DT	Date	CI_BSEG
EST_SW	A1	CI_BSEG
CLOSING_BSEG_SW	A1	CI_BSEG
PREM_ID	A10	CI_BSEG
CIS_DIVISION	A5	CI_SA
SA_TYPE_CD	A8	CI_SA
SAT_DESCR	A60	CI_SA_TYPE_L
BILL_PRT_PRIO_FLG	A2	CI_SA_TYPE
GRAPH_UOM_CD	A4	CI_SA_TYPE
HEADER_SEQ	N3	CI_BSEG_CALC
BCALC_START_DT	Date	CI_BSEG_CALC
BCALC_END_DT	Date	CI_BSEG_CALC
RS_CD	A8	CI_BSEG_CALC
EFFDT	Date	CI_BSEG_CALC
BILLABLE_CHG_ID	A12	CI_BSEG_CALC
CALC_AMT	N15.2	CI_BSEG_CALC
BCALC_DESC_BILL	A80	CI_BSEG_CALC
SEQNO	N5	CI_BSEG_CALC_LN
PRT_SW	A1	CI_BSEG_CALC_LN
APP_IN_SUMM_SW	A1	CI_BSEG_CALC_LN
CALC_LN_AMT	N15.2	CI_BSEG_CALC_LN
DST_ID	A10	CI_BSEG_CALC_LN
MSR_PEAK_QTY_SW	A1	CI_BSEG_CALC_LN

DESCR_LN	A80	CI_BSEG_CALC_LN
STATE	A6	CI_PREM
CITY	A30	CI_PREM
POSTAL	A12	CI_PREM
SIBLING_ID	A12	CI_FT

Account Summary Information

Field Name	Format	Source/Value/Description
CURRENCY_CD	A3	Derived from CI_BILL_SA or CI_FT.
SUM_CUR_AMT	S15.2	Get Bill's Current Balance from CI_BILL_SA Bill's Current Charge from CI_FT Bill's Current Correction Charge from CI_FT Bill's Current Adjustment from CI_FT Bill's Current Payment from CI_FT
SUM_TOT_AMT	S15.2	Get Bill's Total Balance from CI_BILL_SA Bill's Total Charge from CI_FT Bill's Total Correction Charge from CI_FT Bill's Total Adjustment from CI_FT Bill's Total Payment from CI_FT
BILL_PRT_FLG	A4	Set as follows: ASBL - Indicator for Bill's Balance from CI_BILL_SA ASBS - Indicator for Bill's Current Charge from CI_FT ASBC - Indicator for Bill's Correction from CI_FT ASAD - Indicator for Bill's Adjustment from CI_FT ASPS - Indicator for Bill's Payment from CI_FT

Current Charge Information

Field Name	Format	Source/Value/Description
SUM_CUR_AMT	S15.2	Derived from CI_FT (Only the current charge bill segments.)
SUM_TOT_AMT	S15.2	Derived from CI_FT (Only the current charge bill segments.)
CURRENCY_CD	A3	CI_BILL_SA or CI_FT
DEBT_CL_CD	A4	Debt Class of SA Type
DESCR	A60	Debt Class Description

Premise Information

One record for every premise linked to the bill.

Field Name	Format	Source/Value/Description
PREM_ID	A10	Premise Id obtained from Bill Segment

PREM_TYPE_CD	A8	CI_PREM
LL_ID	A10	CI_PREM
MAIL_ADDR_SW	A1	CI_PREM
TREND_AREA_CD,	A8	CI_PREM
COUNTRY	A3	CI_PREM
ADDRESS1	A64	CI_PREM
ADDRESS2	A64	CI_PREM
ADDRESS3	A64	CI_PREM
ADDRESS4	A64	CI_PREM
CITY	A30	CI_PREM
NUM1	A6	CI_PREM
NUM2	A4	CI_PREM
HOUSE_TYPE	A2	CI_PREM
COUNTY	A30	CI_PREM
STATE	A6	CI_PREM
POSTAL	A12	CI_PREM
GEO_CODE	A11	CI_PREM
IN_CITY_LIMIT	A1	CI_PREM

Auto Pay Information

Field Name	Format	Source/Value/Description
BILL_ID	A12	CI_APAY_CLR_STG
APAY_SRC_CD	A12	CI_APAY_CLR_STG
EXT_ACCT_ID	A50	CI_APAY_CLR_STG
SCHED_EXTRACT_DT	Date	CI_APAY_CLR_STG. The date that the automatic payment will be downloaded.
APAY_SRC_NAME	A30	CI_APAY_SRC_L
APAY_SRC_DESCR	A60	CI_APAY_SRC_L
TNDR_TYPE_DESCR	A60	CI_TENDER_TYPE_L
TENDER_TYPE_CD	A4	CI_TENDER_TYPE

Bill Segment Readings

Field Name	Format	Source/Value/Description
BSEG_ID	A12	CI_BSEG_READ
SP_ID	A10	CI_BSEG_READ
SEQNO	N5	CI_BSEG_READ
REG_CONST	S12.6	CI_BSEG_READ
USAGE_FLG	A2	CI_BSEG_READ

USE_PCT	S3	CI_BSEG_READ
HOW_TO_USE_FLG	A2	CI_BSEG_READ
MSR_PEAK_QTY_SW	A1	CI_BSEG_READ
UOM_CD	A4	CI_BSEG_READ
TOU_CD	A8	CI_BSEG_READ
SQI_CD	A8	CI_BSEG_READ
START_REG_READ_ID	A12	CI_BSEG_READ
START_READ_DTTM	DTTM	CI_BSEG_READ
START_REG_READING	S15.6	CI_BSEG_READ
END_REG_READ_ID	A12	CI_BSEG_READ
END_READ_DTTM	DTTM	CI_BSEG_READ
END_REG_READING	S15.6	CI_BSEG_READ
MSR_QTY	S18.6	CI_BSEG_READ
FINAL_UOM_CD	A4	CI_BSEG_READ
FINAL_TOU_CD	A8	CI_BSEG_READ
FINAL_SQI_CD	A8	CI_BSEG_READ
FINAL_REG_QTY	S18.6	CI_BSEG_READ
BADGE_NBR	A30	CI_MTR
NBR_OF_DGTS_RGT	S2	CI_REG

Bill Segment Consumption History

Field Name	Format	Source/Value/Description
SA_ID	A10	CI_BSEG
START_DT	Date	CI_BSEG
BSEG_ID	A12	CI_BSEG
END_DT	Date	CI_BSEG
UOM_CD	A4	CI_BSEG_SQ
TOU_CD	A8	CI_BSEG_SQ
SQI_CD	A8	CI_BSEG_SQ
QTY	S18.6	Service Quantity from CI_BSEG_SQ

Payment Information

One record for every payment linked to the bill.

Field Name	Format	Source/Value/Description
PAY_DT	Date	CI_PAY_EVENT
FT_TYPE_FLG	A2	Pay or Pay Cancel ('PS' or 'PX')
CAN_RSN_CD	A4	CI_PAY
CURRENCY_CD	A3	CI_FT.

CUR_AMT	S15.2	Derived from CI_FT. Sum of CUR_AMT for the bill with FT Type Flag = (PS or PX) and Show on Bill Switch = 'Yes'
TOT_AMT	S15.2	Derived from CI_FT. Sum of CUR_AMT for the bill with FT Type Flag = (PS or PX) and Show on Bill Switch = 'Yes'
TOT_PREV_BAL	S15.2	Calculated as Previous Balance = Ending Balance - (current charges + payments + adjustments + corrections)
PAY_CAN_RSN_DESCR	A60	Description of cancel reason code. (Note, that this is retrieved in a special stored procedure that retrieves all the cancel reason codes and descriptions.)

Adjustment Information

One record for every adjustment linked to the bill.

Field Name	Format	Source/Value/Description
BILL_ID	A12	CI_FT
ACCOUNTING_DT	Date	CI_FT
ARS_DT	Date	CI_FT
PARENT_ID	A12	CI_FT
SIBLING_ID	A12	CI_FT
FT_ID	A12	CI_FT
CUR_AMT	S15.2	CI_FT
TOT_AMT	S15.2	CI_FT
FT_TYPE_FLG	A2	Adjustment or Adjustment Cancel ('AD' or 'AX')
SHOW_ON_BILL_SW	A1	CI_FT
CURRENCY_CD	A3	CI_FT
XFERRED_OUT_SW	A1	CI_FT
ADJ_TYPE_CD	A8	CI_ADJ
CAN_RSN_CD	A4	CI_ADJ
SA_ID	A10	CI_ADJ
CHAR_PREM_ID	A10	CI_SA
DESCR_ON_BILL	A80	CI_ADJ_TYPE_L
ADJ_CAN_RSN_DESCR	A60	Description of cancel reason code. (Note that this is retrieved in a special stored procedure that retrieves all the cancel reason codes and descriptions.)

Bill Message Information

One record for every bill message linked to the bill.

Field Name	Format	Source/Value/Description
MSG_PRIORITY_FLG	A12	CI_BILL_SA or CI_FT
INSERT_CD	S15.2	CI_BILL_SA or CI_FT

MSG_ON_BILL	S15.2	CI_BILL_SA or CI_FT
-------------	-------	---------------------

Billed Revenues by Rate - CI_BILREV

Parameters

Parameter	Parameter Code	Description
Accounting Period	P_ACCT_PERIOD	Defines the accounting period used for the report. A valid fiscal year and accounting period for a valid accounting calendar must be provided.
Account Type Characteristic	P_CHAR_TYPE	Defines a Characteristic Type for a characteristic linked to the Distribution Code to define an Account Type.
Account Type	P_REV_ACCTY_CHAR	Account Type Char Value for Revenue related GL Accounts. The char type defined for this parameter should match the Char Type code defined as parameter #2. The parameter value indicated for this parameter should be one that represents revenue accounts.

Report Description

This is an analysis report for the billed revenues for an accounting period according to the various rates, which were in effect in the system. The information in this report helps to adjust rates in order to achieve better financial results and comply with regulations and market trends.

This report selects all records in the financial transaction GL collection that satisfy the following criteria:

- The financial transaction is frozen.
- The Accounting Date on the financial transaction within input accounting period (parameter 1)
- The distribution code associated with the GL entry has a characteristic type and value that matches the input Account Type Characteristic and Account Type (parameters 2 & 3)

Performance Consideration. If your implementation chooses to use this report, you may consider adding an index to the CI_FT table on ACCOUNTING_DT to aid in performance. When making this decision, carefully weigh the benefit of improving report performance against the possible degradation to the performance of day-to-day processing as a result of defining a new index. Note that many companies opt to create a reporting database that is a shadow of production to ensure that indexes defined to benefit reports may be created without any affect on the production environment.

Case Statistics By Case Type - CI_CSESTS

Parameters

Parameter	Parameter Code	Description
Start Date	P_FROM_DT	See the report's description for how this field is used. If start date is not specified, it is defaulted to 7 days prior to

		the end date.
End Date	P_TO_DT	See the report's description for how this field is used. If end date is not specified, it is defaulted to the current processing date.
Case Condition (Open, Closed)	P_COND_FLG	If specified, only cases in this condition are included in the report. If left blank, the reports produces statistics for both open and closed cases.

Report Description

This report provides two types of statistics:

1. Open cases whose creation date falls between the input Start Date and End Date (inclusive)
2. Closed cases whose closing date falls between the input Start Date and End Date (inclusive)

The third parameter is only used if you want to restrict the statistics to only open or closed cases. If you leave this parameter blank, both open and closed statistics will be produced.

The following information is provided in graphical format:

- Number of cases by case type
- Percentage of cases by case type

Case Statistics for a Given Status - CI_CSESGS

Parameters

Parameter	Parameter Code	Description
Start Date	P_FROM_DT	See the report's description for how this field is used. If start date is not specified, it is defaulted to 7 days prior to the end date.
End Date	P_TO_DT	See the report's description for how this field is used. If end date is not specified, it is defaulted to the current processing date.
Case Type/Status	P_CASE_STATUS_CD	This is the desired Case Type and Status that will be reported on.
Responsible User	P_CASE_OWNER	If specified, only cases with this responsible user are included in the report.
First Bucket High Limit	P_B1_LIMIT	Cases that took <= this number of days to reach the given status will be grouped together for statistical reporting.
Second Bucket High Limit	P_B2_LIMIT	Cases that took <= this number of days but more than the first bucket high limit to reach the given status will be grouped together for statistics reporting.
Third Bucket High Limit	P_B3_LIMIT	Cases that took <= this number of days but more than the second bucket high limit to reach the given status will be grouped together for statistics reporting. Cases that took more than this number of days are included in another group.

Report Description

This report shows cases of a given case type that transitioned to a given status during a given date range.

Graphs are printed to show the number and percentage of cases grouped by the time it took to reach the status. These statistics are grouped into age buckets whose size is controlled by the last 3 parameters.

Summary statistics are also printed showing the minimum, maximum, average and median times for these cases.

Collection Summary - CI_CLLSUM**Parameters**

Parameter	Parameter Code	Description
Start Date	P_FROM_DT	If from date is not set, the default is one month prior to the current date.
End Date	P_TO_DT	End date of the date range. If not defined by user, it is set to one month after the current date

Report Description

This report provides detailed monthly summary information of all collection activities. The report is typically used by a collection department for resource planning and performance review purposes.

This report selects pending and completed collection events whose event trigger date falls between the input start and end dates.

Customer Contacts by Type - CI_CUSTCN**Parameters**

Parameter	Parameter Code	Description
Start Date	P_FROM_DT	Start date to use for reporting customer contacts. If not defined, the start date is set to the current date minus 7 days.
End Date	P_TO_DT	End date to use for reporting customer contacts. If not defined, End Date is set to the current date.
Customer Contact Class / Type	P_CC_TYPE_CD	Specify a Customer Contact Class / Type to restrict the report output to a specific class / type.

Report Description

This report lists all customer contacts in the system created within the input date range. You may optionally restrict the report to customer contacts for a given Customer Contact Class / Type (parameter 3).

Graphs. The information on this report is shown in both textual and graphical formats.

Performance Consideration. If your implementation chooses to use this report, you may consider adding an index to the CI_CC table on CC_DTTM to aid in performance. When making this decision, carefully weigh the benefit of improving report performance against the possible degradation to the performance of day-to-day processing as a result of defining a new index. Note that many companies opt to create a reporting database that is a shadow of production to ensure that indexes defined to benefit reports may be created without any affect on the production environment.

Customers with Life-Support / Sensitive-Load - CI_PMLSSL

Parameters

Parameter	Parameter Code	Description
Service Type	P_SERVICE_TYPE	Service type used to restrict the report to premises with services of this type.

Report Description

This reports displays a detailed list of premises that are coded with Life-Support (LS) or Sensitive-Load (SL) information. It may optionally restrict the output to premises with service points for input service type.

This information is used by a company to make sure that customers at these premises are dealt with appropriately in the case of an outage (planned and unplanned) or service cut due to non-payment.

The report provides detailed information about the premise and the related facility elements providing service to that premise (e.g. substation, feeder and node for electric service).

Note. If ANY person connected to an account has an LS/SL indication in the Person record, then all the premises connected to this account will be treated as LS/SL. This means, for example, that a premise connected to an account that has a 3rd party guarantor with LS/SL is treated as a premise with LS/SL.

Field Order Print in BI Publisher and Crystal - CI_FOPRNT

Note. Field Order Print in batch for BI Publisher is not supported yet.

Parameters

Parameter	Parameter Code	Description
CC&B user	P_USER_ID	No default value. Required. Returns an error message if user ID is null, empty or does not exist in database.

Batch Switch	P_BATCH_SW	For a single FO print, Batch Switch is 'No' and for multiple FO print, Batch Switch is 'Yes'.
First FO	P_FROM_FO_ID	First FO ID to print for the FO Range.
Last FO	P_TO_FO_ID	Last FO ID to print for the FO Range.
Batch Code	P_BATCH_CD	Batch Code passed in by the system the report is submitted in batch. Only field orders that reference this batch code are selected.
Batch Number	P_BATCH_NBR	Batch Code passed in by the system the report is submitted in batch. Only field orders that reference this batch number are selected.
Extract Algorithm	P_EXTRACT_ALG	Passed in by the system the report is submitted in batch. Only field orders whose dispatch groups reference this FO extract algorithm are selected.

Report Description

This report is used with BI Publisher and Crystal Reports to display all the Field Order information for the field order ID or range of field order IDs entered in the input parameters. For a report submitted in batch to Crystal Reports, the parameters batch code, batch number and extract algorithm are also used to limit the field orders selected. Refer to the description of the parameters above. The field order will only be displayed if the field order's status is dispatched.

The following sections highlight the data that is extracted for this report.

Contents

- Field Order Information
- Account / Person Information
- Person Phone Information
- Premise Geo Information
- SP Type Information
- SP Installed Meter Information
- SP Installed Item Information
- SP Geo Information
- SP Characteristics Information
- SP Multi Item Information
- FA Severance Process Information

Field Order Information

Field Name	Format	Source/Value/Description
LANGUAGE_CD	A3	SC_USER
SORT_KEY	A10	FO Id
FO_ID	A10	CI_FO
PREM_ID	A10	CI_FO
FO_SCHED_DTTM	Date	CI_FO.SCHED_DTTM
FO_STATUS_FLG	A2	CI_FO
WORK_DTTM	Date	CI_FO
DISP_GRP_CD	A8	CI_FO

REP_CD	A8	CI_FO
WORKED_BY	A8	CI_FO
EXTRACT_NEXT_SW	A1	CI_FO
EXTRACT_DTTM	Date	CI_FO
FO_DESCR	A254	CI_FO.DESCR254
BATCH_CD	A8	CI_FO_STG_DWN
BATCH_NBR	N10	CI_FO_STG_DWN
FA_ID	A10	CI_FA
SP_ID	A10	CI_FA
FA_TYPE_CD	A8	CI_FA
FA_PRIORITY_FLG	A2	CI_FA
FA_CREATED_BY_FLG	A2	CI_FA
FA_SCHED_DTTM	Date	CI_FA.SCHED_DTTM
FA_STATUS_FLG	A2	CI_FA
ELIG_DISPATCH_SW	A1	CI_FA
CRE_DTTM	Date	CI_FA
INSTRUCTIONS	A254	CI_FA
TEST_SEL_ID	A10	CI_FA
FA_DESCR	A254	CI_FA.DESCR254
FA_CAN_RSN_CD	A8	CI_FA_STEP
STEP_SEQ_NBR	N3	CI_FA_STEP
FA_STEP_TY_ACT_FLG	A2	CI_FA_STEP
STP_ENTITY_FLG	A4	CI_FA_STEP
SP_MTR_HIST_ID	A10	CI_FA_STEP
SP_ITEM_HIST_ID	A10	CI_FA_STEP
MR_ID	A12	CI_FA_STEP
MTR_CFG_MTR_ID	A10	CI_FA_STEP
MTR_ID	A10	CI_FA_STEP
ITEM_ID	A10	CI_FA_STEP
CC_ID	A10	CI_FA_STEP
SPAWNED_FA_ID	A10	CI_FA_STEP
ACCT_ID	A10	CI_FA_STEP
DV_TEST_ID	A10	CI_FA_STEP
PREM_TYPE_CD	A8	CI_PREM
LL_ID	A10	CI_PREM
MAIL_ADDR_SW	A1	CI_PREM
KEY_SW	A1	CI_PREM

KEY_ID	A10	CI_PREM
OK_TO_ENTER_SW	A1	CI_PREM
MR_INSTR_CD	A4	CI_PREM
MR_INSTR_DETAILS	A250	CI_PREM
MR_WARN_CD	A4	CI_PREM
TREND_AREA_CD	A8	CI_PREM
COUNTRY	A3	CI_PREM
ADDRESS1	A64	CI_PREM
ADDRESS2	A64	CI_PREM
ADDRESS3	A64	CI_PREM
ADDRESS4	A64	CI_PREM
CITY	A30	CI_PREM
NUM1	A6	CI_PREM
NUM2	A5	CI_PREM
HOUSE_TYPE	A2	CI_PREM
COUNTY	A30	CI_PREM
STATE	A6	CI_PREM
POSTAL	A12	CI_PREM
GEO_CODE	A11	CI_PREM
IN_CITY_LIMIT	A1	CI_PREM
FA_STEP_TY_DESCR	A60	CI_LOOKUP.DESCR / FA Step Type Action Flag
OPTIONAL_SW	A1	CI_FA_STEP_TYPE
FA_STEP_TYPE_DESCR	A60	CI_FA_STEP_TYPE_L.DESCR
REP_DESCR	A60	CI_REP_L.DESCR
FA_TYPE_DESCR	A60	CI_FA_TYPE_L.DESCR
DISP_GRP_DESCR	A60	CI_DISP_GRP_L.DESCR
LL_ACCT_ID	A10	CI_LANDLORD.ACCT_ID
LL_DESCR	A60	CI_LANDLORD_L.DESCR
MR_WARN_DESCR	A60	CI_MR_WARN_L.DESCR
MR_INSTR_DESCR	A60	CI_MR_INSTR_L.DESCR

Account / Person Information

Field Name	Format	Source/Value/Description
PREM_ID	A10	CI_FO
ACCT_ID	A10	CI_SA
MAIN_CUST_SW	A1	CI_ACCT_PER
PER_ID	A10	CI_ACCT_PER

ACCT_REL_TYPE_CD	A8	CI_ACCT_PER
ENTITY_NAME	A64	CI_PER_NAME
LS_SL_FLG	A2	CI_PER
LS_SL_DESCR	A254	CI_PER

Person Phone Information

Field Name	Format	Source/Value/Description
PER_ID	A10	CI_ACCT_PER - Primary person
SEQ_NUM	N3	CI_PER_PHONE
PHONE_TYPE_CD	A12	CI_PER_PHONE
COUNTRY_CODE	A3	CI_PER_PHONE
PHONE	A24	CI_PER_PHONE
EXTENSION	A6	CI_PER_PHONE
DESCR	A60	CI_PHONE_TYPE_L

Premise Geo Information

Field Name	Format	Source/Value/Description
PREM_ID	A10	CI_FO
GEO_TYPE_CD	A8	CI_PREM_GEO
GEO_VAL	A50	CI_PREM_GEO
GEO_TYPE_DESCR	A60	CI_GEO_TYPE_L.DESCR

SP Type Information

Field Name	Format	Source/Value/Description
SP_ID	A10	CI_FA
SP_DESCR	A254	CI_SP.DESCR254
SP_TYPE_CD	A8	CI_SP
SP_TYPE_DESCR	A60	CI_SP_TYPE_L.DESCR
SP_SUBTYPE_FLG	A2	CI_SP_TYPE
SP_SUBTYPE_DESCR	A60	CI_LOOKUP/SP Subtype Flag
SP_STATUS_FLG	A2	CI_SP
INSTALL_DT	Date	CI_SP
SP_SRC_STATUS_FLG	A2	CI_SP
DISCON_LOC_CD	A4	CI_SP
DISCON_LOC_DESCR	A60	CI_DISCON_LOC_L
MR_CYC_CD	A4	CI_SP
MR_CYC_DESCR	A60	CI_MR_CYC_L
MTR_LOC_CD	A4	CI_SP

MTR_LOC_DESCR	A60	CI_MTR_LOC_L
MTR_LOC_DETAILS	A250	CI_SP
FAC_LVL_1_CD	A8	CI_SP
FAC_LVL_1_DESCR	A60	CI_FAC_LVL_1_L.DESCR
FAC_LVL_2_CD	A8	CI_SP
FAC_LVL_2_DESCR	A60	CI_FAC_LVL_3_L.DESCR
FAC_LVL_3_CD	A8	CI_SP
FAC_LVL_3_DESCR	A60	CI_FAC_LVL_3_L.DESCR

SP Installed Meter Information

Field Name	Format	Source/Value/Description
SP_ID	A10	CI_FA
SP_MTR_HIST_ID	A10	CI_SP_MTR_HIST
MTR_CONFIG_ID	A10	CI_SP_MTR_HIST
EFF_DTTM	Date	CI_MTR_CONFIG
MTR_ID	A10	CI_MTR_CONFIG
INSTALL_DTTM	Date	CI_MR.READ_DTTM
READ_DTTM	Date	CI_MR
BADGE_NBR	A30	CI_MTR
MTR_TYPE_CD	A8	CI_MTR
MTR_STATUS_FLG	A2	CI_MTR
MFG_CD	A8	CI_MTR
MODEL_CD	A8	CI_MTR
SERIAL_NBR	A16	CI_MTR
RECEIVE_DT	Date	CI_MTR
MTR_DESCR	A254	CI_MTR.DESCR254
REG_ID	A10	CI_REG
READ_SEQ	N2	CI_REG
UOM_CD	A4	CI_REG
TOU_CD	A8	CI_REG
REG_CONST	N12.6	CI_REG
CONSUM_SUB_FLG	A2	CI_REG
HOW_TO_USE_FLG	A2	CI_REG
NBR_OF_DGTS_LFT	N2	CI_REG
NBR_OF_DGTS_RGT	N2	CI_REG
FULL_SCALE	N18.7	CI_REG
READ_OUT_TYPE_CD	A8	CI_REG

PROTOCOL_CD	A8	CI_REG
TOLERANCE	N14.5	CI_REG
REG_READ_ID	A12	CI_REG_READ
MR_ID	A12	CI_REG_READ
READ_TYPE_FLG	A2	CI_REG_READ
REG_READING	N15.6	CI_REG_READ
MTR_TYPE_DESCR	A60	CI_MTR_TYPE_L.DESCR
MFG_DESCR	A60	CI_MFG_L.DESCR
MODEL_DESCR	A60	CI_MODEL_L.DESCR

SP Installed Item Information

Field Name	Format	Source/Value/Description
SP_ID	A10	CI_FA
SP_ITEM_HIST_ID	A10	CI_SP_ITEM_HIST
ITEM_ID	A10	CI_SP_ITEM_HIST
INSTALL_DTTM	Date	CI_SP_ITEM_EVT.EVENT_DTTM
BADGE_NBR	A30	CI_ITEM
ITEM_TYPE_CD	A8	CI_ITEM
ITEM_STATUS_FLG	A2	CI_ITEM
MFG_CD	A8	CI_ITEM
MODEL_CD	A8	CI_ITEM
SERIAL_NBR	A16	CI_ITEM
RECEIVE_DT	Date	CI_ITEM
ITEM_DESCR	A254	CI_ITEM.DESCR254
ITEM_TYPE_DESCR	A60	CI_ITEM_TYPE_L.DESCR
MFG_DESCR	A60	CI_MFG_L.DESCR
MODEL_DESCR	A60	CI_MODEL_L.DESCR

SP Geo Information

Field Name	Format	Source/Value/Description
SP_ID	A10	CI_FA
GEO_TYPE_CD	A8	CI_SP_GEO
GEO_VAL	A50	CI_SP_GEO
GEO_TYPE_DESCR	A60	CI_GEO_TYPE_L.DESCR

SP Characteristics Information

Field Name	Format	Source/Value/Description
SP_ID	A10	CI_FA

CHAR_TYPE_CD	A8	CI_SP_CHAR
EFFDT	Date	CI_SP_CHAR
CHAR_VAL	A16	CI_SP_CHAR
ADHOC_CHAR_VAL	A254	CI_SP_CHAR
CHAR_VAL_FK1	A50	CI_SP_CHAR
CHAR_VAL_FK2	A50	CI_SP_CHAR
CHAR_VAL_FK3	A50	CI_SP_CHAR
CHAR_VAL_FK4	A50	CI_SP_CHAR
CHAR_VAL_FK5	A50	CI_SP_CHAR
CHAR_TYPE_FLG	A4	CI_CHAR_TYPE
FK_REF_CD	A8	CI_CHAR_TYPE
CHAR_TYPE_DESCR	A60	CI_CHAR_TYPE_L.DESCR

SP Multi Item Information

Field Name	Format	Source/Value/Description
SP_ID	A10	CI_FA
EFFDT	Date	CI_MULT_ITEM
ITEM_TYPE_CD	A8	CI_MULT_ITEM
ITEM_CNT	N11,2	CI_MULT_ITEM
ITEM_TYPE_DESCR	A60	CI_ITEM_TYPE_L.DESCR

FA Severance Process Information

Field Name	Format	Source/Value/Description
FA_ID	A10	CI_FA
SEV_PROC_ID	A10	CI_SEV_PROC
SA_ID	A10	CI_SEV_PROC
SEV_ARS_DT	Date	CI_SEV_PROC
ACCT_ID	A10	CI_SA

GL Accounting Summary - CI_GLACSM

Parameters

Parameter	Parameter Code	Description
Accounting Period	P_ACCT_PERIOD	Defines the accounting period used for the report. A valid fiscal year and accounting period for a valid accounting calendar must be provided.
Account Type Characteristic	P_CHAR_TYPE	Defines a Characteristic Type for a characteristic linked to the Distribution Code to define an Account Type. The account types for the GL accounts are used for grouping the

		output to the report.
--	--	-----------------------

Report Description

This is a financial audit report used to check the financial details in Oracle Utilities Customer Care and Billing for an accounting period against the GL system. The report summarizes all financial transaction (FT) information for a given accounting period according to the different operating and GL divisions and according to various levels of the account GL information.

Performance Consideration. If your implementation chooses to use this report, you may consider adding an index to the CI_FT table on ACCOUNTING_DT to aid in performance. When making this decision, carefully weigh the benefit of improving report performance against the possible degradation to the performance of day-to-day processing as a result of defining a new index. Note that many companies opt to create a reporting database that is a shadow of production to ensure that indexes defined to benefit reports may be created without any affect on the production environment.

Letter Print - BI Publisher and Crystal Sample Welcome Letters - CI_LTRGN_ENG REPORT

Note. Generating letters in batch is not supported yet.

Parameters

Parameter	Parameter Code	Description
Batch Switch	P_BATCH_SW	For reports accessed online, set the switch to 'No'. Otherwise, it should be set to 'Yes'.
Customer Contact	P_CC_ID	References the customer contact associated with the main customer linked to the account.

Report Description

This sample report templates for BI Publisher and Crystal Reports produce letters that are not associated with any other object (i.e., the template does not have to extract information from another object to merge into a letter). You can use these templates as a welcome letter for a new customer.

By default, the English versions of the report templates are provided with the base product. If multilingual report templates are required, your implementation should provide reports for each language. When a letter is generated, the system uses the report template based on the customer's language.

The address and name for the company are extracted from the [installation options](#). The text for the letter is defined in the report layout and not provided by Oracle Utilities Customer Care and Billing. Reports are printed according to the customer's language definition and not based on the user's language definition.

This sample report uses the following text:

Welcome to %1. You have been filed with ID Number %2.

We hope to provide you with our best possible service. If you experience any problems or have any questions, please contact one of our customer service representatives at (800)1234567.

%1 is the company name stored as a message on the [installation options](#).

%2 is Person Id stored on the Customer Contact.

Meter Reads Performance - CI_MTREAD

Parameters

Parameter	Parameter Code	Description
Accounting Period	P_ACCT_PERIOD	Defines the accounting period used for the report. A valid fiscal year and accounting period for a valid accounting calendar must be provided.

Report Description

This report selects bill segments for the input accounting period and displays the total number of read meters and unread meters for these bill segments grouped by route type.

Meters that are considered read are meters whose register reads have a status of **Regular** or **Verified**. Meters that are considered unread are meters whose register reads have a status of **System Estimate**, **Office Estimate**, **System Prorate** or **Billing Force**.

Performance Consideration. If your implementation chooses to use this report, you may consider adding an index to the CI_FT table on ACCOUNTING_DT to aid in performance. When making this decision, carefully weigh the benefit of improving report performance against the possible degradation to the performance of day-to-day processing as a result of defining a new index. Note that many companies opt to create a reporting database that is a shadow of production to ensure that indexes defined to benefit reports may be created without any affect on the production environment.

Open Cases By Type - CI_CSEOPN

Parameters

Parameter	Parameter Code	Description
Start Date	P_FROM_DT	See the report's description for how this field is used. If start date is not specified, it is defaulted to 7 days prior to the end date.
End Date	P_TO_DT	See the report's description for how this field is used. If end date is not specified, it is defaulted to the current processing date.
Case Type	P_CASE_TYPE_CD	If specified, only cases of this type are included in the report.
Responsible User	P_CASE_OWNER	If specified, only cases with this responsible user are included in the report.

First Bucket High Limit	P_B1_LIMIT	Cases that are open for less than or equal to this number of days will be grouped together for statistical reporting.
Second Bucket High Limit	P_B2_LIMIT	Cases that are open less than or equal to this number of days but more than the first bucket high limit will be grouped together for statistics reporting.
Third Bucket High Limit	P_B3_LIMIT	Cases that are open less than or equal to this number of days but more than the second bucket high limit will be grouped together for statistics reporting. Cases that took more than this number of days are included in another group.

Report Description

This is a report on open cases that were created between a given date range.

The report can be limited to a specific type and/or responsible user.

For each case type, the report shows the following:

- Number of open cases by age bucket (the last 3 parameters control the size (in days) of each bucket)
- Percentage of open cases by age bucket
- Details of the open cases

Payments Balance - CI_PMTBAL

Parameters

Parameter	Parameter Code	Description
Start Date	P_FROM_DT	Report gets all the payments that have been received during a given date range (from and to date parameters). If start date is not defined by the user, it is set to 7 days prior to the current date.
End Date	P_TO_DT	End date of the date range. If not defined by user it is set to the current date

Report Description

This report provides an overall view of all payments created within the input date range. It is typically used for financial control and audit purposes. The report provides summary information about valid payments received and about canceled payment. Data is summarized by the tender source and the type of payment.

Receivables Aging - CI_RCVAGA

Parameters

Parameter	Parameter Code	Description
Cutoff Date	P_CUTOFF_DATE	The date from which the arrears buckets are calculated. If no value is entered, the default is the current date minus 7

		days.
1 st Bucket High Limit	P_B1_LIMIT	High limit of 1st bucket.
2 nd Bucket High Limit	P_B2_LIMIT	High limit of 2 nd bucket.
3 rd Bucket High Limit	P_B3_LIMIT	High limit of 3 rd bucket.

Report Description

This report lists all accounts and their arrears information as of the input cutoff date using a balance forward accounting method.

Outstanding debt is placed into the buckets provided as input using the age of the debt as of the cutoff date. Credits are applied to the oldest debt first. For each account a separate bucket is used to display new charges. In addition, the total accounts receivable balance is displayed for each account.

Performance Consideration. If your implementation chooses to use this report, you may consider adding an index to the CI_FT table on ARS_DT to aid in performance. When making this decision, carefully weigh the benefit of improving report performance against the possible degradation to the performance of day-to-day processing as a result of defining a new index. Note that many companies opt to create a reporting database that is a shadow of production to ensure that indexes defined to benefit reports may be created without any affect on the production environment.

Tax Payables Analysis - CI_TXPYBL

Parameters

Parameter	Parameter Code	Description
Start Date	P_FROM_DT	Show summary of the tax amounts starting from this date. If not specified, the system will default this value to the current date minus 7 days.
End Date	P_TO_DT	Show summary of the tax amounts up to this date. If not specified, the system will default this value to the current date.
Account Type Characteristic	P_CHAR_TYPE	Defines a Characteristic Type for a characteristic linked to the Distribution Code to define an Account Type.
Account Type	P_TAX_ACCTY_CHAR	Account Type Char Value for tax related GL Accounts. The char type defined for this parameter should match the Char Type code defined as parameter #3. The parameter value indicated for this parameter should be one that represents tax liability accounts.

Report Description

This report displays a summary of the tax amounts that were levied by the company to customers within the input date range. It also includes the tax exemption information for that period.

This report select all records in the financial transaction GL collection that satisfy the following criteria:

- The financial transaction is frozen.
- The Accounting Date on the financial transaction within the input date range
- The distribution code associated with the GL entry has a characteristic type and value that matches the input Account Type Characteristic and Account Type (parameters 3 & 4)

The report also provides tax exemption information for bill segments whose financial transactions satisfy the above criteria. The tax exemption information is retrieved by looking at the bill calculation lines associated with the FT's bill segment.

Performance Consideration. If your implementation chooses to use this report, you may consider adding an index to the CI_FT table on ACCOUNTING_DT to aid in performance. When making this decision, carefully weigh the benefit of improving report performance against the possible degradation to the performance of day-to-day processing as a result of defining a new index. Note that many companies opt to create a reporting database that is a shadow of production to ensure that indexes defined to benefit reports may be created without any affect on the production environment.

To Do Entries - CI_TDENTR

Parameters

Parameter	Parameter Code	Description
ToDo Entry Status	P_ENTRY_STATUS_FLG	Defines if the To Do entries on the report should be limited to those with a given status value. If this parameter is left blank, the report will show all <i>open</i> and <i>being worked</i> To Do entries.
ToDo Type	P_TD_TYPE_CD	Defines if the To Do entries on the report should be limited to those of a given ToDo type. If this parameter is left blank, the report will show all ToDo type that have at least one <i>open</i> or <i>being worked</i> entry.

Report Description

The report shows open and being worked To Do entries.

You can limit the report to entries in a given status by specifying the desired **To Do Status** (open or being worked). If you don't specify a status, all *open* and *being worked* To Do entries will appear on this report.

You can also limit the report to entries of a given To Do Type by specifying the desired **To Do Type**. If you don't specify a To Do Type, all To Do Types with at least one entry in the *open* / *being worked* state will appear on this report.

Graphs. The information on this report is shown in both textual and graphical formats.

Umbrella Agreement Summary - CI_UASUMM

Parameters

At least one parameter is required. If the umbrella agreement is entered, no other parameter value is allowed. The report definition uses the validation algorithm [RPTV-UASUMM](#) to check these conditions.

Parameter	Parameter Code	Description
Umbrella Agreement	P_UA_ID	Specify the Umbrella Agreement to restrict the report to this umbrella agreement (regardless of its status). When this parameter is specified, all other parameters are not allowed.
Account Management Group	P_AMG	Specify a value for this parameter to restrict the report to umbrella agreements related to this Account Management Group.
Umbrella Agreement Characteristic Type	P_CHAR_TYPE	Specify a value for this parameter to restrict the report to completed umbrella agreements related to this Characteristic type and value (parameter 4).
Umbrella Agreement Characteristic Value	P_CHAR_VALUE	Specify a value for this parameter to restrict the report to completed umbrella agreements related to this Characteristic type (parameter 3) and value.
Number of Days Before Expiration	P_EXPIRE_IN_X_DAYS	Specify a value for this parameter to restrict the report to completed umbrella agreements whose End Date is on or before the current date less the Number of Days before Expiration.

Report Description

This report provides summary information about one or more [umbrella agreements](#).

If the user does not supply an umbrella agreement ID, but enters one or more of the other parameters, the report only selects umbrella agreements that match the input criteria AND the match the following criteria:

- The status of the umbrella agreement is **complete**
- The start and end dates of the umbrella agreement include the current date

The report does not include canceled terms of service records or canceled service agreements linked to the above umbrella agreements.

For each umbrella agreement that is included in the report, the output includes information about the umbrella agreement, its collection of terms of service records and for each TOS record, its collection of service agreements.

In addition, the report includes the following graphs:

- Umbrella agreement summary graph showing billing history by accounting period for the effective dates of the umbrella agreement for all service agreements linked to the umbrella agreement's terms of service records.
- Terms of service summary graph showing billing history by accounting period for the effective dates of the umbrella agreement for all its service agreements

Unbilled Revenues - CI_UBLREV

Parameters

Parameter	Parameter Code	Description
Accounting Period	P_ACCT_PERIOD	Defines the accounting period used for the report. A valid fiscal year and accounting period for a valid accounting calendar must be provided.
Account Type Characteristic	P_CHAR_TYPE	Defines a Characteristic Type for a characteristic linked to the Distribution Code to define an Account Type.
Account Type	P_REV_ACCTY_CHAR	Account Type Char Value for Revenue related GL Accounts. The char type defined for this parameter should match the Char Type code defined as parameter #2. The parameter value indicated for this parameter should be one that represents revenue accounts.

Report Description

This report provides a simplified calculation of estimated unbilled revenue for a given month.

It processes frozen financial transactions associated with bill segments and canceled bill segments whose bill segment end date is within the **Accounting Period** (parameter 1). It determines the billed revenue for the FT and then uses this information to calculate the unbilled revenue.

To determine "revenue", the report summarizes amounts posted to any distribution code with a characteristic entry that matches the input **Account Type Characteristic** type and **Account Type** characteristic value (parameters 2 and 3).

The estimate for the unbilled portion is calculated as (Bill Amount / Number of Days in the Bill Period) * Number of Unbilled Days for the Accounting Period.

For example, consider a bill segment on a monthly-billed cycle for the period of 3/10/03 - 4/9/03, and for \$150 (revenue part of the bill). Assume we are in the April/03 accounting period, which covers 4/1/03 through 4/30/03. This means 21 days are unbilled for April. The unbilled revenue is calculated as $150/31 \times 21 = \$101.61$.

The following are some points to note about this report.

- If the report runs for an historical date, it still estimates the unbilled revenue portion based on the above formula even if actual data exists. In other words, it will not try to find actual bills for subsequent months. As a result, this report always shows what would have been the estimated unbilled revenue for a particular month if that month is the most recent one.
- Services can be billed monthly bi-monthly, quarterly or in any desired frequency. This report only performs the revenue recognition estimation using actual bills that were created in the report's accounting period. As a result, it does not estimate unbilled revenue for accounts that were not billed in any portion of the input accounting period.
- Revenue recognition practices are unique and may vary from customer to customer. In a given month we can produce current bills (e.g. for electric service) or future bills (e.g. cable services). In addition in some cases we can have bills that started before the accounting period and ending after it (for example in a cancel-rebill situation). This report will only estimate unbilled revenues for bills with the accounting period equals to the report's accounting period.

Performance Consideration. If your implementation chooses to use this report, you may consider adding an index to the CI_BSEG table on START_DT, END_DT to aid in performance. When making this decision, carefully weigh the benefit of improving report performance against the possible degradation to the performance of day-to-day processing as a result of defining a new index. Note that many companies opt to create a reporting database that is a shadow of production to ensure that indexes defined to benefit reports may be created without any affect on the production environment.

Vacant Premises with Consumption - CI_VACANT

Report Description

This report shows all premises that are considered vacant, and provides information about the level of consumption, the period of vacancy, and the service point and register information.

This report selects service points that are linked to a service agreement that is either **canceled** or **closed**. The report excludes any service points that have never been linked to a service agreement. If consumption has been registered at such a service point since the end date of the service agreement, the service point's details are displayed.

This report may be used by the utility to investigate sites where problems such as service theft or leakage may be occurring.

Performance Consideration. If your implementation chooses to use this report, you may consider adding an index to the CI_SA_SP on STOP_DT to aid in performance. When making this decision, carefully weigh the benefit of improving report performance against the possible degradation to the performance of day-to-day processing as a result of defining a new index. Note that many companies opt to create a reporting database that is a shadow of production to ensure that indexes defined to benefit reports may be created without any effect on the production environment.

Security Addendum

This chapter is an addendum to the general [Defining Security and User Options](#) chapter. This addendum describes security functionality that is specific to Oracle Utilities Customer Care and Billing.

Contents

[Implementing Account Security](#)
[Masking Sensitive Data](#)

Implementing Account Security

Important! This section assumes you understand [The Big Picture of Row Security](#).

When you create an account, you must define which users can access the account's information. For example,

- If you have customers in two geographic territories, you may need to restrict access to accounts based on the office that manages each territory. For example, only users in the northern office may manage accounts in the northern territory.
- If you have industrial and residential customers, you may need to restrict access to these different customer segments based on the skill set of the users. For example, some users are skilled in dealing with industrial customers, while others are skilled in dealing with residential customers.

By granting a user access rights to an account, you are actually granting the user access rights to the account's bills, payment, adjustments, orders, etc.

Refer to [If You Do Not Practice Account Security](#) for setup instructions if your organization doesn't practice account security.

Account security may also affect persons and premises. Refer to [Persons Can Also Be Secured](#) for how access to person information is also restricted by account security. Refer to [Premises Can Also Be Secured](#) for how access to premise information is also restricted by account security.

The topics in this section describe how to implement account security.

Contents

[Persons Can Also Be Secured](#)
[Premises Can Also Be Secured](#)
[Data Becomes Invisible When Access Is Restricted](#)
[Account Security and Control Central](#)
[Restricted Transactions](#)
[Account Security Case Studies](#)
[The Default Access Group](#)
[If You Do Not Practice Account Security](#)

Persons Can Also Be Secured

It's important to be aware that persons can also be secured as a result of "account security". It works like this:

- If a person is linked to at least one account, users will not be allowed access to the person (or the person's related information) unless they have access to at least one of the person's accounts.
- If a person is not linked to any accounts (a rare situation), any user may access the person.

How are persons linked to accounts? A person is linked to an account when an account is created using the methods described under [How To Add A New Customer From Control Central](#) and [Order User Interface Flow](#). In addition, you may manually link and unlink persons from account using the [Account – Person](#) page.

Premises Can Also Be Secured

It's important to be aware that premises can also be secured as a result of "account security". It works like this:

- If a premise is linked to at least one account, users will not be allowed access to the premise (or the premise's related information) unless they have access to at least one of the premise's accounts.
- If a premise is not linked to an account (a rare situation), then all users may access the premise.

How are premises linked to accounts? A premise is indirectly linked to an account. For the purpose of access restriction, we deem a premise as being linked to an account if at least one of its service points is linked to at least one of the account's service agreements.

Data Becomes Invisible When Access Is Restricted

The following points summarize the impact of a user not having access to an account.

Account Security and Control Central

This section summarizes the impact of account security on [Control Central](#):

- Searches are affected as follows:
 - An account will only be visible if a user has access to the account's access group.
 - Persons that are not linked to accounts will be visible to all users.
 - If a person is linked to an account, the person will only be visible if the user has access to at least one of the person's accounts.
 - Premises that are not linked to accounts will be visible to all users.
 - If a premise is linked to an account, the premise will only be visible if the user has access to at least one of the premise's accounts.

- The alerts that highlight the existence of “multiple relationships” are not impacted by account security. Specifically:
 - The alert ***Person has multiple accounts*** will appear if the selected person is linked to multiple accounts, even if the user doesn’t have access to every account. Note well, the person couldn’t have been selected if the user didn’t have access rights to at least one account.
 - The alert ***Premise has multiple accounts*** will appear if the selected premise is linked to multiple account, even if the user doesn’t have access to every account. Note well, the premise couldn’t have been selected if the user didn’t have access rights to at least one account.
- Only accounts to which the user has access will be displayed in the person tree.
- Only accounts to which the user has access will be displayed in the account tree.
- All other pages contain information related to Control Central’s current account context. The current account context can never reference an inaccessible account and therefore these pages are not impacted by account security.

Contents

[Account Security and Searches \(and Maintenance Pages\)](#)

[Account Security and To Do Lists](#)

Account Security and Searches (and Maintenance Pages)

Searches are the gateway to the information that appears on maintenance pages. In general, account-related information is suppressed when a user doesn’t have access rights to the account. This suppression is true for rows that directly reference an account AND for rows that indirectly reference an account. For example:

- A user can only see bills associated with accounts to which they have access rights.
- A user can only see financial transactions associated with service agreements that are, in turn, associated with accounts to which they have access rights.

Person and premise searches are also impacted. Keep in mind that information will be suppress from both person and premise-oriented searches if the person / premise is related to accounts. Refer to [Persons Can Also Be Secured](#) for how access to person information is also restricted by account security. Refer to [Premises Can Also Be Secured](#) for how access to premise information is also restricted by account security.

Account Security and To Do Lists

Account security does NOT impact the information that appears in a user’s To Do list. Rather, we have assumed that your To Do roles (and the users assigned to these roles) are consistent with your account security requirements. This can result in anomalies. For example, it’s possible for a supervisor to assign a bill segment error to a user who doesn’t have access to the bill segment’s account. This user will then see the related To Do entry in their Bill Segments In Error To Do list. However, when they drill down on the entry, account security will manifest itself (i.e., the user won’t be able to display the bill segment that’s in error). This happens because the drill down causes the bill segment search logic to execute. This logic inhibits the selection of bill segments if the user can’t access the related account.

To minimize these anomalies, we recommend the following:

- Setup [To Do Roles](#) consistent with your Data Access Roles.
- Setup [Account Management Groups](#) that are consistent with your Access Groups.
- Setup default To Do Roles on your Account Management Groups for each [To Do type](#).

Restricted Transactions

The following table lists all transactions that have some type of account security. The following notation is used to describe the type of account security:

- **Account-oriented.** This notation is used if the respective transaction uses basic account security (i.e., the user must belong to at least one data access role that has access to the account's access group in order to see the information).
- **Person-oriented.** This notation is used if the respective transaction uses person-oriented account security. Refer to [Persons Can Also Be Secured](#) for more information.
- **Premise-oriented.** This notation is used if the respective transaction uses premise-oriented account security. Refer to [Premises Can Also Be Secured](#) for more information.
- None of the above. Some unusual transactions have unusual implementations of account security. These are described below.

Transaction	Type of Account Security
Account	Account-oriented
Account Bill / Payment History	Account-oriented
Account Financial History	Account-oriented
Account Interval Info	Account-oriented
Account Payment History	Account-oriented
Account Person Replicator	Account-oriented
Account SAs for Debt Class	Account-oriented
Adjustment	Account-oriented
Adjustment Calculation Line Characteristics	Account-oriented
Appointment	Premise-oriented
Bill	Account-oriented
Bill Print Group	Person-oriented
Bill Segment	Account-oriented
Billable Charge	Account-oriented
Budget Review	Account-oriented
Case	Account-oriented, Person-oriented and Premise-oriented
Collection Agency Referral	Account-oriented
Collection Process	Account-oriented
Contract Option	Account-oriented
Control Central	Account-oriented, Person-oriented and Premise-oriented

Transaction	Type of Account Security
Customer Contact	Person-oriented
Cut Process	Account-oriented
Declaration	Account-oriented
Deposit Review	Account-oriented
Field Activity	Premise-oriented
Field Order	Premise-oriented
Financial Transaction	Account-oriented
Financial Transaction on a Bill	Account-oriented
Financial Transaction on a Payment	Account-oriented
Interval Profile	Account-oriented (for SA-specific profiles)
Landlord Agreement	Account-oriented
Loan	Account-oriented
Match Event	Account-oriented
Multi-Cancel/Rebill	Account-oriented
Non-billed Budget	Account-oriented
Order	Account-oriented, Person-oriented and Premise-oriented
Overdue Process	Account-oriented
Pay Plan	Account-oriented
Payment	Account-oriented
Payment Arrangement	Account-oriented
Payment Arrangement for Bills	Account-oriented
Payment Event	The user must have access to ALL accounts linked to the payment event.
Payment Event Quick Add	The user must have access to ALL accounts linked to the payment event(s).
Payment Quick Add	Account-oriented
Payment / Tender Search	Account-oriented
Person	Person-oriented
Premise	Premise-oriented
Premise Management	Premise-oriented
Quote	Account-oriented
SA Billing History	Account-oriented
SA Cash Accounting Balance	Account-oriented
SA Financial History	Account-oriented
SA Relationship	Account-oriented
Service Agreement	Account-oriented
Service Credit Event	The user must have access to ALL accounts linked to the service

Transaction	Type of Account Security
	credit membership that has the service credit event.
Service Credit Membership	The user must have access to ALL accounts linked to the service credit membership.
Service Provider SA Relationship	Account-oriented
Severance Process	Account-oriented
Start/Stop	Account-oriented
Statement	The user can see the statement if they have access to at least one account on the statement's statement construct.
Statement Construct	The user can see the statement if they have access to at least one account on the statement construct.
Terms of Service	Account-oriented (User must have access to at least one account linked to the SA collection in the TOS.)
TOU Map	Account-oriented (for SA-specific TOU maps)
Umbrella Agreement	Account-oriented (User must have access to at least one account linked to the SA collection in the UA's TOS.)
Write Off	Account-oriented
Write Off Process	Account-oriented

Account Security Case Studies

The topics in this section contain examples of how to implement account security. Use these examples to form an intuitive understanding of these objects. Once this intuition is obtained, you'll be ready to design the account security objects for your own company.

Contents

[Securing Accounts Based On Customer Class](#)

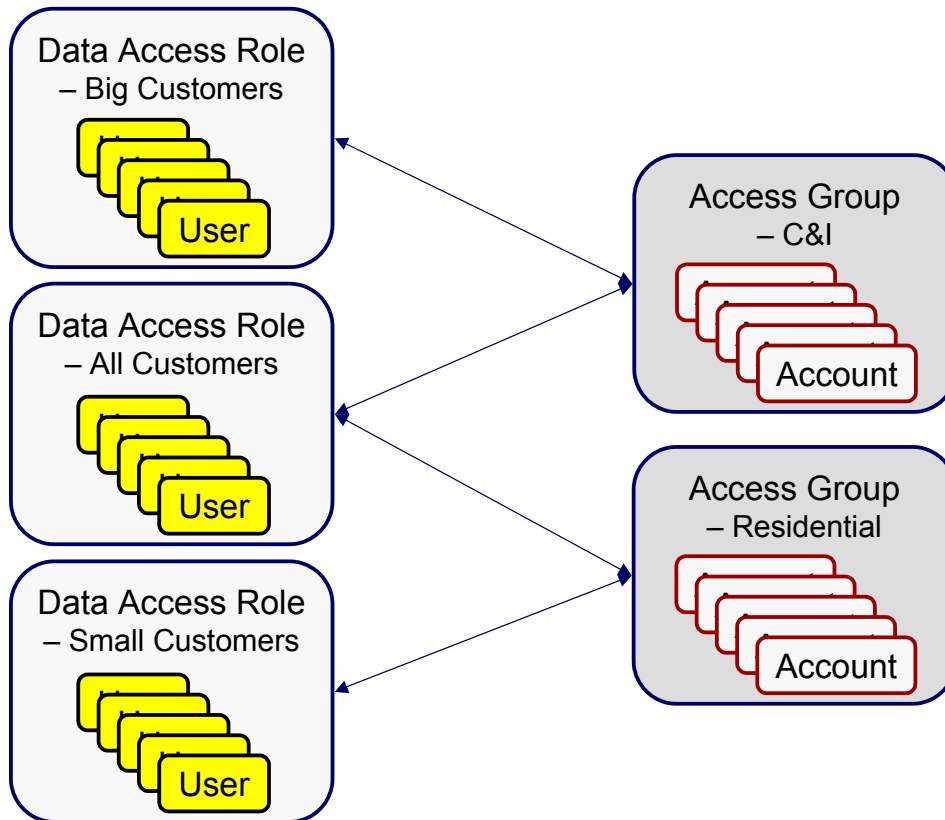
[Securing Accounts Based On Region](#)

Securing Accounts Based On Customer Class

Assume the following security requirement exists:

- You have two broad groups of accounts:
 - Residential accounts.
 - Commercial / Industrial accounts.
- Users can be classified as have one of the following access rights:
 - May access all accounts.
 - May only access residential accounts.
 - May only access commercial / industrial accounts.

The following diagram illustrates the access groups and data access roles required to implement these requirements:



Notice the following about the above:

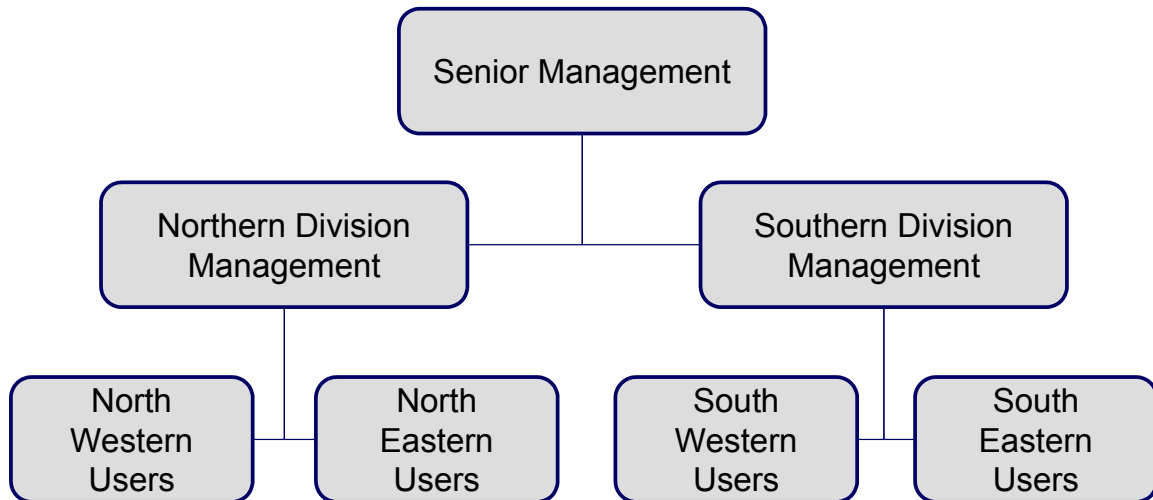
- There are 2 access groups because access to accounts is based on whether the account is considered to be residential or commercial/industrial.
- The Big Customers data access role is only linked to the C&I access group.
- The Small Customers data access role is only linked to the Residential access group.
- The All Customers access role is linked to both the C&I and Residential access groups. Users with this role can therefore access all accounts.

Securing Accounts Based On Region

Assume that accounts are classified as belonging to one of the following regions:

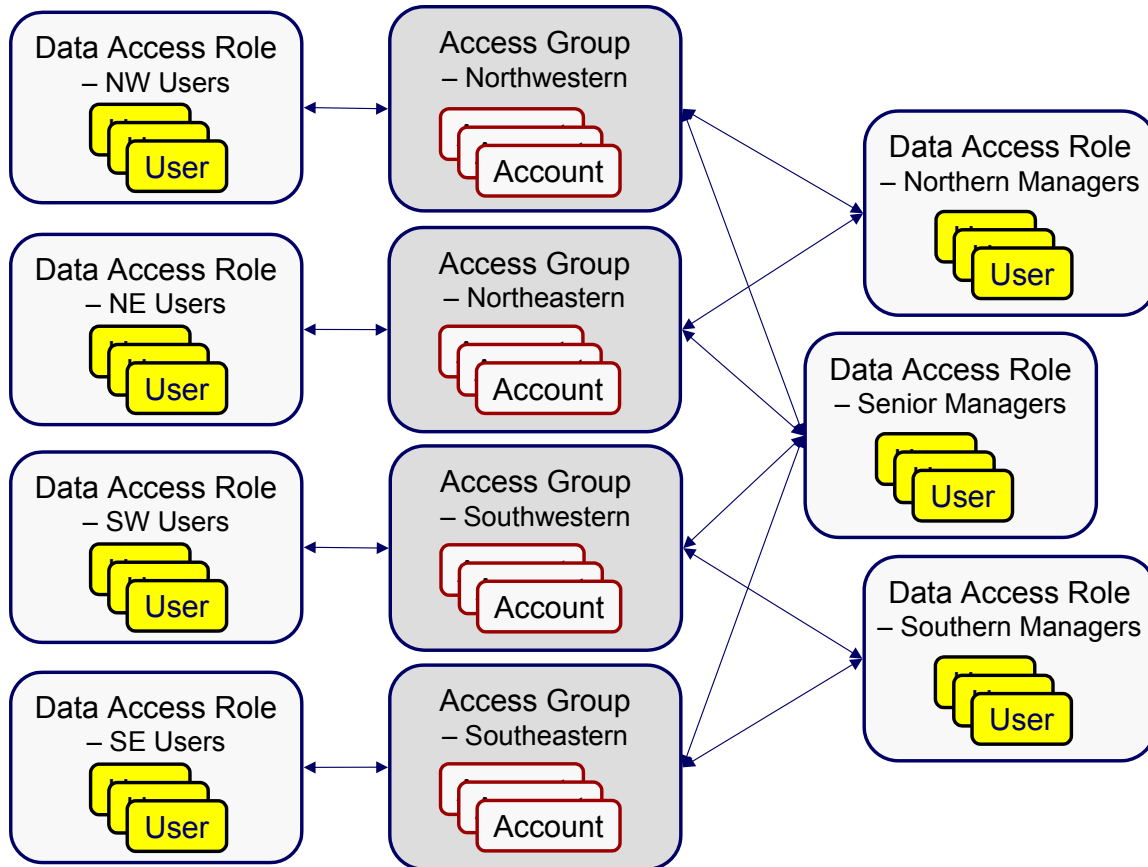
- Northwestern
- Northeastern
- Southwestern
- Southeastern

Assume the following company hierarchy exists:



- Senior Management has access to all customers
- Northern Division Management has access to all customers in the Northwestern and Northeastern divisions.
- Southern Division Management has access to all customers in the Southwestern and Southeastern divisions.
- Northwestern Users have access to all customers in the Northwestern division.
- Northeastern Users have access to all customers in the Northeastern division.
- Southwestern Users have access to all customers in the Southwestern division.
- Southeastern Users have access to all customers in the Southeastern division.

The following diagram illustrates the access groups and data access roles required to implement these requirements:



Notice the following about the above:

- There are 4 access groups because access to accounts is based on the region in which they are located (and there are 4 regions).
- There are 7 data access roles because each component of every layer of the access hierarchy requires a separate data access role.

The Default Access Group

There are two ways an access group can be assigned to an account:

- The base package will default an account's access group based on the user who adds the account. It uses the user's [default access group](#) to do this.
- If you can conceive of a rule to assign an appropriate access group to a newly added account, you can have your programming staff introduce a user exit to the account row program to implement this rule. For example, a user exit could be developed that assigns an access group to an account based on its customer class. The name of the user-exit is AFTER-MOVE-CORR-ADD and the name of the account row program is CIPCACCR.

Warning! Regardless of the method used to assign an account's access group, please be aware that the user who adds an account must have access to this access group.

Subsequent changes to an account's access group. A user may change an account's access group to any access group to which they have access.

If You Do Not Practice Account Security

If you do not restrict access to accounts (i.e., all users can access all accounts), you must set up one access group and one data access role and then indicate all users are part of this role. You should also define the access group as the default access group on all of your users (so that new accounts are all labeled with this access group).

Masking Sensitive Data

Refer to [Masking Data](#) for instructions describing how to configure the system to mask sensitive data like a customer's social security number or bank account number. If your implementation intends to mask any of the information that appears in the [Customer Information Zone](#) please navigate to this zone's documentation for special instructions.

Defining Overdue Processing Options

The system periodically monitors how much your customers owe to ensure they haven't violated your collection rules. When a violation is detected, the system initiates the appropriate activities (e.g., letters, disconnect notices, collection agency referrals, and eventually write off). The topics in this section describe how to configure the system to manage your overdue processing requirements.

Collecting on unpaid bills. The overdue processing module has been designed to collect on virtually anything from an unpaid bill to an unmatched financial transaction. You tell the system what you collect on by configuring the various overdue processing control tables. In this release, the base-package is delivered with algorithms that support collecting on overdue bills. If your organization practices balance-forward accounting (i.e., collection is based on overdue service agreement balances), you will not use this functionality. Rather, you will use the functionality described under [Defining Credit and Collections Options](#).

Straightforward rules = straightforward set up. Setting up this module is as challenging as your organization's collection rules. If you have simple rules, the set up process will be straightforward. If your rules are complicated (e.g., they differ based on the type of customer, the age of debt, the type of service, etc.), your setup process will be more challenging.

Contents

- [Case Study - Collecting On Overdue Bills](#)
- [How Does The Overdue Monitor Work?](#)
- [The Big Picture Of Overdue Processes](#)
- [Bill-Oriented Collection - Advanced Topics](#)
- [Creating Overdue and Cut Procedures](#)

Case Study - Collecting On Overdue Bills

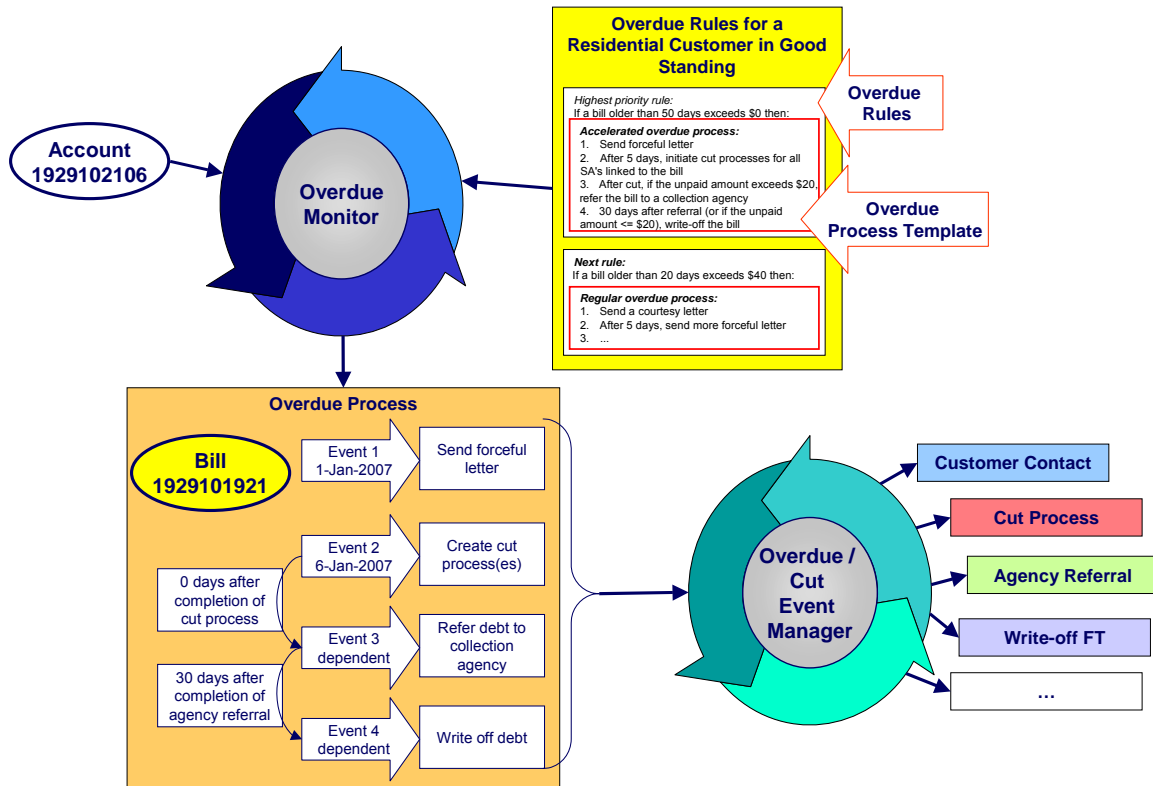
The following topics introduce a case study that describes how overdue processing can be used to collect on overdue bills. This is just an example as virtually every aspect of overdue processing is configurable. Use this case study to familiarize yourself with the overdue processing core concepts.

Contents

- [Monitoring Overdue Bills](#)
- [Cutting Service Agreements](#)

Monitoring Overdue Bills

The following diagram illustrates the objects and processes involved with collecting overdue bills.



There are many important concepts illustrated above:

The Overdue Monitor checks if your accounts have bills that violate your overdue rules

The [Overdue Monitor](#) is a background process that periodically reviews your account's financial obligations.

Note well: every account belongs to a collection class. There are two types of collection classes: those whose accounts are monitored by the [Account Debt Monitor](#), and those that are monitored by the Overdue Monitor. This chapter describes the Overdue Monitor.

Overdue rules define when and how unpaid bills are collected

An account's [collection class overdue rules](#) have algorithms that monitor an account's financial obligations. These algorithms are invoked by the Overdue Monitor when it's [time to review](#) an account's obligations.

These algorithms can contain any type of criteria. However, most are defined using a combination of a threshold age and monetary amount. For example, a classic algorithm would check if a bill has unpaid financial transactions more than 20 days old that exceed \$50.

In the case of bill-oriented collection, the monitoring algorithms look at each of the

account's bills to determine if they are paid. Note, a bill is considered paid if its financial transactions (FTs) are linked to a **balanced** match event. If a monitoring algorithm finds an unpaid bill, it can check if it old enough (and large enough) to be considered a violation.

When you set up a monitoring algorithm, you define the type of overdue process that should be created when an overdue bill is detected. You do this by defining the appropriate "overdue process template".

An overdue process template defines how to handle an overdue bill

An [overdue process template](#) contains one or more [overdue event types](#). These define the number and type of events that are created to prod the customer to pay. For example, you might set up an overdue process template with event types to send a series of letters followed up by a call.

The overdue process template has contains the rules defining [when events are activated](#).

The specific action that's performed by an overdue event is controlled by the **Activation** algorithm defined on its event type. Refer to [Overdue Event Type - Main](#) for a list of the various **Activation** algorithms delivered with the base package.

Multiple objects can be associated with a single process

The above diagram shows a single bill linked to an overdue process. It should be noted that an overdue process is capable of referencing multiple bills (or other objects).

Note well: while a single overdue process can reference many overdue objects, all such objects must be of the same type. For example, you cannot commingle bills and service agreements under a single overdue process. The type of object managed by an overdue process is defined on its [overdue process template](#).

If a customer pays the bill, the overdue process is cancelled

If an overdue bill is paid, the [overdue process is canceled](#) real-time. You control if and how an overdue process is cancelled by setting up the appropriate rules on the [overdue process template](#).

The Overdue / Cut Event Manager activates and triggers overdue events

The [Overdue / Cut Event Manager](#) is a background process that activates overdue events on the appropriate date. On this date, the event's **Activation** algorithm(s) are called.

This Overdue / Cut Event Manager also has

the responsibility of recursively activating later events that are dependent on the completion of earlier events.

Events can be activated real-time

[Overdue Process - Main](#) has a button that allows users to activate (and recursively trigger) overdue events online / real-time. This means you don't have to wait for a batch job to activate events.

Overdue events can wait for related activities to complete

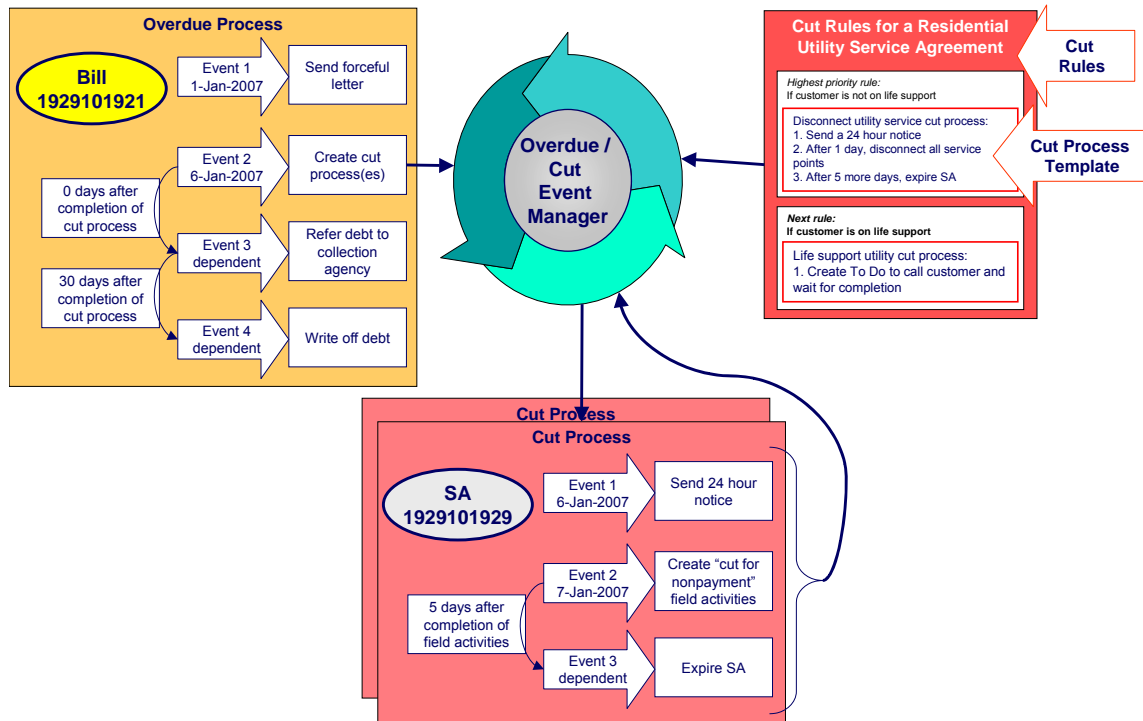
As described above, an overdue event's **Activation** algorithm can create virtually any object. What wasn't explained is that the event can be set up to [wait](#) for the ancillary object to finish before it completes. For example, an event can create a To Do entry and wait for it to complete before the next event is triggered. You can introduce plug-ins to create and wait on virtually any object.

While an overdue event is in the **Wait** state, the Overdue / Cut Event Manager monitors the state of the related object(s). When the related object completes, the event is transitioned to the Complete state (thus triggering dependent overdue events). Please see [Some Events Wait For Something Before Completing](#) for more information.

Cutting Service Agreements

An overdue process may contain an overdue event that creates a cut process(es). A cut process is used to cut (i.e., stop) a service agreement. The following diagram illustrates the objects and processes involved with cutting a service agreement.

Cut processes aren't required. It's quite conceivable to design an overdue process that doesn't cut service. For example, the overdue process may just contain an event that creates a case and the case manages the collection activities.



An overdue process may contain an overdue event that cuts (i.e., stops) service

If an overdue process's bill remains unpaid, one of the latter overdue events typically creates one or more [cut processes](#). A cut process contains the activities to stop a service agreement (in the hopes that lack of service will inspire the customer to pay). It should be noted that a separate cut process is created for each service agreement.

Cut rules define how to sever service agreements

The system allows you to define rules to control the type of cut process created. For example, you may have a different cut process if the customer has life support equipment, or if it's winter, or ...

The cut rules are defined on the service agreement's [SA type](#). This allows different rules for different types of service agreements.

Overdue events can "wait" for related activities to complete

When an overdue event creates cut process(es), it typically enters the **Wait** state. It enters this state as it is waiting for the cut process(es) to complete before it can itself complete. While in the **Wait** state, the Overdue / Cut Event Manager monitors the state of the related cut process(es). When the cut process(es) complete, the system transitions the originating overdue event to the **Complete** state (thus triggering dependent

A cut process template defines how to cut a service agreement

events).

Cut processes and events are created using a [cut process template](#). A cut process template defines the actions involved in cutting a given type of service agreement. A cut process template usually contains several cut events. These events are a series of letters and / or disconnection field activities that eventually result in the expiration of a service agreement if payment is not received.

An **Activation** algorithm controls the specific action associated with a cut event and therefore an event can do practically anything. Refer to [Cut Event Type - Main](#) for a list of the various **Activation** algorithms delivered with the base package.

Cut events are also managed by the Overdue / Cut Event Manager

The same [background process](#) that manages the activation of overdue events manages cut events. This means you only have to submit one batch job to activate and trigger both overdue and cut events.

Events can be activated real-time

[Cut Process - Main](#) has a button that allows users to activate (and recursively trigger) cut events online / real-time. This means you don't have to wait for a batch job to activate events.

Cut events can "wait" just like overdue events

Cut events support the ability to wait for something to complete just like overdue events. For example, if a cut event creates a field activity, it enters the **Wait** state. While in the **Wait** state, the Overdue / Cut Event Manager monitors the state of the related field activities. When the field activities complete, the system transitions the originating cut event to the **Complete** state.

And, just like for overdue events, the notion of a cut event creating something and then waiting for it to complete is not limited to field activities. For example, you could have a cut event create a To Do entry and wait for it to complete before the next event is triggered.

Please see [Some Events Wait For Something Before Completing](#) for more information.

After a service agreement is stopped, it will be final billed

Many cut processes are configured so their last cut event [expires the service agreement](#). What happens at expiration depends on the type of service agreement. However, eventually the service agreement is transitioned into the **Stopped** state.

If a customer doesn't pay their final bill, the final bill will be processed using a different type of overdue process

When the last active service agreement linked to an account is stopped, the system changes the account's bill cycle to bill that evening. If only one of many SA's is stopped, the SA will only be final billed as per the account's original bill cycle schedule.

Final bills only differ from ongoing bills in that the service agreements associated with the bill's FTs are not active. This means that there is nothing to cut. This means that the type of overdue process used to handle a "final" bill should differ from that used to handle "ongoing" bills.

As described above, overdue rules define how unpaid bills are handled (both ongoing and final). We recommend configuring your overdue rules to use different overdue process templates for final versus ongoing bills.

The last overdue event typically causes the debt to be written off

Ultimately, if the overdue and cut events fail to inspire payment, the debt will be written off. The method used to write-off a bill is controlled by an overdue event **Activation** algorithm.

How Does The Overdue Monitor Work?

This section describes how the Overdue Monitor background process (batch control: **C1-ADMOV**) uses your overdue rules to collect overdue debt.

Recommendation. We recommend that you familiarize yourself with the concepts described in the [case studies](#) before reading this section.

Collecting overdue bills. The examples in the following section frequently refer to how the Overdue Monitor is configured for an organization that collects on unpaid bills. It should be noted that these are just examples as the Overdue Monitor can be used to collect on virtually anything (if you create the appropriate plug-in algorithms).

Contents

- [Different Overdue Rules For Different Customers](#)
- [Overdue Rules Are Embodied In Algorithms](#)
- [When Is An Account Monitored?](#)
- [Collection Class Defines If And How Accounts Are Monitored](#)

Different Overdue Rules For Different Customers

The Overdue Monitor uses rules to control how it monitors an account's debt. The system allows you to define different rules for different combinations of collection class, division and currency code. For example,

- You probably have different collection rules for different jurisdictions (i.e., CIS Divisions). For example, if you have customers in different states / provinces, you may have different rules applied to each jurisdiction's debt. The **CIS division on each customer's account** defines the jurisdictional rules applied to the account's debt.
- You probably have different collection rules for different customer segments. For example, bills for large customers might be overdue if they are more than 10 days late, whereas small customers might have 24 days. You differentiate your customers in respect of the overdue via the **collection class on the customers' accounts**. An account's initial collection class is defaulted from its customer class. You may override an account's collection class at will.
- You will have different criteria for every currency in which you work because the monitoring process always compares a customer's debt against some value and this value must be denominated in the customer's currency. A customer's currency is defined using a **currency code on the account**.

The above means that every combination of CIS division, collection class, and currency code can have different rules. The following matrix is used to illustrate a sample organization's rules (note, we assume a single currency and therefore avoid the third dimension):

Account's Collection Class → Account's Division ↓	Commercial Customer	Residential Customer
North	<p>Highest Priority: If a bill exists with unpaid FT's > \$0 that is older than 45 days, create the commercial 45 days late overdue process.</p> <p>Next Priority: If a bill exists with unpaid FT's > \$100 that is older than 30 days, create the commercial 30 days late overdue process.</p>	<p>Highest Priority: If a bill exists with unpaid FT's > \$0 that is older than 50 days, create the accelerated overdue process for residential customers.</p> <p>Lower Priority: If a bill exists with unpaid FT's > \$25 that is older than 25 days, create the courtesy reminder overdue process for residential customers.</p>
South

Notice that there can be multiple criteria for each cell in the matrix. What differentiates one criteria from another is its priority. The higher priority criteria will be compared first. If the debt violates the criteria, the overdue process is initiated and no further comparisons are performed.

Overdue Rules Are Embodied In Algorithms

Your organization's overdue rules are defined in algorithms plugged in on [Collection Class Overdue Rules](#) (in the **Overdue Monitor Rule** system event). When the Overdue Monitor analyzes an account, it uses the rules associated with the account's collection class, CIS division and currency code. To analyze an account, it simply invokes these algorithms in sequence order, i.e., the lower the sequence, the higher its priority.

An **Overdue Monitor Rule** algorithm has two responsibilities:

- it determines if an account violates its overdue rules,
- if so, it creates one or more overdue processes using an [overdue process template](#)

When Is An Account Monitored?

The Overdue Monitor determines if an account violates your overdue rules at least every X days, where X is defined on the [Customer Class - Controls](#) associated with the account's customer class and division (in the field Min Credit Review Freq (Days)).

In addition, the Overdue Monitor examines an account's debt as follows:

- X days after an account's bill due date (X is defined in the field Credit Review Grace Days on [customer class control](#)).
- If a payment is canceled with a cancellation reason that indicates non-sufficient funds.
- If a payment arrangement is broken (assuming you use the base package's break payment arrangement plug-in). Refer to [Breaking A Bill Oriented Payment Arrangement](#) for more information.
- If a pay plan is broken. Refer to [The Pay Plan Monitor](#) for more information.
- If a [match event](#) is added, changed or deleted.
- When a written off bill is [reversed](#).

Additional events. Your implementation can have other events trigger the analysis of an account by the Overdue Monitor. To do this, add logic to insert a row on the CI_ADM_RVW_SCH table when the event occurs. This row simply references the account ID to be reviewed and the desired review date.

Collection Class Defines If And How Accounts Are Monitored

As described above, every account references a collection class. The collection class defines if and how its accounts are monitored. There are three options:

- The accounts are monitored by the [Account Debt Monitor](#)
- The accounts are monitored by the Overdue Monitor (this is described in this chapter).
- The accounts are not monitored for overdue debt.

Migration. If you plan to migrate from the Account Debt Monitor (ADM) method of collection to the Overdue Monitor method, a special Overdue Monitor algorithm ([C1-ACT-CSW](#)) has been supplied that will skip accounts that are eligible for review by the Overdue Monitor if they have an active collection, severance or write-off process. This algorithm should be plugged in the applicable Collection Class Overdue Rules so that it will be invoked first. This allows accounts with active ADM-oriented collection activities to work themselves through the system. When an account no longer has any active ADM-oriented activities, monitoring responsibilities will be assumed by the Overdue Monitor.

The Big Picture Of Overdue Processes

As described above, the Overdue Monitor subjects your accounts to overdue rules. If a rule is violated, an overdue process is created. The topics in this section provide background information about overdue processes.

Contents

- [How Are Overdue Processes Created?](#)
- [The Components Of An Overdue Process](#)
- [Experimenting With Alternative Overdue Process Templates](#)
- [Overdue Process Information Is Overridable](#)
- [Original and Unpaid Amounts](#)
- [Overdue Processes Are Highlighted Elsewhere](#)
- [How Are Overdue Processes Cancelled?](#)
- [Overdue Processes Are Created From Templates](#)
- [The Big Picture Of Overdue Events](#)
- [The Big Picture Of Cut Processes](#)
- [Cut Events Are Like Overdue Events](#)
- [Write Offs Are Implemented Using Overdue Events](#)
- [Calendar vs. Work Days](#)

How Are Overdue Processes Created?

As described [above](#), the system creates an overdue process when an account violates your overdue rules. In addition, a user can manually create an overdue process at their discretion.

The Components Of An Overdue Process

The following topics describe the major components of an overdue process.

Contents

- [Overdue Objects](#)
- [Overdue Events](#)
- [Overdue Log](#)

Overdue Objects

When an overdue process is created, the system links the overdue object(s) to the process. For example, if an overdue bill is detected, the bill is linked to the overdue process.

When you set up an [overdue process template](#), you define the type of object it collects on by defining the [foreign key characteristic type](#) used to reference the object. For example, when you set up an overdue process template to collect on bills, you define a foreign key characteristic type that references the bill object.

Overdue Events

An overdue process has one or more overdue events. These events are the actions designed to encourage the customer to pay. For example, you might set up overdue events that:

- Send letters (via the creation of a customer contact)
- Create To Do entries

- Impact the account's credit rating
- Create a case (e.g., to seize the customer's assets)
- ... (the list is only limited by your imagination as algorithms are used to perform the event's actions)

You define the number and type of events by configuring [overdue process templates](#). When the system creates an overdue process, it copies the events defined on the specified template.

It's important to note that all overdue events are created when the overdue process is created. A separate background process, the [Overdue / Cut Event Manager](#), is responsible for activating, monitoring, and triggering overdue events. Activation of an event causes the system to do whatever the event indicates (e.g., send a letter, send a To Do entry to a user, start a cut process, refer debt to a collection agency, write-off debt, etc.).

Overdue Log

Every overdue process has a log holding its history. Entries are added to the log when meaningful events occur, for example:

- When the process is created, a log entry is created to describe why the process was started.
- When an overdue event is activated, a log entry is created. These entries frequently contain a foreign key to the object that the event created so that users can easily drill down to the object from the log. For example, if an event creates a To Do entry, the To Do entry's foreign key is placed on the log and this allows a user to drill down on the log entry to see the To Do entry.
- When a process is canceled, a log entry is created to describe the circumstances of the cancellation (e.g., manual versus automated).
- Users can manually add log entries (you might want to think of these as "diary" entries) as desired.
- ...

Many of the base-package algorithms involved in overdue processing insert log entries so that a thorough audit trail is maintained. These algorithms have been designed to allow you to control the verbiage in each log entry by defining the desired message number using an algorithm parameter.

The log is viewable on the [Overdue Process - Log](#) page.

More than just an audit trail. Please note that the log entries are more than just an audit trail. The system makes use of the log entries to know what it did. For example, when an overdue event needs to monitor the state of the To Do entries that it created, it uses the log to determine the identity of these To Do entries.

Experimenting With Alternative Overdue Process Templates

The system allows you to determine the efficacy of proposed overdue process templates using a small subset of customers before implementing the templates on the entire customer base. We use the term "champion / challenger" to reference this functionality.

We'll use an example to explain. Let's assume your prevailing overdue process template for residential customers starts with a "gentle reminder" letter followed 10 days later by a letter threatening collection agency referral if payment is not received. You may want to experiment with the impact of a change to this template. For example, you may want to change the "gentle reminder" to something more assertive and follow this up 5 days later with an even sterner warning. You can use the "champion / challenger" functionality to perform this experiment.

The following points describe how to implement "champion / challenger" functionality:

- Set up a "challenger" overdue process template for each template that you want to experiment with.
- Insert a new **Champion/Challenger** option on the [Overdue Processing Feature Configuration](#) for every champion template. Each option's value defines:
 - the "champion" overdue process template code
 - the "challenger" overdue process template code
 - the percentage of the time the system should use the "challenger" template

Keep in mind that you can only experiment with one challenger template per champion template. For example, let's assume you have two prevailing overdue process templates - one for residential customers and another for commercial customers. You can experiment with different challenger templates for the residential and commercial templates. However, you cannot experiment with two different challenger templates for the residential champion template (i.e., a champion template can have 0 or 1 challenger template).

After setting up the above, your implementation's [Overdue Rule Plug-In](#) may include logic to use the challenger template X% of the time rather than the champion template. The sample plug-in provided in the base product, called [C1-CB-CR-RAT](#), includes this logic.

If you are using the Oracle Utilities Business Intelligence product, you can configure analytic zones in innumerable ways to compare the efficacy of the champion versus the challenger. For example,

- You can set up a graph to show the average duration of each type of process.
- You can set up a graph to show the average dollars that were successfully collected.
- You can set up a dimensional scorecard to show how each template performed in different regions (or customer classes or ...).
- Etc (the list is limited by your imagination)

Overdue Process Information Is Overridable

"Overdue process info" is the concatenated string of information that summarizes an overdue process throughout the user interface. The base-package logic constructs this string by concatenating the following information:

- The description of its overdue process template
- Its status
- For **active** processes, the number of days since it was created. For **inactive** processes, the number of days since it was inactivated.
- For **active** processes, the unpaid amount of the objects being collected

If you'd prefer a different info string, you can develop a new algorithm and plug-it in on your [overdue process templates](#). This design allows some / all overdue process templates to have an override info string.

Original and Unpaid Amounts

There are two amounts associated with each overdue object linked to an overdue process: its Original Amount and its Unpaid Amount. These amounts are used throughout overdue processing.

You control how these amounts are calculated by defining the appropriate algorithm on your [overdue process templates](#). For example, you can plug in a base-package algorithm ([C1-CUAOA](#)) if you collect on overdue bills and the original amount is the amount of financial transactions linked to the bill, and the unpaid amount is the sum of financial transactions that are not linked to **balanced** match events.

Overdue Processes Are Highlighted Elsewhere

The topics in this section describe how other parts of the system highlight the existence of overdue and cut processes.

Contents

- [Alert Zone](#)
- [Credit and Collections Zone](#)
- [Account Activity History Zone](#)

Alert Zone

If you plug-in the appropriate alert algorithm ([C1-OD-PROC](#)) on the Installation record, alert(s) will be shown for active overdue processes in the Alert Zone that appears in the Dashboard and on Control Central - Account Information.

Credit and Collections Zone

The Credit and Collections zone on [Control Central - Account Information](#) shows **active** overdue and cut processes.

Account Activity History Zone

The Account Activity History Zone on [Control Central - Account Information](#) shows pending and **waiting** events and **inactive** processes.

How Are Overdue Processes Cancelled?

A user may cancel an overdue process at their discretion, online / real-time using [Overdue Process - Main](#).

The system will automatically cancel an overdue process when the object(s) associated with the overdue process are sufficiently paid. Exactly when the system checks if an overdue process should be cancelled is dependent on your organization's billing and accounting rules. For example, if you practice [open-item accounting](#), you'd want to analyze an account's active overdue processes whenever a match event is added, changed or deleted (as match events are the only objects that impact if debt is considered paid in an open-item world). The base-package supports this specific example. If you need additional events to check if an overdue process should be canceled (e.g., the creation of a pay plan), a base-package change MAY be necessary. Please check with customer support if you have questions.

Two algorithms plugged-in on the [overdue process template](#) handle the cancellation:

- The **Cancel Criteria** algorithm is responsible for determining if an overdue process should be canceled. Algorithms of this type analyze the outstanding debt on the objects linked to the overdue process and indicate whether a process can be cancelled.
- The **Cancel Logic** algorithm is responsible for actually canceling the process. The logic involved in cancellation can be quite sophisticated as canceling an overdue process can result in the cancellation of its pending events and cut processes.

Why two algorithms? The reason two algorithms are involved in cancellation is because we want the cancellation logic to be encapsulated in one place so it can be called during both manual and automated cancellation.

Different logic for different templates. Because both the **Cancel Criteria** and **Cancel Logic** algorithms are plugged-in on the overdue process's template, you can have different cancellation criteria and logic for different templates.

Overdue Processes Are Created From Templates

As described above, you set up [overdue process templates](#) to define the types of events and when they are executed. When an overdue process is created, its events are created by copying the event types from an overdue process template. The remaining topics in this section provide background information to assist you in setting up your templates.

The Big Picture Of Overdue Events

This section describes the various types of overdue events and their lifecycle.

Contents

- [How Are Overdue Events Created?](#)
- [Overdue Events Can Do Many Things](#)
- [Overdue Event Information Is Overridable](#)
- [Overdue Event Lifecycle](#)

How Are Overdue Events Created?

Overdue events are created as follows:

- The [Overdue Monitor](#) invokes **Overdue Monitor Rules** to periodically check your accounts (refer to [Overdue Rules Are Embodied In Algorithms](#) for how this works). An **Overdue Monitor Rule** creates an overdue process when an account violates your overdue rules. The overdue process has one or more overdue event(s). The number and type of events is controlled by the overdue process template specified on the **Overdue Monitor Rule**.
- Users can create an overdue process manually on [Overdue Process - Main](#). To do this, they specify an overdue process template. The number and type of overdue events is defaulted from the template.
- An overdue event may be manually added to an existing overdue process by a user on [Overdue Process - Events](#).

Bottom line. Most overdue events are created by the system when it creates an overdue process for delinquent obligations. If you need to create an ad hoc overdue event, you can either create an overdue process whose template contains the desired event OR link the desired event to an existing process.

Overdue Events Can Do Many Things

An overdue event can perform a wide number of activities as the logic is embodied in an algorithm. The following points describe how this works:

- Every overdue event references an [overdue event type](#).
- The overdue event type, in turn, references an **Event Activation** algorithm.
- The **Event Activation** algorithm is invoked when the event is [triggered](#).

Overdue Event Information Is Overridable

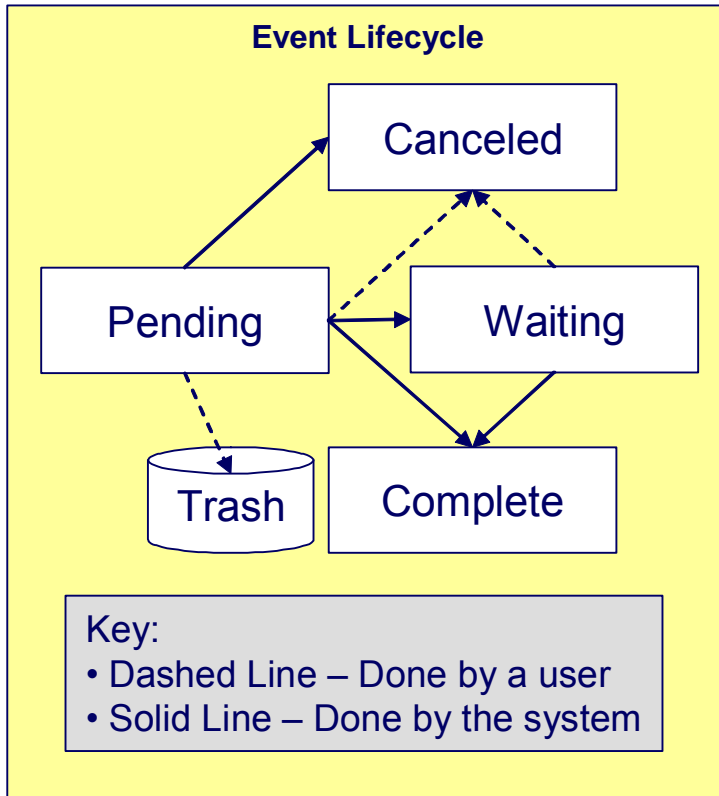
“Overdue event info” is the concatenated string of information that summarizes an overdue event throughout the system. The base-package logic constructs this string by concatenating the following information:

- The event type's description
- The event's status
- If it's **pending**:
 - If the event has a trigger date, the number of days until it's triggered plus the verbiage **day(s) from today**
 - Otherwise, the verbiage **dependent on other events**
- If it's **waiting**, the number of days, hours and minutes that it's been waiting
- If it's **canceled**, the cancel reason code's description
- If it's **complete**, the number of days, hours and minutes that it's been complete

If you'd prefer a different info string, you can develop a new algorithm and plug-it in on your event types. This design allows some / all event types to have an override info string.

Overdue Event Lifecycle

The following diagram shows the possible lifecycle of an overdue event:



Overdue events are initially created in the **Pending** state. An event can take myriad paths after it's created; it all depends on how you've configured the system. The following topics describe an event's lifecycle:

Contents

- [How and When Events Are Activated](#)
- [Activating Events Should Add A Log Entry](#)
- [Holding Events](#)
- [Some Events Wait For Something Before They Activate](#)
- [How Are Events Canceled](#)

How and When Events Are Activated

An overdue event contains the date it should be activated; this is referred to as its trigger date. On this date, the Overdue / Cut Event Manager (a background process (**C1-ODET**)) invokes the **Event Activation** algorithm plugged-in on the event's event type. The **Event Activation** algorithm, in turn, will decide on the state in which to leave the overdue event (e.g., it may transition it to the **Complete** state or the **Waiting** state).

If a user can't wait for the Overdue / Cut Event Manager real-time, they can click a button on [Overdue Process - Main](#) to activate (and recursively trigger) events online / real-time.

You control how an event's trigger date is populated by configuring the [overdue process template](#). You are given two choices when you link an event type to an overdue process template:

- You can indicate the event should be assigned a trigger date when it is first created. You'd use this approach on the first event and events with no dependencies on earlier events. The following points describe how to configure the overdue process template to do this:
 - Indicate the event type is NOT dependent on other events, and
 - Define the number of days after the process's creation to use when calculating the trigger date.
- You can indicate the event should be assigned a trigger date only after earlier events are **Complete**. This technique should be used whenever you have an event that is only executed after other events are **Complete**. The following points describe how to configure the overdue process template to do this:
 - Indicate the event is dependent on other events, and
 - Define the number of days after the completion / cancellation of all dependent event(s) that the trigger date should be set to. The Overdue / Cut Event Manager sets the trigger date on such an event when it detects that all of its dependent events are complete / canceled.

Calendar vs Workdays. When an overdue event is created by the system, its trigger date is set in accordance with your date arithmetic preferences. Refer to [Calendar vs. Work Days](#) for more information.

Activating Events Should Add A Log Entry

As described [above](#), an overdue process has a log holding a history of meaningful events in the process's life. Most **Event Activation** algorithms will add an entry to the process's log.

These log entries are more than just an audit trail as they also reference the objects that are created during activation. For example, if an activation algorithm creates a customer contact, the ID of the customer contact will be referenced on the log (and end-users will be able to drill down on the log entry to see the customer contact).

Holding Events

You can prevent a **pending** event with a trigger date on / before the current date from activating by plugging-in a **Hold Event Activation** plug-in on the overdue process template. This might prove useful, for example, if you want to suspend an overdue process while a [case](#) used to investigate the assertion of a customer is outstanding. Then, when the case closes, the overdue process can start up where it left off.

This technique would prove useful, for example, if your users can grant ad hoc suspend periods (e.g., if a customer wants a few extra days to pay before a cutoff). To do this, create a case type that has two states: Open and Close. Make sure to set up the Open state to have an automatic transition algorithm to close the case after X days. Then, all a user has to do is create a case of this type when they want to provide a suspend period. When the suspend period is over and payment isn't received, the case will close and the overdue process will start up where it left off.

Some Events Wait For Something Before They Activate

Consider this scenario - you want an overdue event to create a To Do entry so a user can authorize the next phase of an overdue process. When this event activates, the event's activation algorithm will create a To Do entry, but it will NOT transition the event to **complete**. Rather, the overdue event will exist in the **waiting** state. While in the **waiting** state, the Overdue / Cut Event Manager will monitor the state of the To Do entry. When the To Do entry completes, the original overdue event can transition to the **complete** state and then latter dependent events can be triggered. The following points describe how to configure the system to support this type of event:

- The event type's **Event Activation** algorithm should behave as follows:
 - It creates the object on which the overdue event waits.
 - It must link this object to the overdue process by creating a log entry where the prime-key of the related object is referenced (in a foreign-key characteristic). This log entry should also reference the event.
 - It should leave the overdue event in the **waiting** state.
- The event type must have a **Monitor Waiting Event** algorithm. This algorithm is invoked each time the [Overdue / Cut Event Manager](#) executes. If the related object has transitioned to a "final" state, the originating overdue event is transitioned to the **complete** state (and then latter dependent events are triggered).

Bottom line. Two algorithms must be set up on an overdue event type to implement waiting functionality: an Event Activation algorithm that creates the monitored object and a **Monitor Waiting Event** algorithm to check on the state of the monitored object. The Overdue / Cut Event Manager has the dual responsibility of activating the event and monitoring its related object for completion (and then triggering the dependent events when it completes).

While the above example illustrated how an overdue event could create and then monitor a To Do entry, you can use this functionality to create and monitor any object that has an initial and final state. If the base package does not contain the algorithms you need, simply develop new ones using the base-package algorithms as examples.

How Are Events Canceled

A **pending** event will be **cancelled** automatically by the system when the overdue process is canceled. Refer to [How Are Overdue Processes Cancelled](#) for more information.

A user may cancel a **pending** or **waiting** event at their discretion.

Regardless of what triggers the cancellation, the **Cancel Logic** algorithm plugged in on the overdue event type handles the cancellation. This allows you to introduce additional cancellation logic should the need arise. Please note that the base package cancel algorithms insert a [log entry](#) when a user manually cancels an event.

The Big Picture Of Cut Processes

While not required, many overdue processes will contain one or more events that will cut (i.e., stop) service. When such an event activates, it creates a "cut process". A cut process is very similar to an overdue process; the major difference is that a cut process is linked to a specific service agreement (the one being cut), whereas an overdue process is linked to the object(s) in arrears.

The topics in this section provide background information about cut processes.

Contents

- [How Are Cut Processes Created?](#)
- [Overdue Events Wait For The Cut Process To Conclude](#)
- [Cut Processes Exist Under An Overdue Process](#)
- [The Components Of A Cut Process](#)
- [Cut Process Information Is Overridable](#)
- [How Are Cut Processes Cancelled?](#)
- [Cut Processes and Events Are Created From Templates](#)

How Are Cut Processes Created?

The activation of an overdue event can create one or more cut processes.

In addition, a user can manually create a cut process at their discretion using [Cut Process - Main](#).

Overdue Events Wait For The Cut Process To Conclude

Overdue events that create cut processes are perfect examples of events that [wait](#) for the object they create to complete before they, in turn, **complete**. After the cut process concludes, the originating overdue event will complete thus triggering its dependent events.

Cut Processes Exist Under An Overdue Process

It's important to note that a cut process exists in respect of an overdue process. In other words, you can't create a cut process without referencing a parent overdue process. There are two reasons why:

- A cut process's overdue process defines the objects in arrears. It is these objects that are monitored to determine if the cut process (and overdue process) can be cancelled.
- The [log](#) associated with a cut process's overdue process contains the history of the cut process(es) and their events; there is no log specific to a cut process.

The Components Of A Cut Process

Cut processes are simpler than overdue processes as they simply contain one or more "cut events". These events are the service agreement-specific actions designed to encourage the customer to pay. For example, you might set up cut events that:

- Create [field activities to cut service](#)
- Send letters (via the creation of a customer contact)
- Create To Do entries
- Impact the account's credit rating
- Create a case (e.g., to manage a customer with life support issues)
- ... (the list is only limited by your imagination as algorithms are used to perform the event's actions)

You define the number and type of events by configuring [cut process templates](#). When the system creates a cut process, it copies the events defined on the related template.

It's important to note that all cut events are created when the cut process is created. The Overdue / Cut Event Manager is responsible for activating the cut events at the appropriate time. Activation of an event causes the system to do whatever the event indicates (e.g., send a letter, send a ToDo entry to a user, create a field activity, etc.)

Cut processes do not have their own log as their history is maintained on the parent overdue process's [log](#).

Cut Process Information Is Overridable

"Cut process info" is the concatenated string of information that summarizes a cut process throughout the system. The base-package logic constructs this string by concatenating the following information:

- Its cut process template's description
- Its status
- For **active** processes, the number of days since it was created. For **inactive** processes, the number of days since it was inactivated.
- For **active** processes, the unpaid amount of the objects being collected on the cut process's overdue process

If you'd prefer a different info string, you can develop a new algorithm and plug-it in on your cut process templates. This design allows some / all cut process templates to have an override info string.

How Are Cut Processes Cancelled?

A user may cancel a cut process at their discretion, online / real-time using [Cut Process - Main](#).

The system will automatically cancel a cut process when the overdue process is cancelled.

Regardless of what triggers the cancellation, the **Cancel Cut Process** algorithm plugged in on the cut process's [template](#) handles the cancellation.

If a cut process is canceled and the cut process's events created field activities, the cancel logic embodied in the algorithm can be quite sophisticated. Refer to [Field Activities To Cut And Reconnect Service](#) for an example.

Cut Processes and Events Are Created From Templates

When a cut process is created, its cut events are created by copying a cut process template's event types. Cut events follow the same lifecycle and possess the same behavior as [overdue events](#).

Cut Events Are Like Overdue Events

Cut events almost identical to [overdue events](#). The major difference is that cut events exists under a cut process and therefore are oriented towards the cut process's service agreement. The following topics describe other differences between cut events and overdue events.

Contents

[Field Activities To Cut and Reconnect Service](#)

[A Cut Event Expires Service Agreements \(and May Trigger Final Billing\)](#)

Field Activities To Cut and Reconnect Service

A cut process that's created for a service agreement whose service can be severed will typically contain a cut event that creates field activity(s) to cut service. The base-package cut event activation algorithm will create field activities for the service points linked to the service agreement being cut. The type of activity is defined on the [field activity type profile](#) defined on each service point's SP type. Please see [C1-CE-CR-FA](#) for the exact details of the base-package algorithm.

If the cut process is canceled after field activity(s) are created and before the cut process completes, different activities can transpire. The specifics of what happens are controlled by a **Cancel Logic** algorithm plugged in on the [cut process template](#). The following points summarize how a base-package algorithm ([C1-CP-DWFA](#)) works:

- It assembles all **Pending**, **Held** and **Complete** cut for nonpayment field activities (FA) created by the cut process (these are defined in the overdue process's [log](#)).
- Each FA is processed as follows:
 - If the FA is **Pending** or **Held** and it is not linked to a field order (i.e., it hasn't been dispatched):
 - The FA is cancelled
 - An entry is added to the overdue process log to indicate the cancellation
 - If the FA is **Pending** or **Held** and it is linked to a field order (i.e., it has been dispatched):
 - A To Do entry is created (the To Do Type is defined on the algorithm)
 - If the FA is **Complete** and the Cut Process's SA is **pending start** or **active** (i.e., it hasn't been expired yet):
 - Reconnect field activity(s) are created for every service point that had a cut for non-payment field activity. The type of activity is defined on the [field activity type profile](#) defined on each service point's SP type.

A Cut Event Expires Service Agreements (and May Trigger Final Billing)

The last cut event is typically set up with an activation algorithm that expires the cut process's service agreement. If earlier cut events created "cut for non-payment" field activities, these field activities will be used as the basis for stopping service. Refer to [Finalizing Pending Stops](#) for how the system use the meter reads on these field activities as the "stop reads" on the service agreement. Note, you can see the field activities that are used to "cut" and "stop" service by viewing the Field Activities grid on [Service Agreement - Service Point](#).

Final billing. When the last service agreement linked to an account is expired, the system will schedule the account for billing (outside of its normal bill cycle schedule).

Write Offs Are Implemented Using Overdue Events

The system has been designed to allow overdue events on the original overdue process to write-off the objects being collected. Refer to [Writing Off Bills](#) for a case-study explaining how to do this.

Calendar vs. Work Days

When you set up your overdue and cut process templates, you supply information that controls how each event's trigger date is calculated. You have two options:

- You can say that an event's trigger date can only be populated after earlier, dependent events are complete. For example, the 2nd event (cut service) is triggered 2 days after the 1st event is complete (48 hour warning letter).
- You can say that an event's trigger date is populated when the process is first created. You simply define the number of days after the start of the process when each such event should be triggered. For example, the 2nd event (send cutoff warning) can be triggered 7 days after the start of the process.

In addition to the above, an option defined on the [Feature Configuration for Overdue Processing](#) plays a part in the calculation of an event's trigger date:

- If you set the option to use **calendar days**, the trigger date of events will be set to the first workday on / after the calculated date. For example, if you indicate that the 2nd event is triggered 7 days after the 1st event, the system will add 7 days to the 1st event's completion date. It then checks if this is a workday (and not a holiday); if so, this is the trigger date of the event; if not, it assigns the trigger date to the next workday.
- If you set the option to use **workdays**, the trigger date will be calculated by counting workdays. For example, if you indicate that the 2nd event is triggered 7 days after the 1st event, the system will count 7 workdays and set the trigger date accordingly.

Account's division controls the work calendar. The system uses the above information in conjunction with the [work calendar](#) associated with the account's division to determine workdays.

Bill-Oriented Collection - Advanced Topics

The topics in this section provide information on features designed to facilitate the collection of overdue bills.

Contents

- [Miscellaneous Bill-Oriented Collection Topics](#)
- [Writing Off Bills](#)
- [Bill-Oriented Payment Arrangements](#)

Miscellaneous Bill-Oriented Collection Topics

The topics in this section provide background information about a variety of bill-oriented collection topics.

Contents

- [Highlights Of The Sample Overdue Monitor Rule Algorithm](#)
- [Holding A Process's Events](#)
- [Bill Info Shows Unpaid and Write-off Amounts](#)

Highlights Of The Sample Overdue Monitor Rule Algorithm

A base-package **Overdue Monitor Rule** algorithm exists to support many different types of overdue rules (see [C1-CB-CR-RAT](#)). Keep in mind that multiple such algorithms can be plugged in on [Collection Class Overdue Rules](#). When the [Overdue Monitor](#) analyzes an account, it calls these algorithms until there are no more to invoke (or until an algorithm tells it that there is nothing more to check). The following points describe its various options:

- **Different rules based on credit ratings.** You can set up algorithms to only be applied to accounts with a credit rating \leq a given value. We anticipate such algorithms will be used in conjunction with other such algorithms to apply different rules based on the customer's credit rating.
- **Different rules based on an account or service agreement characteristic.** You can set up algorithms to only be applied if an account or one of its service agreements has a given characteristic type and value effective within the last X days. For example, you could set up an algorithm that would process an account if it had a broken payment arrangement service agreement within the last 365 days (broken payment arrangement service agreements are marked with a given char type and value). You could set up a different algorithm that treated strategic customers more leniently (given that the definition of "strategic" could be held in an account or service agreement characteristic).
- **Skipping a bill if it has a future postpone date.** You can set up algorithms to ignore an unpaid bill if it has a "postpone date" that's in the future. This date would be defined on the bill using an ad hoc characteristic. The characteristic type is defined as a parameter on the algorithm.
- **Special process for bills with disputed debt.** You can set up an algorithm to create a given type of overdue process if the bill has disputed debt. By "disputed debt", we mean a bill with a financial transaction that's linked to an unbalanced match event that's marked as **Disputed**.
- **Special process for bills with credit balances.** You can set up an algorithm to create a special overdue process if the bill's total charges are negative (i.e., a credit bill). Such bills should be very unusual as we strongly recommend that credit bills should have their credit balance applied to an overpayment service agreement during bill completion.
- **Different rules when all service agreements are inactive.** You can set up an algorithm that should only be applied if all service agreements are inactive. We anticipate this can be used to differ the type of overdue process as if an account does not have active service agreements; there is nothing to stop to encourage them to pay.
- **Different rules based on age and amount of a bill.** You can set up an algorithm to create different types of overdue processes based on the age and amount of the bills.

You are not limited by the base-package's rules. If your implementation requires options that are not supported by the base-package algorithm, simply develop your own.

Holding A Process's Events

Refer to [Holding Events](#) for a description of how to prevent the activation of an overdue / cut processes events while a given condition is true. The condition is defined in an algorithm so you can set up the system to prevent event activation for practically any condition. For example, you might want to hold event activation while the bill's account has an active case (e.g., a user might set up a case to conduct an ad hoc investigation of billing discrepancies). When the case closes, you'd want the process to start up where it left off.

Bill Info Shows Unpaid and Write-off Amounts

The base-package **Bill Information** algorithm (plugged in on the [Installation](#) record) is responsible for constructing a bill's summary information that appears throughout the system. This algorithm can be configured to show the unpaid and write-off amounts of each bill.

Writing Off Bills

The topics in this section provide background information on how to write off bills.

Contents

- [After Service Is Cut, Write-Off Oriented Events Can Commence](#)
- [Collection Agency Referrals](#)
- [Writing Off Bills Will Create Write-Off Adjustments And Possibly A Match Event](#)
- [Small Amount Write-Downs](#)
- [Bill Info Shows Written Off Amounts](#)
- [Online Write Off Of Bills Is Performed On Bill-Main](#)
- [Online Reversal Of Written Off Bills Is Performed On Bill-Main](#)
- [Write Offs And Overdue Process Cancellation](#)
- [An Alert Can Show Written Off Bills](#)

After Service Is Cut, Write-Off Oriented Events Can Commence

After the last service agreement has been stopped, overdue events to manage the write-off of the overdue process's bill(s) should be triggered. Please note that a separate overdue process is NOT created to manage write-offs. Rather, events associated with the original overdue process will handle the write-off activities.

Collection Agency Referrals

Before debt is written off, many implementations refer the unpaid bills to a collection agency. The following points describe how to implement this.

- Set up an overdue event type with an activation algorithm that refers an overdue process's bill(s) to a collection agency.
- When such an event is activated, the system creates a [Collection Referral](#) record for a collection agency. The specific agency to which the debt is referred is controlled by the event type's activation algorithm. The sample algorithm simply refers debt to the collection agency with the least amount of referred debt. If you prefer different logic, you must develop your own algorithm.
- A collection agency referral history record is linked to an account. It contains the amount of debt referred to the collection agency. It is the creation of this record that triggers the interface of information to the collection agency. The method used to interface the information to the agency is defined on the collection agency's record. Refer to [Setting Up Collection Agencies](#) for more information.
- If the collection agency is successful in obtaining the funds, a payment will be added. If the payment satisfies the cancel criteria defined on the overdue process template's cancellation plug-in, the overdue process will cancel. When an overdue process is cancelled, the cancel criteria on the overdue process's template are executed. We strongly recommend plugging in an algorithm that will cancel collection agency referrals when an overdue process is cancelled.

- If the collection agency is not successful in obtaining your funds after a given amount of time, you probably want to cancel the referral and write-off the debt. The cancellation of the referral will happen automatically if you set up your overdue process template to have an event that creates a collection agency cancellation X days after the referral. You can cancel a referral manually by simply creating a new collection agency referral history record (with a type of **cancel**).

Collection agencies are notified of the cancellation of a referral by the creation of a new collection agency referral history record (with a type of cancel). This record will be interfaced to the agency in the same manner used to interface a new referral (see above).

Log entry. The base-package overdue event activation algorithms that make and cancel collection agency referrals insert rows in the overdue process's [log](#) to audit these events.

Writing Off Bills Will Create Write-Off Adjustments And Possibly A Match Event

Most overdue process templates will be configured to contain an event that writes-off the unpaid balance of bills.

Processing is redirected to another algorithm. It should be noted that the base-package overdue event activation algorithm that writes off bills simply redirects the call to the **Write-Off Bill** algorithm plugged in on the account's [Collection Class Overdue Rules](#). The reason for this redirection is because users can manually write-off a bill (using [Bill - Main](#)) and when they do this, we want to invoke the same logic as when an overdue process writes-off a bill.

The base-package **Collection Class Overdue Rules - Write-Off Bill** algorithm determines the amount that should be written off for each distribution code on each unpaid financial transaction. It then creates a separate adjustment for each service agreement where the lines on the adjustment contain the amount to be written off for each distribution code. If the unpaid financial transactions are not linked to a match event, the write-off adjustments plus the unpaid financial transactions will be linked to a new match event. If the unpaid financial transactions were linked to an unbalanced match event, the write-off adjustments will be added to the existing match event (thus making it balanced).

If partial payments were made against a bill, the amount written off will be prorated in light of the partial payments. For example, if 20% of a bill had been paid, 80% of each distribution code will be written off.

For example, assume the original bill's FT looked as follows (note, if the bill has multiple service agreements this will need to be done for each SA's FT's):

- Debit A/R \$110.00
- Credit Flat Charge Revenue (\$50)
- Credit Usage Revenue (\$50)
- Credit City Tax Payable (\$5)
- Credit State Tax Payable (\$5)

Assume the customer had made a partial payment of \$11 and it was matched to this FT. At write-off time, the system will create an adjustment whose adjustment lines will cause each distribution code to be written off by 90% (100% - \$11 / \$110). For example:

- Debit Flat Charge Revenue (or some W/O Expense) \$45
- Debit Usage Revenue \$45
- Debit City Tax Payable \$4.50
- Debit State Tax Payable \$4.50
- Credit A/R \$99

This adjustment will then be linked to the original match event on which the payment was linked (thus making it **Balanced**).

Log entry. The base-package overdue event activation algorithm that writes off bills inserts a row into the overdue process's [log](#) for each bill written off.

Small Amount Write-Downs

Many organizations will write-down a bill whose value is small early in an overdue process. The base-package overdue event activation algorithm has parameters to support this requirement (this algorithm allows you to write-off an overdue process's bill(s) if their value is less than a threshold amount).

If your organization writes-down small amounts differently than large amount, simply set up an overdue event type to reference such an activation algorithm and position it in the appropriate place in the overdue process template.

Bill Info Shows Written Off Amounts

The base-package **Bill Information** algorithm (plugged in on the [Installation](#) record) is responsible for constructing a bill's summary information that appears throughout the system. This algorithm can be configured to show the amount written-off for each bill.

Online Write Off Of Bills Is Performed On Bill-Main

If a bill's account is associated with a [Collection Class Overdue Rule](#) that has a **Write Off Bill** algorithm, a button appears on [Bill - Main](#) if the bill hasn't been written off. When clicked, the **Write Off Bill** algorithm is invoked to write-off the bill.

Online Reversal Of Written Off Bills Is Performed On Bill-Main

If a bill's account is associated with a [Collection Class Overdue Rule](#) that has a **Write Off Bill** algorithm, a button appears on [Bill - Main](#) if the bill has been written off. When clicked, the **Write Off Bill** algorithm is invoked (in reversal mode). The algorithm cancels the write-off adjustments (thus making the bill payable again). It also schedules the account for review by the [Overdue Monitor](#) the next time it runs.

Write Offs And Overdue Process Cancellation

It should be stressed that writing-off a bill may cause an overdue process to be canceled because the bill's FT will be linked to a **balanced** match event (note, the specific [cancel criteria](#) are in a plug-in so this is not a hard rule).

The following points highlight interesting aspects of bill write-off and overdue process cancellation:

- If a user writes off a bill [real time](#) and the bill has an **Active** overdue process, the process's cancel criteria will be invoked. This typically results in the cancellation of the overdue process.
- If an overdue event writes off a bill, the state of the process depends on your cancel criteria and where the overdue event is positioned in the overdue process. For example,
 - Imagine an overdue process that has an overdue event that writes off small amounts of debt early in the process. If such a process is applied against bill with a small amount of debt, the process will be canceled (when the event activates).
 - Contrast this to an overdue process where the last event writes off the bill. Because there are no other events to activate, the process will complete (i.e., it will not be canceled).

An Alert Can Show Written Off Bills

A base-package **Control Central Alert** algorithm ([C1-WO-BILL](#)) can be set up to highlight if the account in context has written off bills. This algorithm is plugged in on the [Installation](#) record.

Bill-Oriented Payment Arrangements

A payment arrangement is an agreement with a customer to payoff severely overdue debt in **billed** installments. Bills sent to customers with payment arrangements contain charges for both their current services and their payment arrangement installment amount.

Nomenclature. Some organizations refer to payment arrangements as “current bill plus” agreements because the customer’s bills contain charges for both their current debt plus their installment amount. After the customer has paid off their overdue debt, the payment arrangement closes and the customer’s bills only contain charges for their current debt.

The topics in this section describe how to set up a payment arrangement and how the system monitors the ongoing arrangements.

Contents

- [Creating Payment Arrangements](#)
- [Installment, Payoff and Current Amounts](#)
- [Breaking A Bill-Oriented Payment Arrangement](#)
- [Online Breaking](#)
- [Online Canceling](#)

Creating Payment Arrangements

When you create a payment arrangement, you are actually creating a service agreement. This service agreement is just like other service agreements in that:

- It holds debt.
- It is periodically billed (thus creating unmatched bill segment financial transactions).
- When a payment is received, the payment segment financial transactions are matched to the bill segment financial transactions.

Debt is transferred to a payment arrangement service agreement (PA SA) from the customer's delinquent bills at the inception of the payment arrangement.

When you transfer debt from the overdue bills to a PA SA, transfer adjustments are created to transfer debt from the delinquent SAs to the PA SA. Match events are created to link the "transfer from" adjustments to the original unpaid financial transactions (FT). When all FT's on a bill are linked to **balanced** match events, the bill is no longer considered overdue and any active overdue process (and its related cut processes) will be cancelled. Refer to [How Are Overdue Processes Canceled](#) for more information.

Use the Payment Arrangement Transaction. Use the [Payment Arrangement - Bill Oriented](#) to set up bill-oriented payment arrangements. This transaction creates a PA SA, transfers overdue bills to it, and sets up the installment amount. This transaction is also used if you need to break or cancel the payment arrangement.

Installment, Payoff and Current Amounts

Warning! If you do not understand the difference between payoff balance and current balance, refer to [Current Amount versus Payoff Amount](#).

When you set up a payment arrangement service agreement (PA SA), you transfer delinquent debt to the PA SA using transfer adjustments. After moneys are transferred, the system sets the PA SA's current balance to zero. At this point, there will be no overdue bills. If the customer neglects to pay bills containing charges associated with the payment arrangement, an overdue process will ensue.

PA SA's start their life with a non-zero payoff balance (i.e., they have debt when first started). This debt is transferred from the bills whose outstanding debt necessitated the creation of the PA SA.

The installment amount that the customer is billed is determined by the number of installments used to payoff the debt. For example, if the customer owes \$500 and they want to pay this off in 10 installments, you'd set up the installment amount to be \$50. The installment amount is saved on the PA SA's recurring charge amount. If the customer again falls into arrears on their bills, you can transfer additional bills to the PA SA. You can also change the installment amount as needed.

A PA SA's payoff balance typically differs from its current balance. The payoff balance is the amount of debt remaining to be paid off under the terms of the payment arrangement. The current balance is the installment amount that has been billed but not paid. For example, a customer who is paying off \$500 with 10 installments of \$50 would have an initial payoff balance of \$500 and a current balance of \$0. After the first bill, the PA SA would still have a payoff balance of \$500, but its current balance would be \$50. When the customer pays, the PA SA's payoff balance would fall to \$450 and its current balance would return to \$0.

The following table contains a financial example of a customer who sets up a payment arrangement to payoff \$1,000 of debt in \$10 installments.

Event	Normal SA's GL Accounting	PA SA's GL Accounting	Normal SA's Current Balance	Normal SA's Payoff Balance	PA SA's Current Balance	PA SA's Payoff Balance
Prior to creation of payment arrangement	N/A	N/A	1000	1000	N/A	N/A
Transfer debt from normal SA(s) to PA SA	Xfer 1000 A/R <1000>	PA A/R 1000 Xfer <1000>	0	0	1000	1000
Set current balance to zero on PA SA	N/A	N/A	0	0	0	1000
Customer is billed (\$50 for new debt and \$10 of payment arrangement debt)	A/R 50 Revenue <50>	N/A	50	50	10	1000
Customer pays \$60	Cash 50 A/R <50>	Cash 10 PA A/R <10>	0	0	0	990

When the customer pays off the payment arrangement debt, the system automatically closes the PA SA after it final bills (assuming the PA SA's SA type references a bill segment type that has a bill segment creation algorithm of ***Recurring Charge With Auto Stop***).

Breaking A Bill-Oriented Payment Arrangement

If a customer neglects to pay a bill, an overdue process is created. If the bill contains charges from a payment arrangement, you'll want to "break" the payment arrangement. By "break", we mean cancel the arrangement and reinstate the debt under the originating bills. To do this, you must configure the [cut process template](#) used to cut the PA SA to contain a cut event that breaks the payment arrangement.

Processing is redirected to another algorithm. It should be noted that the base-package cut event activation algorithm that breaks a payment arrangement simply redirects the call to the ***Bill-Based Payment Arrangement*** algorithm plugged in on the account's [Collection Class Overdue Rules](#). The reason for this redirection is that users can manually break a payment arrangement (using [Payment Arrangement - Bill Oriented](#)). When they do this, we want to invoke the same logic as when a cut event activation algorithm breaks a payment arrangement.

The base-package ***Collection Class Overdue Rule - Bill-Based Payment Arrangement*** algorithm does the following:

- Creates "cancel adjustments" for all unpaid financial transactions associated with the payment arrangement (e.g., billed, but not paid installments).
- Cancels ALL adjustments that were used to transfer the debt to the payment arrangement. When these are cancelled, the original bills will become unpaid (and the debt will be rather

old by this point).

- Syncs up current balance with payoff balance on the PA SA.
- Makes the PA SA **pending stop** and expires it (SA activation will stop the SA when it next runs).
- If there is a credit left on the PA SA (because payments were made against the arrangement), the credit will be distributed amongst the bills that contributed debt to it (the oldest bills are paid off first). This relief will occur via transfer adjustments from the PA SA to the original SA's.
- Note, if there is a debit left (e.g., because LPC were issued or some other type of adjustment was created by an operator), an error is issued as we don't handle new debit on a payment arrangement.
- Inserts a characteristic on the PA SA to indicate that it has been broken (you might want to use this to cause more strict overdue rules to be applied to accounts with broken payment arrangements).
- Marks the account so it will be reviewed by the [Overdue Monitor](#) the next time it runs.

The PA SA must final bill before it closes. It's important to note that the PA SA will only close after the PA SA is final billed. This is OK as it won't have any money left on it.

When the Overdue Monitor next runs, it will analyze the account's reinstated bills. We recommend setting up a **Collection Class Overdue Rule - Overdue Monitor Rule** algorithm that has more stringent rules if the account has a broken payment arrangement within the last X days. The base-package algorithm ([C1-CB-CR-RAT](#)) supports this type of processing.

Log entry. The base-package overdue event activation algorithm that breaks a payment arrangement SA inserts a row into the overdue process's [log](#).

Online Breaking

A user can click a button on the [Payment Arrangement - Bill Oriented](#) page to break a payment arrangement real time. When clicked, the **Collection Class Overdue Rule - Bill-Based Payment Arrangement** algorithm is called to break the payment arrangement. This is the same algorithm called when a cut event break a payment arrangement.

Online Canceling

A user can click a button on the [Payment Arrangement - Bill Oriented](#) page to cancel a payment arrangement real time. Cancellation should be used when you want to "logically delete" a PA SA because it shouldn't have been created.

The logic described above for breaking a payment arrangement is executed when a user cancels a payment arrangement; the only difference is that the PA SA is marked with a different characteristic type / value than when it is broken. The "broken" characteristic type / value can be used to apply stricter rules to the account when it's next reviewed by the [Overdue Monitor](#).

Creating Overdue and Cut Procedures

Your overdue procedures define how your organization collects overdue debt. Your cut procedures define how your organization cuts (i.e., stops) service agreements when collection attempts fail. In this section, we describe how to set up the data that controls these procedures.

Warning! There are numerous ways to design your overdue and cut procedures. Some designs will result in easy long-term maintenance; others will result in maintenance headaches. In this section, we provide information to help you understand the ramifications of the various options. Before you set up your overdue and cut procedures, we encourage you to gain an intuitive understanding of these options by using the system to prototype the alternatives.

Contents

[Designing Your Overdue Procedures](#)
[Set Up Tasks](#)
[Setting Up Overdue Processing](#)

Designing Your Overdue Procedures

The design of your overdue procedures is an iterative process. Over time, you will develop skills that will allow you to skip some steps. However, when you're starting out, we recommend you use the following matrix as your guide. When the matrix is complete, you're ready to set up the overdue processing control tables.

Account's Collection Class →		
Account's Division ↓		

The topics discussed below will gradually complete this matrix using a simple case study.

For more information about how the information in this matrix is used to monitor your customers' debt, refer to [How Does The Overdue Monitor Work](#).

Contents

[Your Divisions Are Frequently Preordained](#)
[Designing Your Collection Classes](#)
[Designing Overdue Monitoring Rules](#)
[Designing Overdue Process Templates and Event Types](#)
[Designing Cut Process Templates and Event Types](#)

Your Divisions Are Frequently Preordained

An account's division defines the jurisdiction whose rules govern the account. For example, an account's division controls how its payments are distributed, if / how late payment charges are levied, etc. Divisions have typically been designed in advance of designing your overdue rules.

In our example, we assume you have two divisions: North and South.

Account's Collection Class →		
Account's Division ↓		
North		
South		

Designing Your Collection Classes

Multiple collection classes are needed when your organization has different overdue rules and / or procedures based on the type of customer. If all customers are treated the same way, you'll have a single collection class (call it **Generic**). However, if you're like many organizations, you will have multiple collection classes.

For example, for commercial/industrial customers, you probably don't worry until they owe you more than, say, \$100 after 20 days. For residential customers, you probably don't worry until they owe you more than, say, \$5 after 20 days. In this situation, you will have at least two collection classes: one for commercial/industrial customers, the other for residential customers.

In our example, we assume you have two collection classes: Residential and Commercial/Industrial.

Account's Collection Class →	Residential	Commercial/Industrial
Account's Division ↓		
North		
South		

Note. There are two very different ways to monitor your accounts for overdue debt. This chapter describes the method referred to as Overdue Processing. Refer to [Collection / Severance / Write Off](#) for a description of the other method. We anticipate that most organizations will only use a single method. If your organization opts to use both methods, you will need to set up the corresponding collection classes.

Designing Overdue Monitoring Rules

At this point, we have the rows and columns defined in our matrix. Now it's time to work on the individual cells.

Single currency. We've assumed that your implementation works in a single currency. If this is not true, you will need to add a 3rd dimension that will have a value for each currency code.

Each cell will contain the rules that the [Overdue Monitor](#) uses to determine if an account has overdue debt. These rules will eventually be configured using one or more algorithms on [Collection Class Overdue Rules](#).

Account's Collection Class →	Residential	Commercial/Industrial
Account's Division ↓		

North		
South		

Note. If the Overdue Monitor encounters an account whose collection class and division does not have overdue rules set up, it will issue an error.

Determining the rules in each cell can be straightforward or complicated; it depends on how your organization works. Our case study assumes the following:

- For residential debt (regardless of division) we have the following rules:
 - Highest priority. If a bill exists with unpaid FT's > \$0 that is older than 50 days, create the "accelerated overdue process" for residential customers. We'll talk more about this process later.
 - Medium priority. If the account's has a broken payment arrangement within the last 60 days with an unpaid bill > \$0 that is older than 20 days, create the "broken payment arrangement overdue process".
 - Lower priority. If a bill exists with unpaid FT's > \$25 that is older than 25 days, create the "courtesy reminder overdue process" for residential customers. We'll talk more about this overdue process later.
- For commercial-industrial debt (regardless of division) we have the following rules:
 - Highest priority. If a bill exists with unpaid FT's > \$0 that is older than 45 days, create the "commercial 45 days late overdue process". We'll talk more about this process later.
 - Medium priority. If the account's credit rating is < 550 and has an unpaid bill > \$0 that is older than 20 days, create the "risky commercial customer overdue process".
 - Lower priority. If a bill exists with unpaid FT's > \$100 that is older than 30 days, create the "commercial 30 days late overdue process". We'll talk more about this overdue process later.

Given the above, our matrix will look as follows:

Account's Collection Class → Account's Division ↓	Residential	Commercial/Industrial
North	<p>Highest Priority: If a bill exists with unpaid FT's > \$0 that is older than 50 days, create the accelerated overdue process for residential customers.</p> <p>Medium Priority: If the account has a broken payment arrangement within the last 60 days with an unpaid bill > \$0 that is older than 20 days, create the broken payment arrangement overdue process.</p> <p>Lowest Priority: If a bill exists with unpaid FT's > \$25 that is older than 25 days, create the courtesy reminder overdue process for residential customers.</p>	<p>Highest Priority: If a bill exists with unpaid FT's > \$0 that is older than 45 days, create the commercial 45 days late overdue process.</p> <p>Medium priority. If the account's credit rating is < 550 and has an unpaid bill > \$0 that is older than 20 days, create the risky commercial customer overdue process.</p> <p>Lowest Priority: If a bill exists with unpaid FT's > \$100 that is older than 30 days, create the commercial 30 days late overdue process.</p>

South	Same as above	Same as above
-------	---------------	---------------

The rules are limited by your imagination (and business requirements). While the base-package **Overdue Monitor Rule** algorithm ([C1-CB-CR-RAT](#)) supports the above scenarios, we'd like to stress these are just examples. Your implementation can operate using very different rules by either configuring the base-package algorithm (it has many parameters that you can use to tailor your rules) OR by introducing a new algorithm. Refer to [Highlights Of The Sample Overdue Monitor Rule Algorithm](#) for the options delivered with the base-package.

Designing Overdue Process Templates and Event Types

The following table shows a sample overdue process template for one of the rules in the Residential / North cell in the previous section's matrix.

Overdue Process Template	Overdue Event Type	When Triggered
Accelerated overdue process for residential customers	Old debt letter	At inception of process
	Cut active service agreements	10 days after inception
	Reduce customer's credit rating	10 days after inception (i.e., at the same time the cut process is created)
	Write down small debt	0 days after <u>completion</u> of the cut process(es)
	Refer debt to collection agent	0 days after attempting the small write down (this means that either the small write-down or the agency referral will take place as if the write-down is successful, the bill's FTs will be matched to balanced match events and the overdue process will stop)
	Cancel collection agent referral	45 days after referral
	Write-off debt	0 days after collection agent cancellation

You should create a similar table for each of the distinct overdue process templates in your matrix.

At this point, you've designed the distinct overdue process templates. Next, you'll need to design the algorithms that control their overdue processes:

- A template's **Calculate Unpaid and Original Amount** algorithm calculates the original and unpaid amounts of the objects being collected by a process. These values are used throughout the overdue processing module.
- A template's **Cancel Criteria** algorithm is executed to determine if a process should be cancelled. Refer to [How Are Overdue Processes Cancelled](#) for the details.

- A template's **Cancel Logic** algorithm is executed to cancel a process. Refer to [How Are Overdue Processes Cancelled](#) for the details. Please note that the logic embodied in this type of algorithm can be sophisticated because it is responsible for stopping an ongoing process's activities (e.g., this could involve cancelling field activities or cases). Cancellation algorithms are also responsible for inserting [log](#) entry(s).
- A template's **Hold Event Activation** algorithm is invoked to determine if the [Overdue / Cut Event Manager](#) should [suspend the activation](#) of the process's events.
- A template's **Information** algorithm is invoked to construct the [override "info string"](#).

Next, extract each unique event type from the above table:

Overdue Event Type	Action
Old debt letter	Create a customer contact
Cut active service agreement(s)	Start a cut process for every active SA with an unpaid FT on the bill
Reduce customer's credit rating	Insert an account credit rating history record
Write down small debt	Create write-down adjustments if unpaid debt is less than \$x
Refer debt to collection agent	Create a collection agency referral
Cancel collection agent referral	Cancel the collection agency referral
Write off unpaid debt	Create adjustments to write-off unpaid debt

At this point, you know the distinct event types. Next, you'll need to design the algorithms that control the lifecycle of each event type:

- The event type's **Event Activation** algorithm(s) are executed by the [Overdue / Cut Event Manager](#) on its trigger date. The following points describe the logic embodied in such an algorithm:
 - The activity that happens on the trigger date (e.g., creation of a customer contact, To Do, etc.). Refer to [Overdue Events Can Do Many Things](#) for the details.
 - Whether the event is transitioned into the **Waiting** or **Complete** state when it's triggered. Refer to [Some Events Can Wait](#) for the details.
 - How the log entry(s) associated with event activation will be constructed. The base-package algorithms allow you to control the verbiage in the log entry by defining the desired message number on the algorithm. This means that you may have to set up new messages. Refer to [Activating Events Should Add A Log Entry](#) for the details.
- The event type's **Cancel Logic** algorithm(s) are invoked when [an event is cancelled](#). The following points describe the logic embodied in such an algorithm:
 - If the event is allowed to be canceled. This logic may be necessary if some conditions prevent events of this type from canceling. For example, you may want to prevent an event from canceling when there are later dependent events that aren't canceled.
 - Any ancillary actions that take place during cancellation.
 - How the [log entry\(s\)](#) associated with event cancellation will be constructed.

- The event type's **Monitor Waiting Event** algorithm(s) are invoked to [monitor a waiting event](#). These algorithms are responsible for transitioning a **Waiting** event to **Complete** if the object on which it's waiting is complete.
- The event type's **Event Information** algorithm is invoked to construct the [override "info string"](#).

Once you've designed each event type's algorithms, you're ready to design your cut processes.

Designing Cut Process Templates and Event Types

While not required, many overdue processes will contain an overdue event that cuts (i.e., stops) service (in the hopes that stopping service will inspire the customer to pay). When such an event activates, it creates one or more cut process(es).

The system allows you to control if and how service is cut by setting up **Cut Process Rule** algorithms on your SA types. This allows different rules for different types of service agreements. In addition, because these rules are embodied in algorithms, a given SA type can have conditional logic that controls the type of cut process created. For example, you may have a different cut process if the customer has life support equipment, or if it's winter, or ...

If a service agreements of a given type should never be cut. If you have certain SA types that should never be cut, do NOT define an algorithm in the SA type's **Cut Process Rule**.

Determining your **Cut Process Rules** can be straightforward or complicated; it depends on how your organization works. Our case study assumes the following:

- For service agreements associated with metered service, we have the following rules:
 - Highest priority. If the customer has life support requirements, create "cut process in light of life support" process.
 - Lowest priority. Otherwise, create a "metered service" cut process.
- For service agreements associated with charitable contributions, create an "expiration only" cut process.

Once you've determined if / how each SA type is cut, you need to design your cut process templates (you'll need one for each unique method of cutting a service agreement). The following table shows the cut process templates referenced above. Adjacent to each process are its events and an indication of when they are triggered.

Cut Process Template	Event Number	Cut Event Type	Dependent On Event(s)	Trigger Date Set To X Days After Completion Of Dependent Events
Metered service	10	Letter - 48 hour disconnect for non-payment warning	N/A – first event	0
	20	Field activity – disconnect for non-payment	10	2
	30	'Service has been disconnected' letter	20	0
	40	Expire service agreement	20	10

Cut process in light of life support	10	Generate delinquent life support customer To Do entry (seeking approval for the cut)	N/A – first event	0
	20	Letter - 72 hour disconnect for non-payment warning	10	0
	30	Generate impending life support cutoff To Do entry to C&C rep	20	3
	40	Field activity – cut for non-payment	30	0
	50	Service has been disconnected letter	40	0
	60	Expire service agreement	40	10
Expiration only	10	Expire service agreement	N/A – first event	

At this point, you've designed the distinct cut process templates. Next, you'll need to design the algorithms that control the lifecycle of their cut processes:

- A template's **Cancel Logic** algorithm is executed to cancel a process. In addition to cancelling the event, these algorithms are also responsible for inserting [log](#) entry(s).
- A template's **Information** algorithm is invoked to construct the [override "info string"](#).

Next, extract each unique event type from the above table:

Cut Event Type	Action
48-hour warning letter	Create a customer contact - 48-hour warning letter (letters are created via customer contacts)
72-hour warning letter	Create a customer contact - 72-hour warning letter
Disconnect for non payment	Create a cut for non-payment field activity
Permission to start a cut process for a life support customer	Create To Do seeking permission to send 72 hour letter
Permission to create a cut field activity for a life support cutoff	Create To Do seeking permission to issue a cut field activity
Service has been disconnected letter	Create a customer contact - service cut letter
Expire service agreement	Expire service agreement

At this point, you know the distinct event types. Next, you'll need to design the algorithms that control the lifecycle of their events:

- The event type's **Event Activation** algorithm(s) are executed by the [Overdue / Cut Event Manager](#) on its trigger date. The following points describe the logic embodied in such an algorithm:
 - The activity that happens on the trigger date (e.g., creation of a customer contact, To Do, etc.). Refer to [Cut Events Can Do Many Things](#) for the details.
 - Whether the event is transitioned into the **Waiting** or **Complete** state when it's triggered. Refer to [Some Events Can Wait](#) for the details.

- How the log entry(s) associated with event activation will be constructed. The base-package algorithms allow you to control the verbiage in the log entry by defining the desired message number on the algorithm. This means that you may have to set up new messages. Refer to [Activating Events Should Add A Log Entry](#) for the details.
- The event type's **Cancel Logic** algorithm(s) are invoked when [an event is cancelled](#). The following points describe the logic embodied in such an algorithm:
 - If the event is allowed to be canceled. This logic may be necessary if some conditions prevent events of this type from canceling. For example, you may want to prevent an event from canceling when there are later dependent events that aren't canceled.
 - Any ancillary actions that take place during cancellation.
 - How the [log entry\(s\)](#) associated with event cancellation will be constructed.
- The event type's **Monitor Waiting Event** algorithm(s) are invoked to [monitor a waiting event](#). These algorithms are responsible for transitioning a **Waiting** event to **Complete** if the object on which it's waiting is complete.
- The event type's **Event Information** algorithm is invoked to construct the [override "info string"](#).

Set Up Tasks

The above topics provided background information about how overdue processing works. The following discussion summarizes the various set up tasks.

Contents

- [Cut Event Types](#)
- [Cut Process Templates](#)
- [SA Type - Cut Process Rules](#)
- [Overdue Event Types](#)
- [Overdue Process Templates](#)
- [Collection Classes](#)
- [Collection Class Overdue Monitor Rules](#)
- [Feature Configuration](#)
- [Overdue and Cut Event Cancellation Reasons](#)
- [Collection Agencies](#)
- [Alert To Highlight Active Overdue Processes](#)
- [Bill-Oriented Collection - Additional Set Up](#)

Cut Event Types

You will find that most of the time spent setting up your cut event types is spent setting up the objects that are referenced on the cut event type algorithms. For example, if you use the base-package algorithms, you may need to set up the following:

- The various "types" for the objects created by the plug-ins. For example, if a cut event type creates a To Do entry, you must supply the desired To Do type.
- [Foreign key characteristic types](#) that are used to reference the ancillary objects in the [log entries](#) (e.g., if an event creates a customer contact, the log references this customer contact using a FK characteristic type). Note, many of these will exist in the base-package.

- [Messages](#) that are used to define the verbiage in the [log entries](#). For example, if you use the base-package algorithm that creates a customer contact, you must supply the desired message category and number that contains the verbiage that appears in the log when customer contacts are created. Note, messages have been set up for all base-package algorithms (this means you should not have to set up new messages).
- Etc.

The only way to compile the complete list is to design the parameters for each cut event type algorithm. Refer to [Cut Event Type - Main](#) for the supported plug-in spots.

After you've set up the objects referenced on the algorithms, you can then set up the algorithms. Only then can you set up the cut event types.

Cut Process Templates

After your cut event types exist, you can set up your cut process templates. You will find that most of the time spent setting up your cut process templates is spent setting up the objects that are referenced on the cut process template algorithms. Refer to [Cut Process Template - Main](#) for the supported system events.

SA Type - Cut Process Rules

After you've created your cut process templates, you can set up the algorithms that hold your [cut process rules](#). These are plugged-in on [SA Type - Algorithm](#) in the **Cut Process Rule** system event.

Overdue Event Types

You will find that most of the time spent setting up your overdue event types is spent setting up the objects that are referenced on the overdue event type algorithms. For example, if you use the base-package algorithms, you will set up the following:

- The various "types" for the objects created by the plug-ins. For example,
 - If an overdue event type creates a To Do entry, you must set up the To Do type.
 - If an overdue event type creates a customer contact, you must set up the customer contact type.
 - If an overdue event type writes off debt, you must set up the adjustment types.
 - ...
- [Foreign key characteristic types](#) that are used to reference the ancillary objects in the [log entries](#) (e.g., if an event creates a customer contact, the log references this customer contact using a FK characteristic type). Note, many of these will exist in the base-package.
- [Messages](#) that are used to define the verbiage in the [log entries](#). For example, if you use the base-package algorithm that creates a customer contact, you must supply the desired message category and number that contains the verbiage that appears in the log when customer contacts are created. Note, messages have been set up for all base-package algorithms (this means you should not have to set up new messages).
- Etc.

The only way to compile the complete list is to design the parameters for each overdue event type algorithm. Refer to [Overdue Event Type - Main](#) for the supported plug-in spots.

After you've set up the objects referenced on the algorithms, you can then set up the algorithms. Only then can you set up the overdue event types.

Overdue Process Templates

After your overdue event types exist, you can set up your overdue process templates. You will find that most of the time spent setting up your overdue process templates is spent setting up the objects that are referenced on the overdue process template algorithms. Refer to [Overdue Process Template - Main](#) for the supported system events.

Collection Classes

Set up [collection classes](#) as per your [overdue procedures](#). Make sure to indicate that these collection classes use the **Overdue** collection method (only accounts linked to collection classes designated as using the **Overdue** collection method or processed by the Overdue Monitor).

Collection Class Overdue Monitor Rules

After your overdue process templates exist, you can set up your **Overdue Monitor Rules**. These rules are algorithms plugged in on [Collection Class Overdue Rules](#). You will find most of the time spent setting up these algorithms is spent setting up the objects referenced on the base-package algorithm.

Feature Configuration

You must set up a [Feature Configuration](#) to define parameters that control various overdue processing options.

The following points describe the various **Option Types** that must be defined:

- **Trigger Date: Y-Workdays, N-Calendar Days.** This option controls how the system computes the trigger dates on overdue and cut events. Enter **Y** if the system should use workdays. Enter **N** if the system should use calendar days. Refer to [Calendar vs Work Days](#) for the details.
- **Payment Arrangement Type (B/S/A).** This option indicates whether your implementation uses the [Balance-Oriented Payment Arrangements](#) (value **S**), [Bill-oriented Payment Arrangements](#) (value **B**) or both (value **A**). The value governs the navigation path for the payment arrangement lines in the [Account History](#) and [Credit & Collection](#) zones.
- **Champion Template\$Challenger Template\$Percentage(1-100).** You need only set up options of this type if your implementation implements [Champion / Challenger](#) functionality. Options of this type are entered in the format **A\$B\$nnn** where A is the overdue process template of the champion template, B is the overdue process template of the challenger template, and C is the percent of the time that the system should create the challenger template. The overdue monitor uses this option to override the champion overdue process template X% of the time with the challenger template. You may enter any number of these options (but only one per Champion Template).

Overdue and Cut Event Cancellation Reasons

Overdue events can be cancelled automatically and manually (at the discretion of a user). Regardless of the method of cancellation, a cancellation reason must be supplied. You set up your overdue event cancellation reasons using [Overdue Event Cancellation Reason - Main](#) and [Cut Event Cancellation Reason - Main](#).

Collection Agencies

If you refer debt to collection agents, you must set up your [collection agencies](#).

Alert To Highlight Active Overdue Processes

If you want an alert to appear if the account has active overdue processes, you must configure an appropriate **Control Central Alert** algorithm ([C1-OD-PROC](#)). This algorithm is plugged in on the [Installation](#) record.

Bill-Oriented Collection - Additional Set Up

The topics in this section provide information on additional set up requirements if you collect on unpaid bills.

Contents

- [One Bill Per Match Event](#)
- [Bill-Based Payment Arrangements](#)
- [Bill-Based Write-off](#)
- [Alert To Highlight Written Off Bills](#)
- [Open-Item Bill Amount Plug-In](#)

One Bill Per Match Event

As mentioned earlier, a bill is considered paid if its financial transactions (FTs) are linked to a **balanced** match event. To determine a bill's outstanding amount, FTs from different bills cannot be commingled on the same match event (but it's OK for a bill's FTs to be on multiple match events). If you stick by the rule of "just one bill per match event" you will then be able to determine the outstanding balance of a partially paid bill. However, if you mix more than one bill under a match event, a particular bill's balance may become indeterminate.

The following Open-Item algorithm types have been provided by the base package to help enforce this rule:

- The Distribute by Bill Due Date **Payment Distribution** algorithm ([C1-PYDS-BDU](#)).
- The match by Bill ID **Payment Distribution Override** algorithm ([C1-PDOV-PYBL](#)).
- The FT cancellation **FT Freeze** algorithm ([C1-CFTZ-COFT](#)).

If any of your customized plug-ins and processes create match events, it is important that these too enforce this rule. You may want to refer to the base package algorithms as an example of how to do this.

Bill-Based Payment Arrangements

If you set up [payment arrangements for unpaid bills](#), you must configure the **Bill-Based Payment Arrangement** algorithm and plug it in on your [Collection Class Overdue Rules](#).

It's important to set up the adjustment types used during payment arrangement creation to NOT print on bills. This is because the base-package algorithm will match the adjustments used to transfer debt to the payment arrangement with the adjustment used to reduce the payment arrangement's current amount by the amount of the transfer if all adjustment types are set up to not print.

Bill-Based Write-off

If you [write-off unpaid bills](#), you must set up the following:

- Set up the adjustment type that will be used to write-off an unpaid financial transaction. This adjustment type must be configured as follows:
 - **Adjustment Amount Type** must be **Calculated Amount**
 - Its distribution code is irrelevant as a separate calculation line will be created for each distribution code on the FT's that is written off and these lines will reference the appropriate distribution code.
 - It must reference an adjustment type char type / value that identifies it as one used to write-off a bill's FTs
 - The following algorithms must be defined on the adjustment type:
 - The **Generate Adjustment** system event must reference an algorithm that has the responsibility of determining how to write-off a FT. This algorithm should be determining the FT's GL details and creating a separate adjustment calculation line for each GL detail. The base package is supplied with a sample algorithm that does this ([C1-ADJG-WO](#)).
 - The **Adjustment Financial Transaction** system event should reference an algorithm that impacts current, payoff and the GL by the amount being written off.
- The distribution codes referenced on the financial transactions must be set up with a characteristic that holds the distribution code used to write-off the original amounts. For example,
 - Distribution codes used to record tax liabilities will typically reference the same distribution code for write-off (most organizations reverse tax liabilities at write-off time)
 - Distribution codes used to record revenue will typically reference a write-off distribution code used to record a write-off expense.
 - Distribution codes used to record receivables will typically not reference a write-off distribution code because receivables are implicitly written off when revenue and tax liabilities are written off.
- Set up an adjustment cancellation code used when a users reverses a written-off bill (reversal involves canceling the write off adjustments).
- Set up a **Write Off Bill** algorithm and plug it in on your [Collection Class Overdue Rules](#). This algorithm will reference the adjustment type described in the previous point.

Alert To Highlight Written Off Bills

If you want an alert to appear if the account has bills with written-off debt, you must configure an appropriate **Control Central Alert** algorithm ([C1-WO-BILL](#)). This algorithm is plugged in on the [Installation](#) record.

Open-Item Bill Amount Plug-In

You must set up the algorithm that computes the original, unpaid, and write-off amounts of your open-item bills. This algorithm is called by other algorithms when these amounts are needed. This algorithm is plugged-in on [Installation](#) in the **Determine Open Item Bill Amounts** spot.

Setting Up Overdue Processing

The topics in this section describe how to set up the control tables to implement your overdue processing.

Contents

- [Setting Up Overdue Event Types](#)
- [Setting Up Overdue Process Templates](#)
- [Setting Up Cut Event Types](#)
- [Setting Up Cut Process Templates](#)
- [Setting Up Collection Class Overdue Rules](#)
- [Setting Up Overdue Event Cancellation Reasons](#)
- [Setting Up Cut Event Cancellation Reasons](#)

Setting Up Overdue Event Types

An overdue event type encapsulates the business rules that govern a given type of overdue event. Open **Admin Menu, Overdue Event Type** to set up overdue event types.

Recommendation. Before using this transaction, we strongly recommend that you review [The Big Picture Of Overdue Events](#).

Description of Page

Enter a unique **Overdue Event Type** code and **Description** for the overdue event type.

Use **Long Description** to provide a more detailed explanation of the purpose of the overdue event type.

The **Algorithms** grid contains algorithms that control important functions. You must define the following for each algorithm:

- Specify the algorithm's **System Event** (see the following table for a description of all possible events).
- Specify the **Algorithm** to be executed when the System Event executes. Set the **Sequence** to **10** unless you have a **System Event** that has multiple **Algorithms**. In this case, you need to tell the system the **Sequence** in which they should execute.

The following table describes each **System Event** (note, all system event's are optional and you can define an unlimited number of algorithms for each event).

System Event	Optional / Required	Description
<i>Cancel Logic</i>	Required	This algorithm is executed to cancel an overdue event. Refer to How Are Events Canceled for the details. Click here to see the algorithm types available for this system event.
<i>Event Activation</i>	Required	This algorithm is executed to activate an overdue event on its trigger date. Refer to Overdue Events Can Do Many Things and How and When Events Are Activated for the details. Click here to see the algorithm types available for this system event.
<i>Event Information</i>	Optional - only used if you want to override an overdue event's info string	This algorithm is executed to construct an overdue event's override info string. Refer to Overdue Event Information Is Overridable for the details. Click here to see the algorithm types available for this system

		event.
<i>Monitor Waiting Events</i>	Optional - only used if events of this type can enter the <i>Waiting</i> state	This algorithm is invoked by the Overdue / Cut Event Manager for events in the Waiting state. Refer to Some Events Wait For Something Before Completing for the details. Click here to see the algorithm types available for this system event.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference [CI_OD_EVT_TYPE](#).

Setting Up Overdue Process Templates

An overdue process template encapsulates the business rules that govern a given type of overdue process. Open **Admin Menu, Overdue Process Template** to set up overdue process templates.

Recommendation. Before using this transaction, we strongly recommend that you review [The Big Picture Of Overdue Processes](#).

Description of Page

Enter a unique **Overdue Process Template** and **Description** for the overdue process template.

Collecting On Object defines the type of object managed by this overdue process. This field actually references a [foreign key characteristic type](#) that references the managed object. For example, if this overdue process template manages overdue bills, you'd reference a foreign key characteristic that references the bill object.

The **Algorithms** grid contains algorithms that control important functions. You must define the following for each algorithm:

- Specify the algorithm's **System Event** (see the following table for a description of all possible events).
- Specify the **Algorithm** to be executed when the System Event executes. Set the **Sequence** to **10** unless you have a **System Event** that has multiple **Algorithms**. In this case, you need to tell the system the **Sequence** in which they should execute.

The following table describes each **System Event** (note, all system event's are optional and you can define an unlimited number of algorithms for each event).

System Event	Optional / Required	Description
<i>Calculate Unpaid & Original Amount</i>	Required	This algorithm is executed to calculate the unpaid and original amounts of the objects associated with the overdue process. These amounts are shown on the overdue process page and in the base-package overdue info string . Click here to see the algorithm types available for this system event.
<i>Cancel Criteria</i>	Required	This algorithm is executed to determine if an overdue process can be cancelled. Refer to How Are Overdue Processes Cancelled for

		the details. Click here to see the algorithm types available for this system event.
<i>Cancel Logic</i>	Required	This algorithm is executed to cancel an overdue process. Refer to How Are Overdue Processes Cancelled for the details. Click here to see the algorithm types available for this system event.
<i>Hold Event Activation Criteria</i>	Optional - only used if overdue processes of this type can be suspended while some condition is true	This algorithm is executed to determine if the activation of overdue and cut events should be suspended. Refer to Holding Events for the details. Click here to see the algorithm types available for this system event.
<i>Overdue Process Information</i>	Optional - only used if you want to override an overdue process's info string	This algorithm is executed to construct an overdue process's override info string. Refer to Overdue Process Information Is Overridable for the details. Click here to see the algorithm types available for this system event.

The **Event Types** control the number and type of overdue events linked to an overdue process when it is first created. The information in the scroll defines these events and the date on which they will be triggered. The following fields are required for each event type:

- **Event Sequence.** Sequence controls the order in which the overdue event types appear in the scroll.
- **Overdue Event Type.** Specify the type of overdue event to be created.
- **Days After.** If **Dependent on Other Events** is on, events will be triggered this many days after the completion of the dependent events (specified in the grid). Set this value to 0 (zero) if you want the event triggered immediately after the completion of the dependent events. If **Dependent on Other Events** is off, events will be triggered this many days after the creation of the overdue process. Refer to [How and When Events Are Activated](#) for the details.
- If **Dependent on Other Events** is on, define the events that must be completed or cancelled before the event will be triggered.
 - **Sequence** is system-assigned and cannot be specified or changed.
 - **Dependent on Sequence** is the sequence of the dependent event.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference [CI_OD_PROC_TMP](#).

Setting Up Cut Event Types

A cut event type encapsulates the business rules that govern a given type of cut event. Open **Admin Menu, Cut Event Type** to set up cut event types.

Recommendation. Before using this transaction, we strongly recommend that you review [Cut Events Are Like Overdue Events](#).

Description of Page

Enter a unique **Cut Event Type** code and **Description** for the cut event type.

Use **Long Description** to provide a more detailed explanation of the purpose of the cut event type.

The **Algorithms** grid contains algorithms that control important functions. You must define the following for each algorithm:

- Specify the algorithm's **System Event** (see the following table for a description of all possible events).
- Specify the **Algorithm** to be executed when the System Event executes. Set the **Sequence** to **10** unless you have a **System Event** that has multiple **Algorithms**. In this case, you need to tell the system the **Sequence** in which they should execute.

The following table describes each **System Event**.

System Event	Optional / Required	Description
<i>Cancel Logic</i>	Required	This algorithm is executed to cancel a cut event. Refer to How Are Events Canceled for the details. Click here to see the algorithm types available for this system event.
<i>Event Activation</i>	Required	This algorithm is executed to activate a cut event on its trigger date. How and When Events Are Activated for the details. Click here to see the algorithm types available for this system event.
<i>Event Information</i>	Optional - only used if you want to override a cut event's info string	This algorithm is executed to construct a cut event's override info string. Refer to Cut Event Information Is Overridable for the details. Click here to see the algorithm types available for this system event.
<i>Monitor Waiting Events</i>	Optional - only used if events of this type can enter the <i>Waiting</i> state	This algorithm is invoked by the Cut / Cut Event Manager for events in the Waiting state. Refer to Some Events Wait For Something Before Completing for the details. Click here to see the algorithm types available for this system event.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference [CI CUT EVT TYPE](#).

Setting Up Cut Process Templates

A cut process template encapsulates the business rules that govern a given type of cut process. Open **Admin Menu, Cut Process Template** to set up cut process templates.

Recommendation. Before using this transaction, we strongly recommend that you review [The Big Picture Of Cut Processes](#).

Description of Page

Enter a unique **Cut Process Template** and **Description** for the cut process template.

The **Algorithms** grid contains algorithms that control important functions. You must define the following for each algorithm:

- Specify the algorithm's **System Event** (see the following table for a description of all possible events).
- Specify the **Algorithm** to be executed when the System Event executes. Set the **Sequence** to **10** unless you have a **System Event** that has multiple **Algorithms**. In this case, you need to tell the system the **Sequence** in which they should execute.

The following table describes each **System Event**.

System Event	Optional / Required	Description
<i>Cancel Logic</i>	Required	This algorithm is executed to cancel a cut process. Refer to How Are Cut Processes Cancelled for the details. Click here to see the algorithm types available for this system event.
<i>Cut Process Information</i>	Optional - only used if you want to override a cut process's info string	This algorithm is executed to construct a cut process's override info string. Refer to Cut Process Information Is Overridable for the details. Click here to see the algorithm types available for this system event.

The **Event Types** control the number and type of events linked to a cut process when it is first created. The information in the scroll defines these events and the date on which they will be triggered. The following fields are required for each event type:

- Event Sequence.** Sequence controls the order in which the cut event types appear in the scroll.
- Cut Event Type.** Specify the type of cut event to be created.
- Days After.** If **Dependent on Other Events** is on, events will be triggered this many days after the completion of the dependent events (specified in the grid). Set this value to 0 (zero) if you want the event triggered immediately after the completion of the dependent events. If **Dependent on Other Events** is off, events will be triggered this many days after the creation of the cut process. Refer to [How and When Events Are Activated](#) for the details.
- If **Dependent on Other Events** is on, define the events that must be completed or cancelled before the event will be triggered.
 - Sequence** is system-assigned and cannot be specified or changed.
 - Dependent on Sequence** is the sequence of the dependent event.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference [CI CUT PROC TMP](#).

Setting Up Collection Class Overdue Rules

Collection class overdue rules contain algorithms that impact accounts associated with a given collection class, division and currency code are managed. Open **Admin Menu, Collection Class Overdue Rules** to set up collection class overdue rules.

Recommendation. Before using this transaction, we strongly recommend that you review [Different Overdue Rules For Different Customers](#).

Description of Page

Enter the **Collection Class**, **CIS Division** and **Currency Code** to which the rules apply.

The **Algorithms** grid contains algorithms that control important functions. You must define the following for each algorithm:

- Specify the algorithm's **System Event** (see the following table for a description of all possible events).
- Specify the **Algorithm** to be executed when the System Event executes. Set the **Sequence** to **10** unless you have a **System Event** that has multiple **Algorithms**. In this case, you need to tell the system the **Sequence** in which they should execute.

The following table describes each **System Event**.

System Event	Optional / Required	Description
<i>Bill-Based Payment Arrangement</i>	Optional - only specified if your implementation uses bill-oriented payment arrangements	This algorithm is executed to handle the creation, breaking and canceling of a Bill-Oriented Payment Arrangements . Click here to see the algorithm types available for this system event.
<i>Overdue Monitor Rule</i>	Required	This algorithm is invoked by the Overdue Monitor to analyze an account's debt. Refer to How Does The Overdue Monitor Work for the details. If you have multiple rules (and therefore multiple algorithms), please take care when assigning the sequence number, as the Overdue Monitor will invoke these rules in sequence order. Click here to see the algorithm types available for this system event.
<i>Write Off Bill</i>	Option - only specified if your implementation writes-off bills	This algorithm is executed to handle the write-off and write-off reversal of a bill. Refer to Writing Off Bills . Click here to see the algorithm types available for this system event.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference [CI_OD_RULE_ALG](#).

Setting Up Overdue Event Cancellation Reasons

An overdue event cancel reason must be supplied before an overdue event can be canceled. Open **Admin Menu, Overdue Event Cancel Reason** to define overdue event cancellation reasons.

Description of Page

Enter an easily recognizable **Overdue Event Cancel Reason** and **Description** for each cancellation reason.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference [CI_OEVT_CAN_RSN](#).

Setting Up Cut Event Cancellation Reasons

A cut event cancel reason must be supplied before a cut event can be canceled. Open **Admin Menu, Cut Event Cancel Reason** to define cut event cancellation reasons.

Description of Page

Enter an easily recognizable **Cut Event Cancel Reason** and **Description** for each cancellation reason.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference [CI_C EVT_CAN_RSN](#).

Defining Prepaid Metering Options

The topics in this section describe how to set up the system to enable prepaid metering functionality.

Prepaid metering is optional. The system configuration requirements described in this section are only relevant if your organization offers prepaid metering service.

Contents

[The Big Picture of Prepaid Metering](#)
[Setting Up The System To Enable Prepaid Metering](#)

The Big Picture of Prepaid Metering

Prepaid metering allows customers to pay for energy before it is actually used.

Contents

[How Does Prepaid Metering Work?](#)
[Debit Meter vs. Credit Meter](#)
[Prepaid Metering Transactions Result in Adjustments](#)
[Interfacing Prepaid Transactions](#)
[Prepaid Transactions Can Go Into Suspense](#)

How Does Prepaid Metering Work?

A traditional electronic prepaid metering system typically operates at three levels:

- Meters that are installed at the customer's home. These meters dispense energy up to the amount that the customer prepaid.
- Vending stations located at the utility's offices or designated payment agencies. These stations/agencies sell prepaid credits to customers.
- Master stations that group vending stations together for administration, reporting and control purposes. Master stations communicate information – e.g. customer information, tariff changes, etc – to the vending stations. The vending stations, in turn, report detailed customer sales and other transaction information to the master station.

A prepaid customer goes to a vendor and purchases credits for his/her meter to dispense energy. The vendor has knowledge of the customer and the rate that the customer is paying. The vendor issues the credits in any of the following forms: a prepaid card, a token (to be inserted into the meter) or a series of numbers (to be keyed into the meter).

Debit Meter vs. Credit Meter

Prepaid meters are commonly referred to as debit meters. Billed meter types are also called credit meters. A meter's state of being 'credit' or 'debit' translates to a specific meter configuration. Having this property set at the meter configuration level enables meters to be switched between 'credit' and 'debit' if needed.

Prepaid credits may be purchased for either new premises or existing premises. In the latter case, the existing premise could initially have a typical 'credit' meter. In this case, fieldwork may be involved in getting the meter switched from 'credit' to 'debit' (prepaid).

Prepaid Metering Transactions Result in Adjustments

There could be numerous types of prepaid transactions. But the most common types are prepaid sales and cancellations.

A sale or cancellation transaction includes a total amount that usually includes tax. In order to post to the general ledger, the prepaid transaction must result in a financial transaction. Moreover, separate GL entries must be created for the breakdown of revenue and tax. In some cases, it may also be necessary to calculate the usage from the revenue amount.

[Calculated Adjustments](#) are used to book revenue from these types of transactions. A base **Generate Adjustment** algorithm is available to let rate application generate the appropriate GL calculation lines and calculate the corresponding usage, if needed.

Prepaid adjustments do not affect a customer's balance.

Interfacing Prepaid Transactions

An adjustment upload batch process exists for uploading prepaid transactions that result in adjustments. Refer to [Interfacing Adjustments From External Sources](#) for more information.

Prepaid metering options need only be set up if your organization offers prepaid metering service to your customers. Refer to [Defining Prepaid Metering Options](#) for more information.

Prepaid Transactions Can Go Into Suspense

A prepaid transaction goes into suspense if a prepaid SA could not be determined for any of the following reasons:

- The badge number on the transaction is not a valid meter in the system
- The badge number on the transaction is a valid meter in the system, but is:
 - Not linked to a prepaid meter configuration type
 - Not linked to a non-closed, non-canceled prepaid SA

This can be a common occurrence because of the likelihood that transactions get uploaded prior to meter installation/exchange or service agreement information being updated in the system.

When this situation happens, you probably still want to recognize the revenue by posting an adjustment to a suspense SA that you designate. Refer to the base sample **Get Prepaid SA Using Badge Number (C1-SABYBADGE)** algorithm for an example of how the suspense SA is specified.

A batch process exists for automatically resolving suspense adjustments. Refer to [How Are Suspense Adjustments Resolved](#) for more information.

Setting Up The System To Enable Prepaid Metering

The following sections describe the steps in setting up control information for prepaid metering.

Contents

- [Meter Types](#)
- [Characteristic Types](#)
- [Algorithms](#)
- [Rate](#)
- [Adjustment Types](#)
- [Adjustment Type Profiles](#)
- [Service Agreement Types](#)
- [Bill Cycle](#)
- [Vendor Information](#)

Meter Types

Prepaid meters are set up just like any other meter - i.e. the meter must reference a meter type and the meter's configuration must reference a meter configuration type.

To set up prepaid meter types you must do the following:

- Define the meter configuration types that will be used for recording prepaid usage. Mark each of these configuration types as **Prepaid**. Define one register with the appropriate UOM (e.g. **KWH**). See [Setting Up Meter Configuration Types](#) for more information.
- Define the meter types that will be used for prepaid metering. For each meter type, associate one of the **Prepaid** meter configuration types that you defined. You can also mark each meter type as **Prepaid Capable**. See [Setting Up Meter Types](#) for more information.

Note. Base logic does not require the meter type to be marked as **Prepaid Capable**. The **Prepaid** indicator on the meter configuration type is used to identify prepaid meters.

Characteristic Types

The following characteristic types are needed if you are going to upload prepaid transactions as adjustments and need the system to determine the prepaid SA given a prepaid meter's badge number.

Refer to [Setting Up Characteristic Types](#) for more information.

Contents

- [Prepaid Entity Characteristic Type](#)
- [Badge Number Characteristic Type](#)

Prepaid Entity Characteristic Type

Create a characteristic type that identifies whether an SA Type is used for prepaid metering. For example:

- Characteristic Type = <code>

- Description = **Prepaid Entity**
- Char Entities = **SA Type**
- Subclass = **predefined list**
- Values = **Y, N**

Badge Number Characteristic Type

Create a characteristic type for specifying a badge number. For example:

- Characteristic Type = <code>
- Description = **Badge Number**
- Char Entities = **Adjustment, Adjustment Type**
- Subclass = **Adhoc**

Algorithms

The following algorithms need to be set up in order to store prepaid transactions in the system.

Refer to [Setting Up Algorithms](#) for more information.

Note. The following sections describe basic set up needed for uploading 'sale' and 'cancellation' types of prepaid transactions. Your implementation team may have to define additional specific algorithms types for any other prepaid transaction type that you need to upload into or store in the system.

Contents

[Rate Component - Calculation Algorithm](#)
[Adjustment Type - Generate Adjustment](#)
[Adjustment Type - Determine SA](#)
[Adjustment Type - Resolve Suspense](#)
[Adjustment Type - Adjustment Information](#)
[Installation - Adjustment Information](#)

Rate Component - Calculation Algorithm

Create an algorithm of type **Determine Percent Given Total (C1-PCTGVNTOT)** specifying your parameter values for UOM / TOU / SQL.

Create an algorithm of type **Back Into Revenue (C1-BACKINREV)** specifying your parameter values for UOM / TOU / SQL.

Adjustment Type - Generate Adjustment

Create an algorithm of type **Adjustment Generation - Apply Rate (ADJG-RT)**, specifying your [prepaid rate](#) and parameter values for UOM / TOU / SQL.

Adjustment Type - Determine SA

Create an algorithm of type ***Get Prepaid SA Using Badge Number (C1-SABYBADGE)***, specifying your badge number characteristic type, prepaid characteristic type and suspense SA ID (i.e. SA to which the adjustment needs to post when a valid prepaid SA is not found).

Adjustment Type - Resolve Suspense

Create an algorithm of type ***Cancel Suspense Adjustment (C1-CANSUSADJ)*** specifying the following:

- [Badge number characteristic type](#)
- [Prepaid entity characteristic type](#)
- [Prepaid adjustment type](#)
- Adjustment cancel reason

If you want this algorithm to create to do entries for adjustments that have been in suspense for too long, specify additional parameter values for number of days and to do type and to do role (optional).

Adjustment Type - Adjustment Information

Specify the SUSPENSE_DESCR parameter on your adjustment information algorithm if you want the information to include an indication of suspense.

Installation - Adjustment Information

Specify the SUSPENSE_DESCR parameter on your adjustment information algorithm if you want the information to include an indication of suspense.

Rate

In order to generate separate calculation lines for the revenue and tax breakdown of the prepaid amount, you need to set up a prepaid rate with components that calculate revenue amount and tax amount given just a total amount.

Create a rate schedule with the following components:

- Tax Rate Component
 - Rate Component Type = ***Calculation Algorithm***
 - Value Type = ***Percentage***
 - Value Source = ***Bill Factor***
 - Bill Factor = your tax bill factor
 - Calculation Algorithm = the one created for algorithm type ***C1-PCTGVNTOT***. Refer to [Rate Component - Calculation Algorithm](#) for information.
 - Distribution Code = *your distribution code for tax liability*
- Revenue Rate Component
 - Rate Component Type = ***Calculation Algorithm***

- Derive SQ = checked
- Value Type = **Unit Charge**
- Value Source = **Bill Factor**
- Bill Factor = your rate/kWh bill factor
- UOM/TOD/SQI to use for the resulting calculated usage, for example **KWH**
- RC Cross Reference = indicate the sequence for the tax rate component (above)
- Calculation Algorithm = the one created for algorithm type **C1-BACKINREV**. Refer to [Rate Component - Calculation Algorithm](#) for information.
- Distribution Code = your distribution code for revenue

Adjustment Types

You must define an adjustment type for each type of transaction that you want to store in the system. Refer to [Setting Up Adjustment Types](#) for more information.

For prepaid sales and cancellation transactions, you need to define [calculated adjustment types](#). In addition, plug-in the following algorithms:

- **Adjustment FT Creation - GL Only** algorithm - so that transaction amounts do not affect the customer's balance. Specify calc lines as the source for adjustment distribution codes.
- [Generate Adjustment](#)
- [Determine SA](#)
- [Resolve Suspense](#), if applicable.
- [Adjustment Information](#), if applicable

Adjustment Type Profiles

Create appropriate [adjustment type profiles](#) for your prepaid service agreements.

Service Agreement Types

You must create a prepaid [SA Type](#) for each unique CIS Division. Each SA Type must be set up as non-billable and requiring a characteristic premise. The SA Types must also indicate that they are used for prepaid by specifying a [prepaid entity characteristic](#) value.

Bill Cycle

Create at least one bill cycle that will be used for prepaid accounts. This bill cycle should not have a schedule defined.

Vendor Information

A vendor could be a vending station, payment agency or master station.

You must create a [person](#) for each of your vendors. Information about the vendor may include names, an address, phone numbers, etc.

Any other miscellaneous information about the vendor can be stored as characteristics on the person.

Hierarchical relationships between vendors can be established through characteristics. For instance, a vending station's person record may contain a foreign key characteristic that points to the master station that the vending station reports to.

Conservation Programs

Oracle Utilities Customer Care and Billing allows you to define conservation (or energy efficiency) programs and provide rebates to customers.

Contents

- [The Big Picture of Conservation Programs](#)
- [GL Accounting Example](#)
- [Setting Up Conservation Programs](#)

The Big Picture of Conservation Programs

The purpose of using conservation programs is to provide rebates to customers based on eligibility and verification of newly purchased appliances and hardware that are rated to conserve the demand for energy. To redeem their rebates, customers have to submit a rebate application to the utility with receipts, and the utility has to administer and report on the programs.

Admin and transaction objects are provided in the product to support the definition of conservation programs (admin data) and the subsequent rebate claims (transaction data). Portals and BPA scripts are used to maintain the conservation programs and rebate claims. Adjustments are used to recognize the expense and to process refunds.

The following sections discuss the maintenance objects that support this functionality.

Contents

- [Conservation Program Maintenance Object](#)
- [Conservation Program Rebate Definition Maintenance Object](#)
- [Rebate Claim Maintenance Object](#)
- [Rebate Claim Line Maintenance Object](#)

Conservation Program Maintenance Object

A conservation program is an admin maintenance object (MO) used to support the definition of a conservation program. It holds rules that control how the claims for a conservation program are managed. If your organization wishes to use this MO, you can either use the business object (BO) supplied in the base product or configure your own BO.

This MO provides the following functionality:

- A business object option **Display Statistics Service Script** is provided for business objects of this type. The script plugged into this option retrieves information displayed on the **Conservation Program Statistics Zone**.
- A business object option **Display Statistics UI Map** is provided for business objects of this type. This is the map used on the **Conservation Program Statistics Zone** to display statistics.
- A separate maintenance object is provided to capture rebate definitions for a conservation program
- Logs are not provided.
- The standard characteristics collection is provided.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference [CI_CONSV_PROG](#).

For more information, about this MO and to review the business objects defined for this MO, navigate to **Admin, Maintenance Object** and view the MO **C1-CPROG**.

Conservation Program Rebate Definition Maintenance Object

A conservation program has rebate definitions that define the rebate amounts and energy savings for different types of products. The superset of such lines is the "rebate matrix". Each row in the matrix defines the rebate and presumed energy savings for a product purchased by the consumer. If your organization wishes to use this maintenance object (MO), you can either use the business object (BO) supplied in the base product or configure your own BO.

This MO provides the following functionality:

- The Rebate Definition table can be used to capture information for rebate claim lines such as, appliance type, refund amount, energy savings, and applicable manufacturers and models.
- Logs are not provided for the Rebate Definition.
- A characteristics collection is not provided.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference [CI_REBATE_DEFN](#).

For more information, about this MO and to review the business objects defined for this MO, navigate to **Admin, Maintenance Object** and view the MO **C1-RDEF**.

Rebate Claim Maintenance Object

When a customer files a claim for a rebate, a rebate claim will be created. The rebate claim maintenance object (MO) is used to support the definition of a claim. If your organization wishes to use this MO, you can either use the business object (BO) supplied in the base product or configure your own BO.

This MO provides the following functionality:

- Depending on the utility's requirements, you can specify single or multiple items on the Rebate Claim. A separate maintenance object is provided to capture rebate claim lines for a rebate claim.
- The standard characteristics collection is provided.
- A log is provided. This can be used to track approval information for the claim, to capture adjustments created as the claim is processed, etc.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference [CI_REBATE_CLAIM](#).

For more information, about this MO and to review the business objects defined for this MO, navigate to **Admin, Maintenance Object** and view the MO **C1-RCLAIM**.

Rebate Claim Line Maintenance Object

A rebate claim has a rebate claim line for each product eligible for a refund. You use the rebate claim line maintenance object (MO) to create a rebate line. If your organization wishes to use this MO, you can either use the business object (BO) supplied in the base product or configure your own BO.

This MO provides the following functionality:

- SA (service): The SA table is the real service SA. There is a separate SA under which the rebate adjustments are stored.
- Logs are not provided for Rebate Claim Line
- The standard characteristics collection is provided.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference [CI_REBATE_LINE](#).

For more information, about this MO and to review the business objects defined for this MO, navigate to **Admin, Maintenance Object** and view the MO **C1-REBLN**.

GL Accounting Example

The following table shows the financial transactions that are issued when processing energy conservation rebates. This example shows the financial transactions when a customer files a claim for a dishwasher and insulation.

Notice how the expense adjustments are atomized:

- Separate adjustments are created to expense each item individually.
- A single adjustment is created for the A/P check request.

Event	GL Accounting
Rebate adjustment is created - dishwasher	Rebate Expense 35 A/P <35>
Rebate adjustment is created - insulation	Rebate Expense 45.45 A/P <45.45>
A/P check request issued	A/P 80.45 Cash <80.45>

Setting Up Conservation Programs

Conservation Programs allow administrators to create and maintain conservation programs for which customer can submit rebate claims. To set up a conservation program, open **Admin Menu**, **Conservation Program**.

For additional information on working with rebate claims, see [Rebate Claims](#).

The topics in this section describe the base-package zones that appear on the Conservation Program portal.

Contents

- [Conservation Program List Zone](#)
- [Conservation Program Zone](#)
- [Rebate Definition Zone](#)
- [Rebate Claim Statistics Zone](#)

Conservation Program List Zone

The Conservation Program [List zone](#) lists every conservation program. The following functions are available:

- Click a [broadcast](#) button to open other zones that contain more information about the adjacent conservation program.
- Click the **Add** link in the zone's title bar to add a new conservation program.

This zone displays the following for each conservation program defined in the system:

- Conservation Program Information (Description, Status, Start Date, End Date)
- The number of Complete Claims
- The total Complete Claims Amount
- The number of Incomplete Claims
- The Statistics Date/Time is the date that the statistics were last calculated

Use the following procedure to create a new conservation program:

- Select **Admin Menu, Conservation Program +** or click **Add** on the title bar of the Conservation Program [List zone](#).
- Enter basic information about the conservation program in the **Main** section, including:
 - Conservation Program code
 - Description
 - Start Date/End Date
 - Statistic Update Frequency (in hours)
- Enter **Financial Information** about the conservation program, including:
 - CIS Division
 - SA Type

- A/P Adjustment Type
- A/P Taxable Adjustment Type
- Enter **Approval Information** about the conservation program, including:
 - Approval To Do Type
 - Duplicate Claim To Do Role
 - Approval Levels and Threshold Amounts
- Click **Save**. To return to the Conservation Program portal without saving the new program, click **Cancel**.

To view a specific conservation program, click the broadcast icon for the conservation program you wish to view. The remaining zones in the Conservation Program portal open displaying details about the selected conservation program.

Conservation Program Zone

You use the Conservation Program zone to view and maintain individual conservation program. The Conservation Program zone displays the following information about the selected conservation program:

- Basic information about the conservation program, including
 - Conservation Program
 - Description
 - Start Date/End Date
 - Statistic Update Frequency (in hours)
 - Status
- Financial Information about the conservation program, including
 - CIS Division
 - SA Type
 - A/P Adjustment Type
 - A/P Taxable Adjustment Type
- Approval Information about the conservation program, including
 - Approval To Do Type
 - Duplicate Claim To Do Role
 - Approval Levels and Threshold Amounts

Please see the zone's help text for information about this zone's fields.

Contents

- [Conservation Program Status](#)
- [Conservation Program Actions](#)
- [Editing Conservation Programs](#)
- [Deleting Conservation Programs](#)

[Activating Conservation Programs](#)
[Deactivating Conservation Programs](#)
[Setting the Status of Conservation Programs to Pending](#)

Conservation Program Status

The Status of a conservation program indicates the current state of the program within the system. Valid statuses include:

- **Pending** indicates the program is pending. This is the initial state of a conservation program when first created.
- **Active** indicates the conservation program is currently active.
- **Inactive** indicates the conservation program is currently inactive.

Conservation Program Actions

You can perform a number of actions on a conservation program, including:

- **Edit:** Used to edit a conservation program
- **Delete:** Used to delete a conservation program
- **Activate:** Used to activate a pending or inactive conservation program
- **Refresh Statistics:** Used to refresh the statistics of an active conservation program
- **Deactivate:** Used to deactivate a pending or active conservation program
- **Pend:** Used to change the status of an active or inactive conservation program to Pending

The actions available are based on the current status of the conservation program. The table below summarizes the actions available at each status.

Status	Valid Action
Pending	Edit, Delete, Activate, Deactivate
Active	Edit, Refresh Statistics, Deactivate, Pend
Inactive	Activate, Pend

Editing Conservation Programs

Use the following procedure to edit an active or pending conservation program:

- Click **Edit**.
- Edit the details of the conservation program as needed.
- Click **Save**.

Deleting Conservation Programs

Use the following procedure to delete a conservation program:

- Click **Pend** to change the status of the conservation program to Pending (if needed).
- Click **Delete**.

- Click **OK** on the Confirm Delete dialog. To close the dialog without deleting the conservation program, click **Cancel**.

Activating Conservation Programs

To activate an inactive or pending conservation program, click **Activate**.

Deactivating Conservation Programs

To deactivate an active or pending conservation program, click **Deactivate**.

Setting the Status of Conservation Programs to Pending

To set the status of an active or inactive conservation program to Pending, click **Pend**.

Rebate Definition Zone

You use the Rebate Definition zone to add, view, and edit rebate definitions associated with a conservation program. A rebate definition defines the types of rebates allowed for a specific conservation program. The Rebate Definition zone displays the following details for each claim line:

- Rebate Definition
- Status (Active or Inactive)
- Rebate Amount per Unit
- Icons to Edit, Delete, and Activate/Deactivate rebate definitions

Contents

[Adding Rebate Definitions](#)
[Editing Rebate Definitions](#)
[Deleting Rebate Definitions](#)
[Activating Rebate Definitions](#)
[Deactivating Rebate Definitions](#)

Adding Rebate Definitions

Use the following procedure to add a rebate definition:

- Click **Add** in the title bar of the Rebate Definition zone.
- Enter a Description of the rebate definition.
- Select the **Product** for the rebate definition from the drop-down list.
- Select the **Service Type** for the rebate definition from the drop-down list.
- Enter the **Rebate Per Unit Amount** for the rebate definition.
- Enter the **Rebate Unit of Measure** for the rebate definition.
- Select the **Expense Adjustment Type** for the rebate definition from the drop-down list.
- Enter the **Presumed Energy Savings** for the rebate definition.

- Select the **Presumed Energy Savings** Unit of Measure for the rebate definition from the drop-down list.
- Enter the **Manufacturer** and **Model** for each item eligible to be submitted as a rebate claim line.
- To upload manufacturer/model information from a comma-separated-values file, click **CSV File to Upload**.
 - Click **Browse** on the File Upload dialog, and browse to the file to be uploaded.
 - Click **Upload**.
- Click **Save**. To return to the Rebate Definition zone without adding the rebate definition, click **Cancel**.

Editing Rebate Definitions

Use the following procedure to edit a rebate definition:

- Click the edit icon for the rebate definition you wish to delete.
- Edit the details of the claim line as appropriate.
- Click **Save**. To return to the Rebate Definition zone without changing the rebate definition, click **Cancel**.

Note: You can edit a rebate definition only when it is Active.

Deleting Rebate Definitions

Use the following procedure to delete a rebate definition:

- Click the delete icon for the rebate definition you wish to delete.
- Click **OK** on the Confirm Delete dialog. To close the dialog without deleting the rebate definition, click **Cancel**.

Activating Rebate Definitions

To activate an inactive rebate definition, click the **Activate** button for the rebate definition you wish to activate.

Note: You can only activate rebate definition that is currently Inactive.

Deactivating Rebate Definitions

To deactivate an active rebate definition, click the **Deactivate** button for the rebate definition you wish to deactivate.

Note: You can only deactivate rebate definition that is currently Active.

Rebate Claim Statistics Zone

You use the Rebate Claim Statistic zone to view statistics for submitted rebate claims based on the current conservation program. This zone displays the following:

- Conservation Program
- Statistics Date/Time
- Statistics Charts

Statistics Graphs

This zone displays statistics for submitted rebate claims using the following charts:

- **Statistics by Status:** A pie chart that displays the number and percentage of claims for each status.
- **Statistics by Month:** A line chart that displays the number of claims completed each month.
- **Statistics by Product:** A pie chart that displays the total value, percentage, and number of claim lines completed for each product.
- **Statistics by Claim Age:** A line and bar chart that displays the number of new claims per month (line graph) and the number of claims per age grouping (bar graph).

To refresh statistics, click **Refresh Statistics** on the title bar of the Rebate Claim Statistics zone.

Defining Batch Schedule Options

The topics in this section describe how to set up the system to periodically execute batch job streams.

The batch scheduler is optional. Setting up the batch scheduler is only necessary if your organization uses the product's batch scheduler. If your organization uses a third party tool to manage the periodic execution of batch jobs, this section is not applicable.

Separate module. Please note that batch scheduler functionality is associated with separate **Workflow Scheduling** module. If this module is not applicable to your business you may turn it off. Refer to [Turn Off A Function Module](#) for more information.

Contents

[The Big Picture of Scheduling Batch Jobs](#)
[Setting Up The Batch Scheduler](#)
[Maintaining Job Stream Creation Schedules](#)
[How To Copy Job Streams From The Demonstration Database](#)

The Big Picture of Scheduling Batch Jobs

The following points provide an overview of how batch jobs are scheduled to run periodically.

Contents

[Job Stream, a Definition](#)
[A Workflow Process Is Created Each Time A Job Stream Executes](#)
[A Workflow Process Template Defines The Batch Jobs In A Job Stream](#)
[How To Start A Job Stream](#)
[Activating A Workflow Event Causes A Batch Run To Be Submitted](#)
[Workflow Event Status Reflects The Status Of The Batch Run](#)
[Technical Implementation Of The Batch Scheduler](#)

Job Stream, a Definition

The term "job stream" refers to a suite of batch jobs that run periodically. Most organizations have multiple job streams. For example,

- You'll have a job stream that contains batch jobs that run nightly
- You'll have a different job stream that contains the hourly batch jobs
- You'll have a different job stream that contains the batch jobs that extract data for your data warehouse
- Etc.

Refer to [Batch Process Dependencies](#) for a description of sample job streams.

A Workflow Process Is Created Each Time A Job Stream Executes

The system creates a [workflow process](#) each time a job stream executes. The workflow process has a separate workflow event for each batch job in the job stream. The batch jobs are submitted when the workflow process's events are activated (i.e., each workflow process event submits a specific batch job).

A Workflow Process Template Defines The Batch Jobs In A Job Stream

The system creates a job stream's workflow process using a [workflow process template](#). This means that a separate workflow process template exists for each job stream. For example, there is a workflow process template for the nightly job stream and another for the weekly job stream, etc.

There are typically dependencies between a job stream's batch jobs. For example, the billing batch job might be dependent on the successful execution of the payment upload batch job. Dependencies between batch jobs are defined by setting up workflow event dependencies on the workflow process. For example, if the billing batch job should only run after the payment upload batch job completes, you'd set up the workflow event that submits the billing job to be dependent on the completion of the event that submits the payment upload job. Note, you can define multiple dependencies between batch jobs (i.e., you can indicate that both the meter read upload and payment upload must complete before billing starts).

Once you define your job streams using workflow process templates, you indicate your job streams in the [batch scheduler feature configuration](#). The [job stream summary](#) page and the [job stream creation schedule](#) use this information to display the appropriate job stream templates.

Importing workflow process templates from the demonstration database. Refer to [How To Copy Job Streams From The Demonstration Database](#) for a description of how to copy sample workflow process templates from the demonstration database.

How To Start A Job Stream

You can manually start a job stream using the [Job Stream Summary](#) page.

Optionally, you can [set up a schedule](#) defining when the system should start a job stream (i.e., create a workflow process). For example, you can set up a schedule for the "nightly" workflow process template to indicate that a workflow process should be created every night at 5:30 pm, Monday through Friday.

- You can monitor the status of your job streams on the [Job Stream Summary](#) page.
- You can monitor the status of a specific execution of a job stream on the [Job Stream Details](#) page. This page shows the status of the events on a workflow process and a summary of the execution status of each event's batch job.

Refer to [Technical Implementation Of The Batch Scheduler](#) for more information.

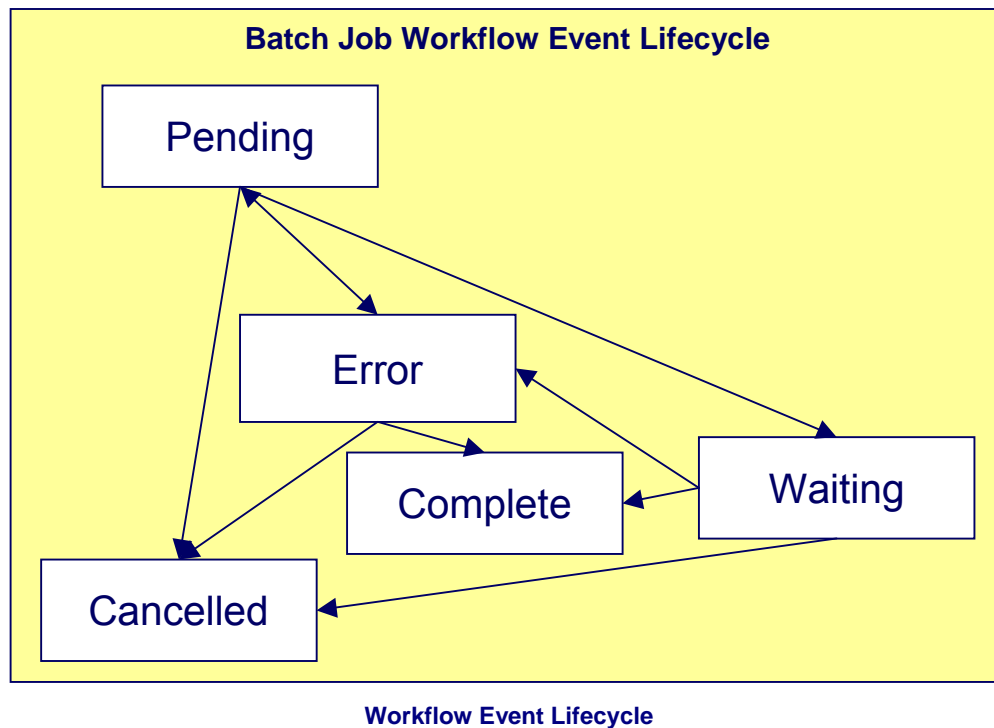
Activating A Workflow Event Causes A Batch Run To Be Submitted

When a workflow event is activated, its activation algorithm creates a **Pending** batch run request. To be specific - the workflow event's **Activation** algorithm defines the batch process and the [number of parallel threads](#) that should be executed when the batch job is submitted. Please see the activation algorithm's parameters for a description of exactly how the **Pending** batch run request is created.

What activates workflow events? A batch process exists that is responsible for activating the workflow events associated with your job streams' workflow processes. This batch process is known as **C1-WFSUB**. It should be noted that when this process is submitted, it only activates the workflow events associated with a specific workflow process template (the workflow process template is identified by a parameter supplied to **C1-WFSUB**). You may wonder why the standard **WFET** batch process (i.e., the standard workflow event activator) doesn't process the batch scheduler workflow events. The reason is that **C1-WFSUB** is executed many times during the processing of your job streams and we didn't want to clutter up the run statistics for **WFET** with the myriad executions of **C1-WFSUB**.

Workflow Event Status Reflects The Status Of The Batch Run

The system creates a [workflow process](#) each time a job stream executes. The workflow process has a separate workflow event for each batch job in the job stream. The batch jobs are submitted when the workflow process's events are activated (i.e., each workflow process event submits a specific batch job). The following diagram shows the potential state of these workflow events:



The following points explain the relationship between a workflow event's status and the state of the corresponding batch job that it submits:

- Workflow events are initially created in the **Pending** state. The event's batch job has not been submitted when it's in this state.
- When the workflow event is activated, its activation algorithm attempts to submit a request to execute a batch job:
 - If there is something wrong with the activation algorithm's parameters, the event will enter the **Error** state. If this happens, you can:
 - Resubmit the batch job by changing the event's state back to **Pending**.
 - Cancel the batch job by changing the event's status to **Canceled**.
 - Skip this batch job by changing the event's status to **Complete**. Do this if the subsequent dependent batch jobs should proceed despite these errors.
 - If the batch run is submitted successfully, the workflow event enters the **Waiting** state (it is waiting for the batch job to complete).
- When the batch job completes, the workflow event transitions into either the **Complete** or **Error** state:
 - If the batch run aborts due to too many errors, it transitions into the **Error** state. If this happens, you can:
 - Restart a batch job that aborted by changing the event's status back to **Pending**.
 - Cancel the batch job by changing the event's status to **Canceled**.
 - Skip this batch job by changing the event's status to **Complete**. Do this if the subsequent dependent batch jobs should proceed despite these errors.
 - If the batch run doesn't abort, due to too many errors, it transitions to the **Complete** state.
- You can **Cancel** a **Pending** event if you don't want the batch job to be submitted.
- A **Pending** event will be **Canceled** automatically by the system if the workflow process is canceled by a user.
- A **Waiting** event will be **Canceled** automatically by the system if the workflow process is canceled by a user.

Note. You can monitor the status of a workflow process's events and their related batch jobs on the [Job Stream Details](#) page.

Technical Implementation Of The Batch Scheduler

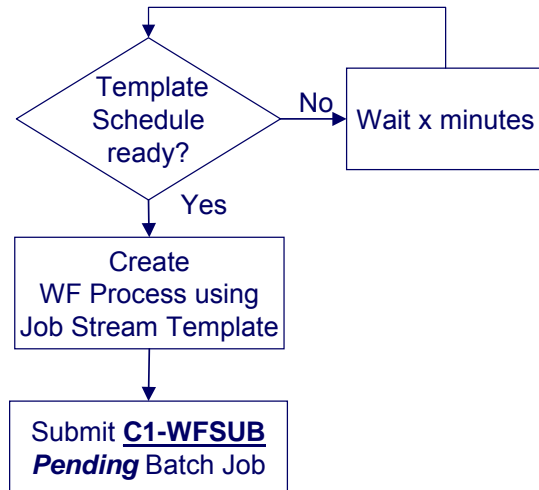
The topics in this section provide information about how your job streams are executed. This section is intended for your technical personnel who are responsible for configuring the program that periodically submits batch jobs.

Contents

[A PERL Program Determines If There's Work To Do](#)
[Completing A Batch Program That's Part Of A Job Stream](#)
[Polling Frequency](#)

A PERL Program Determines If There's Work To Do

A PERL program runs in the background looking for job stream (workflow process) templates whose [schedule](#) is ready to be processed. The following flowchart provides a schematic of its logic:



When **C1-WFSUB** executes, it causes the workflow events to activate. The activation plug-in on these workflow events creates **Pending** Batch Jobs

The following points summarize important concepts illustrated in the flowchart:

- When the job stream template [schedule](#) indicates it's time to start a job stream, a workflow process is created by copying the event types from the job stream (workflow process) template.
- In addition, a **Pending** batch job that activates the workflow process's events is created (i.e., the **C1-WFSUB** batch job is submitted).
- When **C1-WFSUB** executes, it activates the workflow process's events. The activation plug-in of these workflow events creates the job stream's **Pending** batch jobs. In addition, the activation algorithm transitions the workflow events into the **Waiting** state (they are waiting for the batch job to complete).

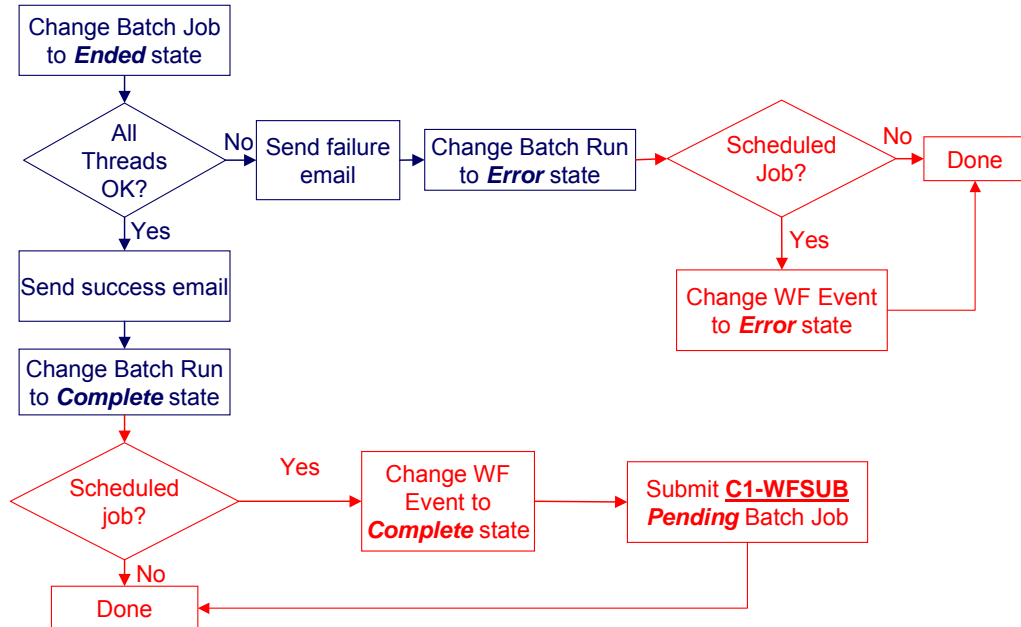
Note. The identity of **C1-WFSUB** is not hard-coded. Rather, it's defined on the batch scheduler's [Feature Configuration](#).

The same PERL program described above looks for **Pending** batch jobs and then executes them for the standard batch job submission functionality. Refer to [Technical Implementation of the Online Batch Submission](#) for more information.

Number of Threads. For batch jobs related to a job stream, the number of parallel threads to execute is defined on a parameter on the workflow event's activation algorithm.

Completing A Batch Program That's Part Of A Job Stream

When a batch program completes, a common routine is called. Refer to [A Common Routine Is Invoked When Batch Programs Complete](#) for information about common batch submission logic related to the process. The following flowchart provides a schematic of this routine's logic where the additional logic applicable to the batch scheduler is highlighted in red.



The following points summarize important concepts regarding the batch scheduler logic illustrated in the flowchart:

- If at least one thread fails and this batch job is one related to a scheduled job stream, the status of the corresponding workflow event is changed to **Error**.
- If all threads are successful, and this batch job is one related to a scheduled job stream,
 - The status of the corresponding workflow event is changed to **Complete**.
 - In addition, a **Pending** batch job is submitted to activate workflow events that are dependent on the completion of a batch job. When this job executes, the "downstream" batch jobs will be submitted (and then the logic shown above starts again).

Batch Run Number. The routine also creates a characteristic for the workflow event to capture the batch run number associated with this batch run. The characteristic type to use is defined on the [feature configuration](#).

Polling Frequency

Usage of the batch scheduler requires that your system administrator create a job in the operating system to periodically execute the PERL program described above. It is important to schedule the execution of this PERL program very frequently (e.g., every five minutes) in order to avoid lengthy time gaps between the completion of a batch job and the triggering of the dependent jobs.

Refer to [online batch submission polling frequency](#) for more information about configuring this setting.

Setting Up The Batch Scheduler

The following topics summarize how to set up the system to enable the scheduling of job streams.

Contents

- [Create A Workflow Event Type - Activation Algorithm For Every Batch Job](#)
- [Create A Workflow Event Type For Every Batch Job](#)
- [Create A Workflow Process Template For Every Job Stream](#)
- [Set Up A Work Calendar](#)
- [Set Up A Job Stream Creation Schedule For Each Job Stream](#)
- [Set Up Characteristic Types](#)
- [Set Up A User ID For Batch Jobs](#)
- [Set Up A Feature Configuration](#)

Create A Workflow Event Type - Activation Algorithm For Every Batch Job

As described under [The Big Picture of Scheduling Batch Jobs](#), the system creates a workflow process when a job stream is initiated. The workflow process's events cause **Pending** batch jobs to be created when the workflow events are activated. .

A base-package Workflow Event Type - Activation algorithm exists that creates a **Pending** batch job (the identify of the batch job is defined as a parameter on the algorithm). This means that you need to set up a **Workflow Event Type - Activation Algorithm** for every batch job.

Besides defining the batch job's [batch control](#) code, the base-package algorithm, [C1-WFAC-CRBj](#) also allows you to define the following values on each algorithm:

- **The number of parallel threads that should be submitted.** If this parameter is left blank, the batch job is submitted using a single thread. If a value greater than one is defined, the system automatically initiates this number of parallel threads. For example, if you want to kick the **BILLING** batch process using 20 parallel threads, specify a value of 20 in the respective algorithm's parameters. Refer to [Optimal Thread Count for Parallel Background Processes](#) for more information of executing a batch process in parallel threads.
- **The override email address used to inform that a batch job has finished.** If this parameter is specified, email is sent to this address when a batch job completes informing the recipient of its successful completion (or failure). Note, this address overrides the email address defined on the [Feature Configuration](#) for the batch scheduler (meaning that you need only populate this parameter if you want the email to be sent to someone other than the global email recipient defined on the Feature Configuration).
- **The value of all batch job-specific parameters.** Some batch jobs have job-specific parameters. For example, the Account Debt Monitor (ADM) batch process has a parameter that controls if calendar or workdays should be used when calculating dates. Values for the job-specific parameters may be defined on the batch code. In addition, override values may be defined on the algorithm's parameters. For example, you can define if calendar days or workdays are used when you set up the algorithm that submits the Account Debt Monitor batch process.

Importing saves time. If you [import job streams](#) from the demonstration database, the import process will set up a separate Workflow Event Type - Activation Algorithm for every batch job in the system. You need only add additional algorithms if your implementation develops new batch processes.

Demonstration data and parameter values. Please note that the workflow event type activation algorithms provided in the demonstration database do not define any job-specific parameter values. In addition, most batch control records in the demonstration database do not define default job-specific parameter values. Your organization must determine the appropriate values for your business and define these values accordingly.

Create A Workflow Event Type For Every Batch Job

As described under [The Big Picture of Scheduling Batch Jobs](#), the system creates a workflow process when a job stream is initiated. The workflow process's events reference workflow event types. A separate workflow event type must be set up for every batch job (and you must reference the appropriate activation algorithm on each workflow event type).

Importing saves time. If you [import job streams](#) from the demonstration database, the import process will set up a separate Workflow Event Type for every batch job in the system. You need only add additional workflow event types if your implementation develops new batch processes.

Create A Workflow Process Template For Every Job Stream

As described under [The Big Picture of Scheduling Batch Jobs](#), the system creates a workflow process when a job stream is initiated. The system uses a workflow process template to create a workflow process. This means you must create a workflow process template for every batch job.

Importing saves time. If you [import job streams](#) from the demonstration database, the import process will set up a workflow process templates for the classic job streams used in the demonstration database. We recommend that you modify imported workflow process templates to only include workflow event types for the batch jobs applicable to your implementation (the workflow process templates in the demonstration database contain a workflow event type for every possible batch job and it is unlikely that your implementation uses every possible batch job).

Set Up A Work Calendar

As described under [How To Start A Job Stream](#), you must set up a Job Stream Creation Schedule to define when each job stream should be started. You do this using the [Job Stream Creation Schedule](#) transaction. This transaction allows you to create the submission dates and times using a "recurrence schedule" (i.e., you don't have to type in 52 entries for a weekly job stream, you can rather have the system create the 52 entries for you). This feature uses a given [work calendar](#) to know what days are workdays and holidays. You must set up a work calendar (or reuse an existing work calendar) if you intend to use this feature.

Set Up A Job Stream Creation Schedule For Each Job Stream

As described under [How To Start A Job Stream](#), you must set up a [Job Stream Creation Schedule](#) to define when each job stream should be initiated. For example, you can set up a schedule to have your "nightly" job stream run at 5pm, Monday through Friday.

Set Up Characteristic Types

When the system activates a workflow event that submits a **Pending** batch job, it updates the workflow event with information about the batch job. It does this by populating characteristics on the workflow event. In order for the system to do this, you must set up the following characteristic types:

- Batch Job ID. This must be FK characteristic to the Batch Job object.
- Batch Control Code. This must be a FK characteristic to the Batch Control object.
- Batch Run Number. This must be an ad hoc characteristic.

In addition to the above workflow event characteristics, you must also set up a workflow process characteristic:

- Business Date. This must be an ad hoc characteristic. It holds the business date that's passed to the job stream's batch jobs (all batch jobs in a job stream use the same business date).

Set Up A User ID For Batch Jobs

When batch jobs execute, they must reference a given user's information for the following:

- Batch jobs can cause "auditable" events to occur. These events have a user ID associated with them.
- Some batch jobs access language-specific data, the language used is defined on a given user.
- When batch jobs complete, an email message is sent to a user's email address.

When a batch job is submitted, it references a specific user ID; this user ID is used for the above points. We recommend setting up a "dummy" user for this purpose (e.g., user ID = **BATCH**).

Set Up A Feature Configuration

After completing the above set up tasks, you must set up a [Feature Configuration](#) to provide the batch scheduler with the information it needs to operate.

The following points describe the various **Option Types** that must be defined for the feature configuration:

- **Active Job Stream (WF Template Code).** Set up a separate option for every job stream. The option's value is the Workflow Process Template associated with the job stream. Note, only these job streams appear on the [Job Stream Summary](#) page and are available in the [Job Stream Creation Schedule](#) page.
- **Batch Code FK Char Type.** Set up a single option to define the foreign key characteristic type used to reference the batch code on the workflow events. Refer to [Set Up Characteristic Types](#) for more information.
- **Batch Code of WF Event Activation Process.** Set up a single option to define the batch code of the batch job that is responsible for [activating the workflow events](#) that cause **Pending** batch jobs to be created.
- **Batch Job ID FK Char Type.** Set up a single option to define the foreign key characteristic type used to reference the batch job on the workflow events. Refer to [Set Up Characteristic Types](#) for more information.
- **Batch Run Number Ad hoc Char Type.** Set up a single option to define the ad hoc characteristic type used to reference the batch run's run number on the workflow events. Refer to [Set Up Characteristic Types](#) for more information.
- **Business Date Ad hoc Char Type.** Set up a single option to define the ad hoc characteristic type used to reference the batch run's business date on the workflow process (a single business date is used for all batch jobs initiated by a job stream). Refer to [Set Up Characteristic Types](#) for more information.
- **Email Address on Batch Jobs.** Set up a single option to define the email address used to notify a user when a batch job completes (note, the Email's subject line indicates if the batch job failed or succeeded). Note, you can override the email address on a specific batch job using a parameter on the Workflow Event Activation algorithm used to create the **Pending** batch job. If you leave this field blank (and no override is defined for the batch job), no email will be sent.
- **SMTP Server Name** and **SMTP Port Number.** If you want email sent informing of the completion of a batch job, define the SMTP server and port number.
- **User ID on Batch Jobs.** Set up a single option to define the user ID controlling user-oriented functionality. Refer to [Set Up A User ID For Batch Jobs](#) for more information.
- **Work Calendar.** Set up a single option to define the work calendar used when Workflow Process Recurrence Schedules are created. Refer to [Set Up A Work Calendar](#) for more information.

Maintaining Job Stream Creation Schedules

As described under [How To Start A Job Stream](#), you can set up a job stream creation schedule to define when a job stream (workflow process) should be automatically created by the system. Open **Admin Menu, Job Stream Creation Schedule** to set up a job stream creation schedule.

Description of Page

On this page, you define when the system should automatically create a workflow process for a job stream template. To do this:

- Enter the **Job Stream Template**.

Note. Only job streams defined on the [Feature Configuration](#) for the Batch Scheduler appear on this page.

- Enter the **Dates and Times** on which the system should automatically create the workflow process.

When a schedule record is added, the system sets its status to **Pending**.

Click the **Hold** button to change a **Pending** schedule to **Hold** if you don't want the system to create a workflow process on the date and time. You'd hold a record rather than delete it if you intend to release the hold in the near future.

Click the **Release** button to change a **Held** schedule to **Pending** if you want to release the hold.

When the system creates a workflow process for a job stream schedule record, the schedule record's status is changed to **Process Created**. There is one exception to this statement - if multiple **Pending** schedule records exist for a given workflow process template, the system will create a single workflow process for the one with the latest date / time and mark all others as **Skipped Due To Overlap**.

Schedule records in the **Process Created** and **Skipped Due To Overlap** states are protected.

Rather than entering the individual Dates and Times, you can press the **Recurrence** button to have the system generate these for you. Pushing this button causes the **Recurrence Window - Daily Pattern** to open.

The **Range From** and **To** define the dates during which recurring entries will be created.

The **Pattern** defines the frequency of recurrence:

- The **Daily** pattern will create a schedule for every date in the range defined above (subject to exceptions defined below). You can define the following for this pattern:
 - Use **Repeat Every** to define if the schedule should be created every day (a value of **1**), every other day (a value of **2**), every third day (a value of **3**), etc.
 - Enter the **Start Time** defined on the schedule records.
 - Turn on **Include Weekends** if schedule records should also be created on weekends.
- The **Hourly** pattern will create a schedule for every hour in the date range defined above (subject to exceptions defined below). You can define the following for this pattern:
 - Use **Repeat Every** to define if schedule records should be created every hour (a value of **1**), every other hour (a value of **2**), every third hour (a value of **3**), etc.
 - Enter the **Between** start time and end time to restrict the hours during which schedule records are created.
 - Turn on **Include Weekends** if schedule records should also be created on weekends.
- The **Weekly** pattern will create a schedule for every week in the date range defined above (subject to exceptions defined below). You can define the following for this pattern:

- Enter the **Start Time** defined on the schedule records.
- Select the day(s) of the week that the schedule records should be created.

How To Copy Job Streams From The Demonstration Database

Warning! If you are not familiar with the concepts described in the [ConfigLab](#) chapter, this section will be difficult to understand. Specifically, you need to understand how a *Compare* database process is used to copy objects between two databases. Please take the time to familiarize yourself with this concept before attempting to copy a job stream (i.e., a workflow process template) from the demonstration database.

Workflow Process Template = Job Stream. Please note that a separate workflow process template exists for each job stream. When you copy job streams from the demonstration database, you are actually copying workflow process templates.

The demonstration database contains many sample workflow process templates. The topics in this section describe how to copy these templates to your implementation's database.

Contents

[If You Work In A Non-English Language](#)

[One Time Only - Set Up A DB Process To Copy Workflow Process Templates](#)

[Run The Copy Workflow Process Templates DB Process](#)

If You Work In A Non-English Language

The demonstration database is installed in English only. If you work in a non-English language, you must execute the [NEWLANG](#) background process on the demonstration database before using it as a *Compare Source* supporting environment. If you work in a supported language, you should apply the language package to the demonstration database as well.

If you don't execute NEWLANG on the demonstration database, any objects copied from the demonstration database will not have language rows for the language in which you work and therefore you won't be able to see the information in the target environment.

One Time Only - Set Up A DB Process To Copy Workflow Process Templates

You need a "copy workflow process templates" [database process](#) (DB process) in the target database (i.e., your implementation's database) that looks like the following:

		Seq	Description	Maintenance Object	Role	Parent Seq
+	Rule(s)	10	WF Process Template	WF PROC TMPL	Primary	0
+	Instruction	20	WF Event Type	WF EVT TYPE	Child	10
+	Instruction	30	Event Activation Algorithm	ALGORITHM	Child	20

DB Process To Copy Workflow Process Templates

The demonstration database contains such a DB process; it's called **CI_COPWF**. In order to copy workflow process templates from the demonstration database, you must first copy this DB process from the demonstration database.

Warning! The remainder of this section is confusing as it describes a DB process that copies another DB process. Fortunately, you will only have to do the following once. This is because after you have a “copy workflow process templates” DB process in your target database, you can use it repeatedly to copy workflow process templates from the demonstration database.

You can copy the **CI_COPWF** DB process from the demonstration database by submitting the **CL-COPDB** background process in your target database. When you submit this process, you must supply it with an [environment reference](#) that points to the demonstration database. If you don't have an environment reference configured in your target database that references the demonstration database, you must have your technical staff execute a registration script that sets up this environment reference. Refer to [Registering ConfigLab Environments](#) for more information.

CL-COPDB is initially delivered ready to copy every DB process that is prefixed with **CI** from the source database (there are numerous sample DB processes in the demonstration database and this process copies them all). If you only want to copy the **CI_COPWF** DB process, add a [table rule](#) to the primary instruction of the **CL-COPDB** database process to only copy the **CI_COPWF** DB process. The remainder of this section assumes you have added this table rule.

When the **CL-COPDB** process runs, it highlights differences between the “copy scripts” DB process in your source database and the target database. The first time you run this process, it creates a root object in the target database to indicate the **CI_COPWF** DB process will be added to your target database. You can use the [Difference Query](#) to review these root objects and **approve** or **reject** them.

Automatic approval. When you submit **CL-COPDB**, you can indicate that all root objects should be marked as **approved** (thus saving yourself the step of manually approving them using [Difference Query](#)).

After you've approved the root object(s), submit the **CL-APPCH** batch process to change your target database. You must supply the **CL-APPCH** process with two parameters:

- The DB Process used to create the root objects (**CL-COPDB**)
- The environment reference that identifies the source database (i.e., the demonstration database)

Run The Copy Workflow Process Templates DB Process

After you have a “copy workflow process templates” DB process in the target database, you should add a [table rule](#) to its primary instruction to define which workflow process template(s) to copy from the demonstration database. For example, if you want to copy a single workflow process template called **CI_DAILY**, you would have the following table rule:

- **Table:** CI_WF_PROC_TMPL (WF Process Template)
- **Override Condition:** #CI_WF_PROC_TMPL.WF_PROC_TMPL_CD = 'CI_DAILY'

If you do not introduce this table rule to the primary instruction of the DB process, ALL workflow process templates in the demonstration database will be copied to the target database (and this may be exactly what you want to do).

Tables rules are WHERE clauses. A table rule is simply the contents of a WHERE clause except the tables names are prefixed with **#**. This means that you can have table rules that contain LIKE conditions, subqueries, etc.

At this point, you're ready to submit the background process identified on your “copy workflow process templates” DB process. This background process highlights the differences between the workflow process templates in the demonstration database and the target database (the target database is the environment in which you submit the background process).

The background process you submit is typically named the same as the DB process that contains the rules. If you used the **CL-COPDB** background process to transfer the “copy workflow process templates” DB process from the demo database, it will have also configured a batch control and linked it to the “copy workflow process templates” DB process. This batch control has the same name as its related DB process (this is just a naming convention, it's not a rule). This means that you'd submit a batch control called **CI_COPWF** in order to execute the **CI_COPWF** DB process.

When you submit the **CI_COPWF** background process, you must supply it with an [environment reference](#) that points to the source database (i.e., the demonstration database).

When the **CI_COPWF** process runs, it simply highlights differences between the workflow process templates in your source database and the target database. It creates a root object in the target database for every workflow process template that is not the same in the two environments (actually, it only concerns itself with workflow process templates that match the criteria on the table rule described above). You can use the [Difference Query](#) to review these root objects and **approve** or **reject** them.

Auto approval. When you submit **CL_COPWF**, you can indicate that all root objects should be marked as **approved** (thus saving yourself the step of manually approving them).

After you've approved the root object(s) associated with the workflow process template(s) that you want copied, submit the **CL-APPCH** batch process to cause your target database to be changed. You must supply the **CL-APPCH** process with two parameters:

- The DB process of the "copy workflow process templates" DB process (i.e., **CL_COPWF**)
- The environment reference that identifies the source database (i.e., the demonstration database)

Configuring Zones

Many zones in Oracle Utilities Customer Care and Billing do not require configuration by your implementation team. For example, the base package is shipped with the **Account Financial History** zone that appears on the **Control Central - Account Information** portal. This zone does not require configuration because its zone type has no configurable options (i.e., its behavior is static).

Other zones require configuration before they can be used because their behavior is dynamic. The topics in this section provide tips and techniques on how to configure zones in Oracle Utilities Customer Care and Billing.

Refer to [The Big Picture of Portals and Zones](#) for a description of portal and zone functionality.

Configuring Timeline Zones

A timeline zone contains one or more "lines" where each line shows when significant events have occurred. For example, you can set up a timeline zone that has two lines: one that shows when payments have been received from a customer, and another that shows when bills have been sent to the customer.

For a complete description of the numerous features available on a timeline zone, refer to [Timeline Zone - Account Info](#).

The following points describe how to set up a timeline zone:

- Set up an [algorithm](#) for each line in the zone. These algorithms will reference an algorithm type that is plugged into the **Zone - Timeline Line** plug-in spot. Click [here](#) to see the algorithm types available for this plug-in spot. Please note the following about the parameter values defined on these algorithms:
 - You can set up a timeline algorithm to show an object's "info string" when a user clicks on an event on a timeline. The object's info string appears in the zone's info area. When a user clicks on an "info string", they are transferred to a page (typically the one used to maintain the object). For example, if a user clicks on a "bill info" line, they will be transferred to the bill maintenance page.

You control the format of the info string and the destination transaction by defining the appropriate [foreign key reference](#) in each timeline algorithm's parameters. For example, if you were setting up the algorithm for a bill line, you'd reference the foreign key reference used to show bill foreign keys throughout the system.

- You can set up a timeline algorithm to show [BPA scripts](#) when a user clicks on an event on a timeline. For example, if you click on a bill event, BPA script descriptions can appear in the info area. When a user clicks on one of these descriptions, the script will execute and guide them through a respective business process (e.g., initiate a bill dispute, request a bill reprint, etc.). You define the scripts in each timeline algorithm's parameters.

When a script is initiated from a timeline, the system puts the prime key of the event into a field in the page data model. The name of the field is the column name(s) of the event's prime key. For example, when a script associated with a bill event is kicked off, the system populates a field called BILL_ID with the prime-key of the selected bill.

The script can use these page data model field to navigate to the pertinent pages. For example, if you were setting up a script to reprint a bill, the first line of the script would reference a navigation option to transfer the user to the Bill - Routing page where they can initiate the reprint. This navigation option will contain context fields that matched the names of the fields in the page data model (this is how field values are passed to pages).

- You can control every color and icon shown on a timeline by specifying the appropriate color codes on the zone's parameters.
- Set up a [zone](#) that references these algorithms. The zone will reference the **F1-TIMELINE** zone type.
- Link the zone to the appropriate portal(s) (e.g., [Control Central - Account Information](#) or [Control Central - Customer Information](#)).
- Update your users' [portal preferences](#) and [security rights](#) so they can see the zone in the desired location on the portal(s).

You can set up many timeline zones. For example,

- You might want different zones to appear on a portal depending on the type of user. For example, you might want one timeline for billing clerks, and a different one for customer service representatives.
- For aesthetic reasons, you might want multiple simple timeline zones to appear on a given portal rather than one complex timeline zone.
- You might want to set up context specific timeline zones. For example, you might want to have one timeline zone that is premise-oriented and another that is person-oriented.

ConfigLab Addendum

This chapter is an addendum to the [general ConfigLab chapter](#). This addendum describes ConfigLab functionality that is specific to Oracle Utilities Customer Care and Billing.

Account Staging

Some organizations periodically transfer account-related data (e.g., bills, payments, quotes, etc.) to a "test" environment so realistic data can be used to test new configuration data. If your organization does this, you can use this transaction to define the "stable" of accounts to be periodically transferred. Open **Admin Menu, Account Staging** to access this transaction.

Description of Page

This page is dedicated to a grid that shows the accounts to be copied to an environment by a compare DB process. These accounts are defined in respect of a given Environment Reference (meaning you can have a different set of accounts for different target environments). The following information appears in the grid.

- **Account ID** identifies a given account.
- **User** defines the user who added the account.
- **Create Date/Time** is the date and time the account was added to account staging.

Update the DB process that manages the transfer. After defining these accounts, you should update the DB process that transfers accounts to have a primary instruction that references the **ACCT STG** maintenance object. You do this in the environment to which the above accounts will be transferred (as this is the environment that you'll submit the comparison process in).

CTI-IVR Integration

Oracle Utilities Customer Care and Billing provides tools to facilitate the integration with your Computer Telephony Integration/Interactive Voice Response (CTI/IVR) system. The interface provides the following functionality:

- The ability to launch Control Central for a particular account ID or phone number from an external application
- The ability to perform an outbound phone call from within Oracle Utilities Customer Care and Billing
- The ability to accept the next call, as dictated by the CTI software, from the toolbar

This document provides technical information needed by your implementers to fully integrate with your CTI/IVR system.

Contents

[Launching The System From an External Application](#)
[Initiating an External Call](#)
[Receiving the Next Caller in the Queue](#)
[ActiveX Component - CDxCTI](#)

Launching The System From an External Application

The following sections describe possible options to launch the system from an external system.

Contents

[Launching Control Central Using an ActiveX Navigator](#)
[Launching The Application Using a URL](#)

Launching Control Central Using an ActiveX Navigator

Oracle Utilities Customer Care and Billing provides an ActiveX component ***CDxCTI.CDxNavigator*** that external applications, such as an IVR application, can use to launch Control Central for a given account number or phone number.

Enable ActiveX. In order to use the navigator, you must [configure your browser to enable ActiveX](#).

The CDxNavigator object exposes two methods:

- ***ControlCentralByAccountId*** invokes Control Central for a given account ID.
- ***ControlCentralByPhone*** invokes Control Central for a given phone number.

Contents

[Method: ControlCentralByAccount](#)
[Method: ControlCentralByPhone](#)
[Main Processing](#)

Navigation Sample Provided with the System

Method: ControlCentralByAccount

Input:

- **Server URL:** The Oracle Utilities Customer Care and Billing server URL, for example `http://spl-server`
- **AccountId:** The account ID to search for.

VB Example: Navigate to Control Central Using an Account ID

```
Dim nav As New CDxTAPI.CDxNavigator
nav.ControlCentralByAccountId ("http://spl-server:1000", AccountID)
```

Web Page Example: Navigate to Control Central Using an Account ID

```
Dim nav As New ActiveXObject("CDxTAPI.CDxNavigator");
nav.ControlCentralByAccountId ("http://spl-server:1000", AccountID)
```

Method: ControlCentralByPhone

Input:

- **Server URL:** The Oracle Utilities Customer Care and Billing server URL, for example `http://spl-server`
- **PhoneNumber:** The phone number of the person to search for
- **PhoneFormat:** The format in which the phone number is provided

VB Example: Navigate to Control Central Using a Phone Number

```
Dim nav As New CDxTAPI.CDxNavigator
nav.ControlCentralByPhone ("http://spl-server:1000", "(415) 357-5423",
"(999) 999-9999")
```

Web Page Example: Navigate to Control Central Using a Phone Number

```
Dim nav As New ActiveXObject("CDxTAPI.CDxNavigator");
nav.ControlCentralByPhone ("http://spl-server:1000", "(415) 357-5423",
"(999) 999-9999");
```

Main Processing

The ActiveX component performs the following:

- Locate the first Internet Explorer instance running Oracle Utilities Customer Care and Billing.
- If an Internet Explorer session is found, use ActiveX automation to navigate to Control Central. Depending on the search type (account or phone), enter the appropriate values in the Control Central page and launch the search.
- If an Internet Explorer session cannot be found, launch Internet Explorer and go directly to Control Central.

Navigation Sample Provided with the System

An HTML page has been provided to demonstrate the integration. This sample may be found in the following location on your Oracle Utilities Customer Care and Billing server:
`/ci/cti/CTISample.HTM`.

- Start an Internet Explorer session.
- Navigate to URL above.
- Select an account ID or phone number and click **Navigate**.

Sample Data. The accounts and phone numbers included in the sample HTML file correspond to accounts and phone numbers that exist in the demo database.

- A new session is started if one is not already active and Control Central is launched with the account corresponding to the selected account or phone number.

This sample program is provided to illustrate to implementers how to integrate their CTI-IVR system with Oracle Utilities Customer Care and Billing.

Launching The Application Using a URL

You may also launch the application using a URL. With this option you can set the system to launch a script upon startup. You can also indicate to the system to automatically load an appropriate page (if this information is not part of the script).

Refer to [Launching A Script When Starting The System](#) for further information.

Initiating an External Call

This section describes the automated dialer functionality provided with the system as well as information about integrating with your own automated dialer.

Contents

- [Overview of Automated Dialer](#)
- [Technical Implementation of Automated Dialer](#)
- [Customize Integration to Your Automated Dialer Software](#)
- [Customize Automated Dialer User Interface](#)

Overview of Automated Dialer

In order to initiate a call to a customer from within the system, a context menu item **Go To Automated Dialer** is available on the Person context menu. To call a customer displayed in the current context, choose this option from the person context menu and a window appears, showing a list of phone numbers defined for that person.

Select the desired phone number and click **Dial**.

Context Entry Secured. The [navigation key](#) for this window **automatedDialer** refers to an application service to facilitate application security. If your installation does not support an integration with external dialer software, configure the security settings to ensure that users do not have access to the application service for this context entry.

Technical Implementation of Automated Dialer

The popup window is implemented as a JSP page, which calls the JSP page `ext_cti_dialer.jsp` to integrate with an automated dialer. The `ext_cti_dialer.jsp` page provided with the system integrates to the Microsoft Phone Dialer, available on any Windows 2000 workstation.

The Microsoft Phone Dialer is invoked through the `CDxPhoneDialer` ActiveX object.

```
*****
* Invoke an external phone Dialer
* In the sample provided we launch the Microsoft phone Dialer
*****
*/
function callDialer(phoneNumber){
    CDxPhoneDialer.makeCall(phoneNumber);
}
```

Object: `CDxPhoneDialer`

This object is used to call the Microsoft Phone Dialer for an outbound call. It is only used when your implementation uses the Microsoft standard dialer.

Method: `makeCall`

Input: Phone Number

Microsoft Phone Dialer Configuration

If your implementation chooses to use the functionality provided with the system and integrate with Microsoft Phone Dialer, you must perform the following steps:

- Copy the JSP page `ext_cti_dialer.jsp` from the `/cm_templates` directory found under the web application root directory on your Oracle Utilities Customer Care and Billing server to the `/cm` directory.

Enable ActiveX. In order to use the integration with the Microsoft Phone Dialer, you must [configure your browser to enable ActiveX](#).

Customize Integration to Your Automated Dialer Software

In order to integrate with a different automated dialer software application, your implementers must modify the `ext_cti_dialer.jsp` to call the appropriate dialer.

- Copy the JSP page `ext_cti_dialer.jsp` from the `/cm_templates` directory found under the web application root directory on your Oracle Utilities Customer Care and Billing server to the `/cm` directory.
- Make the appropriate changes to the copy of `ext_cti_dialer.jsp` in the `/cm` directory to integrate with your automated dialer.

Customize Automated Dialer User Interface

Your implementation may choose to display a different user interface for the **Go To Automated Dialer** function than the one provided with the system. For example, perhaps there is more information that you would like to display in addition to the person's name and phone numbers. In order to do this, perform the following steps:

- Create your customized component to provide the desired functionality.
- Create a navigation key for your new component and indicate the URL being overridden. The remainder of the section walks you through these steps.

Go to **Utilities, System, Navigation Key +**.

For **Navigation Key**, specify a name for the new navigation key prefixed with CM.

For **URL Location**, select **External (Override)** to override a base navigation key.

When you select **External (Override)**, the **Overridden Navigation Key** becomes available. Select the **automatedDialer** navigation key because that is the key you are overriding.

The **URL Override** is the path on the Web server to your custom component.

When overriding a navigation key, you must flush the system login cache on the Web server. The navigation keys are stored in the system login cache, so the overrides do not become effective until the cache is flushed. To flush the cache, issue the following command in your browser's address bar: **<http://server:port/flushSystemLoginInfo.jsp>**, where server is the name or address of your web server and port is the port number of the application, for example, **<http://CD-Implementation:7500/flushSystemLoginInfo.jsp>**.

Refer to the [Defining Navigation Keys](#) for more information.

Receiving the Next Caller in the Queue

If your CTI-IVR system allows users to request the next caller waiting in a queue, the system provides a mechanism to integrate with this functionality.

A **Next Call** button is available in the toolbar that can be used to request the next call waiting in an inbound queue managed by a CTI application. This button is only enabled if you have set the **CTI Integration** flag to **Yes** on your [installation options](#).

When the next call button is clicked, it launches a browser script function called **launchCTI** located in a file called **ext_cti.jsp**. The **launchCTI** function calls a function called **ctiGetNextCaller** to retrieve the next caller's account ID and uses the **CDxNavigator** object to launch Control Central for that account.

Customize Integration to Your Next Caller Function

The **ext_cti.jsp** file shipped with the base product provides sample functionality that should be replaced with the appropriate integration to your CTI application. In the sample provided, the **ctiGetNextCaller** randomly takes an account ID from a predefined list of accounts.

In order to integrate the next caller functionality with your CTI-IVR system, perform the following steps:

- Copy the JSP page `ext_cti.jsp` from the `/cm_templates` directory found under the web application root directory on your Oracle Utilities Customer Care and Billing server to the `/cm` directory.
- In the `/cm` directory, replace the contents of the `ctiGetNextCaller` function to retrieve the next caller ID from your CTI application.

ActiveX Component - CDxCTI

The system provides an ActiveX component **CDxCTI** that contains all the functionality required for inbound and outbound calling. It contains two objects:

- CDxNavigator
- CdxPhoneDialer

Contents

[Configuring Your Browser to Enable ActiveX](#)

[Installing the CDxCTI ActiveX Component](#)

[Creating an Instance of the CDxCTI Object in a Web Page](#)

Configuring Your Browser to Enable ActiveX

In order to use the automated phone dialer functionality or the next caller functionality, your users must configure their browser to enable ActiveX. Perform the following steps:

- In your Internet Explorer browser window, navigate to **Tools, Internet Options** and go to the **Security** tab. From there, select **Local Intranet**.
- Click **Custom Level**.
- Under the ActiveX controls and plug-ins section, set the following:
 - Download signed ActiveX controls: **Prompt**
 - Download unsigned ActiveX controls: **Disable**
 - Initialize and script ActiveX controls not marked as safe: **Disable**
 - Run ActiveX controls and plug-ins: **Enable**

Installing the CDxCTI ActiveX Component

The CDxCTI ActiveX components install automatically the first time the Automated Phone Dialer is launched or when the `CTISample.htm` is launched. The CDxCTI component is signed using Microsoft Authenticode technology, therefore when it is downloaded the first time, a dialog will appear describing the source of the component and asking the user to accept the installation of the component on the local machine.

Creating an Instance of the CDxCTI Object in a Web Page

To use the CDxCTI objects from a web page, declare them explicitly using the `OBJECT` tag:

```
<OBJECT ID="CDxPhoneDialer"  
CLASSID="CLSID:151A6E91-8C55-4666-BFFB-9EC345583CBD"  
CODEBASE="CDxCTI.CAB#version=1,5,0,8">  
</OBJECT>  
  
<OBJECT ID="CDxNavigator"  
CLASSID="CLSID:E7EF882D-662A-4451-A78C-CD62393F06C6"  
CODEBASE="CDxCTI.CAB#version=1,5,0,8">  
</OBJECT>
```

Alternatively, use the new ActiveXObject function

```
var nav = new ActiveXObject("CDxCTI.CDxNavigator");  
  
var nav = new ActiveXObject("CDxCTI.PhoneDialer");
```

Web Self Service

The Web Self Service (WSS) application is a Web application that is used by a customer of an Oracle Utilities Customer Care and Billing installation to query and update customer information. WSS provides installations with a framework and templates for implementing transactions typical to a WSS application. In addition some sample web pages are provided.

This section describes the sample pages provided and includes guidelines for an implementer to follow when adding additional WSS pages to the pages provided.

Contents

- [WSS Pages Provided with the System](#)
- [Customizing Web Self-Service](#)

WSS Pages Provided with the System

The system provides a limited number of pages for the web self-service application. These pages are provided as an example of how you may implement your WSS application. All the pages provided by the system may be used by your implementers as a sample to use for incorporating such a page into your web application.

Refer to [Customizing Web Self-Service](#) for more information.

Contents

- [Login Related Pages](#)
- [Account and Person Related Pages](#)

Login Related Pages

The following pages are related to registering and logging in to the WSS application.

Contents

- [Login Page](#)
- [Registration Page\(s\)](#)
- [Password Reminder Page](#)

Login Page

The login page allows a registered user to login to the WSS application with a user name and password.

The page also provides links to enable the user to register as a new user and to request a reminder for a forgotten password.

Customer Contact Added. A customer contact is added when a customer successfully logs in or if a customer's attempt to login failed if you have set up the appropriate customer contact classes and types in the configuration properties file. Contact your system administrator for more information.

Registration Page(s)

The registration pages allow a customer to sign up for web self-service.

- The initial Customer Registration page asks the customer to provide the account number and name in order for the system to identify the customer's person record.
- Once the customer's person record has been identified, the customer is asked to provide their email address and to define a user id and password. The customer must also choose a password reminder question and indicate an answer. This information is stored on the [person](#) record. If the base-package algorithm ([C1-WEBID](#)) is specified on the person ID Type, it will ensure that the user id entered is unique.

Upon successful registration, an email confirmation is sent to the customer.

Enabling Email Correspondence. Sending an email is only possible if the WSS application has been configured with the appropriate settings for communicating with an SMTP server. Contact your system administrator to verify that these settings have been defined.

Preventing Web Access. A customer may only view information about related accounts if the [account / person](#) record has been marked to allow web access.

Password Reminder Page

This page is displayed if the customer has clicked the "forgot password" hyperlink from the login page. The password reminder question is displayed and the customer is asked to provide the answer. If the correct answer is provided, an email is sent to the customer's email address with the user name and password.

Enabling Email Correspondence. Sending an email is only possible if the WSS application has been configured with the appropriate settings for communicating with an SMTP server. Contact your system administrator to verify that these settings have been defined.

Account and Person Related Pages

The pages described in this section are available once the customer has successfully logged in.

Contents

- [Account Summary](#)
- [Account Information](#)
- [Enter a Meter Read](#)
- [View Account Financial History](#)
- [View Billing History](#)
- [Make a Payment](#)
- [Stop Service](#)
- [Update Personal Info](#)
- [Change Password](#)

Account Summary

The My Accounts page displays after logging in if the customer's person record is linked to more than one account. For each account, information about the main customer is displayed, along with the account's balance.

Select one of the accounts and you are brought to the Account Information page.

Account Information

The Account Information page displays high-level information about the person and account. This page also displays a list of service agreements linked to the account. For services that are linked to metered service points, the customer may choose to enter a meter read by clicking the Add Meter Read hyperlink.

Enter a Meter Read

This page is accessible from the account information page through the Add Meter Read hyperlink that is available in the service agreements collection for service agreements linked to metered service points.

If the service agreement is linked to more than one service point, then you must choose the desired service point. For each register linked to the service point's meter, you may enter a register reading.

Customer Contact Added. A customer contact is added when a customer enters a meter read if you have set up the appropriate customer contact class and type in the configuration properties file. Contact your system administrator for more information.

To Do Entry Added for Hi/Lo Failure. A To Do entry is added if the entered meter reading fails hi/lo if you have set up the appropriate To Do type in the configuration properties file. Contact your system administrator for more information.

View Account Financial History

The Account Financial History page displays the financial history for the account. The page is similar to the account financial history page in Oracle Utilities Customer Care and Billing, with some minor differences. For example, only the current amount and current balance information is displayed. The payoff amount and payoff balance are not displayed. In addition, the customer does not see any adjustment whose adjustment type's print by default flag is not checked.

View Billing History

The Billing History page displays the information about the customer's bills including the date, amount and the running balance. In addition, if the WSS application has been configured to integrate with DOC1 to display bills, a "view" hyperlink appears for each bill to allow the customer to see an image of each bill.

Integrate with DOC1. Contact your system administrator for more information about whether the WSS application has been configured to integrate with DOC1.

Make a Payment

The Make Payment page allows the customer to enter a credit card payment to pay all or part of the outstanding balance for the account.

Configuration Required. When setting up the WSS application, an appropriate tender type and appropriate auto pay source codes should have been defined in the system and referenced in the configuration properties file. Contact your system administrator for more information.

Customer Contact Added. A customer contact is added when a customer enters a credit card payment if you have set up the appropriate customer contact class and type in the configuration properties file. Contact your system administrator for more information.

Stop Service

The Stop Service page allows the customer to request that service for one or more of their active service points stops on a date entered by the customer.

Customer Contact Added. A customer contact is added when a customer requests service to be stopped if you have set up the appropriate customer contact class and type in the configuration properties file. Contact your system administrator for more information.

To Do Entry Added. A To Do entry is added when a customer requests service to be stopped if you have set up the appropriate To Do type in the configuration properties file. Contact your system administrator for more information.

Update Personal Info

The Update Personal Info page allows a customer to change personal information such as phone numbers, email address and mailing address.

Customer Contact Added. A customer contact is added when a customer has changed personal information if you have set up the appropriate customer contact class and type in the configuration properties file. Contact your system administrator for more information.

To Do Entry Added. A To Do entry is added when a customer has changed personal information if you have set up the appropriate To Do type in the configuration properties file. Contact your system administrator for more information.

Change Password

This page is allows the customer to change their password. After the password is changed, an email is sent to the customer's email address with the user name and password.

Enabling Email Correspondence. Sending an email is only possible if the WSS application has been configured with the appropriate settings for communicating with an SMTP server. Contact your system administrator to verify that these settings have been defined.

Customizing Web Self-Service

WSS provides Oracle Utilities Customer Care and Billing customers with a framework for implementing transactions. This framework enables the expansion of the functions available to your customer to include other business functions supported by the system. The following sections identify the different possibilities for customizing and expanding the WSS application.

Upgrading after Customization. If you would like to modify any of the web application functionality provided by the system, you must make copies of the files provided and only make your changes in the copied files. If you have followed this procedure, every effort will be made to provide a seamless upgrade of the web application for bug fixes and future releases.

Contents

- [Technical Overview](#)
- [Modifying Page Layout](#)
- [Adding a New Transaction](#)
- [Modifying the Menu](#)
- [Style Sheet Modifications](#)
- [WSS Standards](#)

Technical Overview

The Self Service application is a regular J2EE Web application, conforming to the Servlet 2.2 specification. It uses JSP for the front end and a Java servlet in the back end. All Oracle Utilities Customer Care and Billing access is done through XAI and there is no direct connection to the database. The technical design is based on model view controller architecture.

The presentation layer is separated from the data access and business rule layer. The JSP pages contain plain HTML (there are no applets, and very little JavaScript - close to none) with the visual layout of the screens. Dynamic elements, such as customer name, address, balance, etc. are represented as Java Beans (but not EJBs) and are embedded in the page using JSP directives. It is very easy for a Web designer to modify or replace the graphic design of the page without programming knowledge. All the logic of the application, including communication with XAI, sits in the Java Beans (Java classes representing data) and in the servlet.

Each of the Java Beans represents, roughly, an object in Oracle Utilities Customer Care and Billing, or the set of information returned by an XAI service. Each bean knows how to retrieve its data through XAI if needed, and how to make updates (e.g. enter a payment).

The servlet is the controller that holds the pages together and implements security. Every user request gets intercepted by the servlet, which first checks that the user has been authenticated, performs any background actions if necessary, creates Java Beans with all the information to be displayed to the user, and passes the beans to the appropriate JSP page. Each action in Self Service, for example, show account information, show financial history, make payment, is a separate class.

Modifying Page Layout

If you would like to keep the functionality provided by a given web application page, but you would like to change the look and feel of the page, perform the following steps:

- Find the JSP file for the page that you want to modify. The JSP files are found in the SelfService directory.
- Copy the file and name your new file the same name as the file you are copying, but with “CM” in the beginning of the file name.
- Make the desired changes in your new JSP file.

Assuming that you want all the other functionality to remain the same, you are finished. You do not need to modify the menu or any action classes to call your new CM JSP file. The functionality provided always looks for the existence of a “CM” version of the JSP file and uses that file instead if one is found. If a “CM” version is not found, the JSP provided by the system is used.

Adding a New Transaction

As explained in the [technical overview](#), there are two parts for each transaction.

- On the client side, the JSP page contains the UI information
- On the server side, a servlet administers the processing of the transaction using one or more java beans containing the logic to communicate with Oracle Utilities Customer Care and Billing.

If you want to add another action, for example enroll in AutoPay, you must do the following:

- Write another class for the servlet to recognize the action
- Write another bean that would know how to capture AutoPay information
- Add one or more JSP pages for the new screens that are used to interact with the customer.

Naming Convention. In order to ensure that new files added in future releases do not override your files, be sure to use “CM” for “customer modification” for the first two letters of every new file.

Contents

- Identifying a Transaction
- Creating an XAI Schema
- Creating a Java Bean
- Creating an Action Class
- Control Classes
- Creating a JSP Page

Identifying a Transaction

In principal, every transaction that exists in the system could get implemented in the WSS, since all Oracle Utilities Customer Care and Billing objects are described as internal services, which can be accessed via XAI.

The question often will be whether the implementer will want to maintain the degree of diversity and sophistication that exists in many transactions, or if the maintenance of the main object attributes might already fulfill the requirements.

For reasons of simplicity, the following sections demonstrate the implementation of the service address maintenance, which is handled in the system on the premise page. A similar function already exists in the WSS for updating the mailing address in the [Update Personal Info](#) transaction.

Creating an XAI Schema

This section assumes that you are familiar with the topics described in [XML Application Integration](#).

The main steps for our purpose are to go in the [XAI Schema Editor](#) to the “Schemas” menu item, and to create an XAI inbound service. Use the search function to assist you in finding the correct service. Our premise maintenance is based on the meta info file CILCPRMP.

The schema needs to be saved and registered. You should also [test the new service](#).

Naming Convention. In order to ensure that new files added in future releases do not override your files, be sure to use “CM” for “customer modification” for the first two letters of your new service. In addition, set the owner flag to **Customer Modification**.

Creating a Java Bean

The java bean is responsible for retrieving information from the respective Oracle Utilities Customer Care and Billing object. Within the classes directory of the WSS application you will find two kinds of java classes: One group ending their class name with “Action” (these are discussed in the next section) and the other group ending with “Bean” (so called object beans).

You can take almost any java class ending with “Bean” as an example and template for developing a new bean.

Note. Not all classes with names ending with “Bean” are object classes with the characteristics mentioned below. Some of those beans serve only as container classes describing the data members of an Oracle Utilities Customer Care and Billing object, without having any additional logic to retrieve data via XAI. Usually such container classes are used when there are more than two data members in the object and will be referenced by an object class retrieving more than one record. For example, in the Account Info page we are displaying the SAs for an account. The object bean retrieving those SAs is the “SAforAccountBean” and it stores the SAs in a data array, whose members are instances of the container class “SAforAccount”.

The following is a description of the common parts of all object beans:

- All beans belong to the package `com.splwg.selfservice`.
- All beans communicating with XAI will need to import the package `org.dom4j.*`.
- All beans implement the java class `java.io.Serializable`.
- Variables are always defined in private scope, and a public function exists to populate and/or to retrieve their values.
- The property object is used to hold all properties that exist in the `SelfServiceConfig.properties` file (e.g. the XAI server address). This object needs to be transferred from the action bean to the object bean. You may either achieve this by providing the property object as an argument to the constructor, or by explicitly setting it in a separate method.

- One main function contains the XAI request to be sent to the XAI server and the code to parse the response.

For this purpose it might be helpful to use the request and response xml, so that the elements and attributes can be copied into your code. You can see this xml by using the XAI Submission page or the XAI Dynamic Submission page for the appropriate XAI inbound service. In addition, if XAI logging is switched on, you may view the xai.log file to see all requests and responses received or sent by the XAI server. Another option is to use the XAI Schema Editor testing tool, in order to submit such a request to XAI.

All attributes within elements you may want to retrieve from the service need to get parsed and stored into variables.

Any error message returned (or created by the bean) should be stored into a variable (usually named "errorMsg"), and its value will be inquired by the calling method. Refer to [Error Message Handling](#) for more information.

For our test case example of creating a service for service address maintenance, you will likely want to create two main methods.

- One method to retrieve the data according to the primary id to be provided
- The other one to enable the update of the service address according to all attributes provided as input parameters.

Naming Convention. In order to ensure that new files added in future releases do not override your files, be sure to use "CM" for "customer modification" for the first two letters of every new file.

Creating an Action Class

One kind of java class has an "Action" suffix and controls the process flow, which might be more or less complicated according to the required functionality.

This action class is always called from the controller class. Refer to [Control Classes](#) for more information.

The common characteristics of this type of class are:

- All action classes implement the interface SSvcAction.

This interface defines one common method by the name of "perform". The HttpServlet, HttpServletRequest, and HttpServletResponse instances are required arguments, ensuring accessibility to the servlet properties and servlet session to every class.

- A transaction might have different steps. For example, the first step for the update personal info transaction is to show the current personal info data. The second step processes the updated data. According to the different steps, different methods are called. For an example of a multi-step transaction, refer to PayOnlineAction.java.
- An instance of the object bean is created and after setting the input parameters, the main function is called.
- According to the result of the previous step, a JSP page is called (by calling the forward method of the servlet request dispatcher).

For your new transaction, you must create an action class to execute the appropriate steps. You must also go to the Actions section of the `CMSelfServiceConfig.properties` file to indicate the name of the class along with its action (reference) name. This is the name used to reference the class in your object bean.

The `CMSelfServiceConfig.properties` file was configured at installation time. Refer to the installation documentation for more information. This file can be found in the `WEB-INF` directory of the `SelfService` directory in your server. Examples of how to define your action name may be found in this file. The property name used to define the property must start with `com.splwg.selfservice.Action`. The `SSvcController` class will dynamically load these actions in the `initActions` method.

Control Classes

There are some classes that exist to perform a special task.

- The “`SSvcController`” class is the servlet called by all JSP forms on submission. The first time it is called is when starting the `SelfService` application. (The web application directory “`SelfService`” contains the file “`index.html`”, which refers the user to this servlet with a login action.)

According to the action called, the respective action class is invoked.

This class also keeps track of the servlet session instance, and if it has timed out, it forces the user to log on again.

- The “`XAIHTTPCall`” class administers the call to XAI. All object beans communicating with XAI call it. You won’t need to change anything in this class.
- The “`Util`” class contains static methods to be called from various classes in the application. Currently it contains only one method that replaces null values with blanks.
- The “`XMLEncoder`” class may be called to make sure that any input string complies with xml or http standards. For example, special characters like an ampersand are replaced.

Our service address maintenance action class will have two steps. The first step is called when selecting the transaction from the menu, and reads the respective premise record by invoking the created object bean. The second step is called upon submission of the form, and processes the changes submitted.

Creating a JSP Page

Every JSP page consists of three parts: the HTML part, the server script and the client script. All HTML elements are legitimate in a JSP page, and both the client script as well as the server script may dynamically control the display elements of the page.

This capability is used several times in the application. One example is the dynamic creation of lists, like the SA list in the Account Information screen, or the address list in the Update Personal Info screen.

A server script using java may use all packages and classes available to it. Object beans are transferred from the action bean as attributes of either the session or the request object. Depending on this differentiation, the JSP routine `jsp:useBean` will have either the scope request or session.

Other JSP or html files may get included with the include command.

The client script contains mostly validation methods called before the form is submitted.

By sticking to the style sheet with its already defined styles and avoiding style definitions in HTML elements, you will keep graphically in line with the existing pages.

For the service address maintenance you will probably create two pages. One page displays all the data retrieved, where either all or some will be modifiable by the customer. The other page informs the customer that the data was updated successfully.

Naming Convention. In order to ensure that new files added in future releases do not override your files, be sure to use “CM” for “customer modification” for the first two letters of every new file.

Modifying the Menu

A file called Menu.jsp contains a DisplayMenu function that determines what menu items to display. This function first checks whether or not a CMDisplayMenu function exists. If one does, that method is called to display the menu items. If no CM menu file exists, then the CIDisplayMenu function is used.

To customize the items listed in the menu, perform the following steps:

- Edit CMMenu.jsp (found in the SelfService directory) and modify the CMDisplayMenu method to add/remove menu contents as desired. For example, you may choose to hide a menu item that you do not want to provide for your customers and/or you may add new menu items that refer to new pages you have created.

For each new JSP file that you have created, you must do the following:

- Include the files Menu.jsp, CIMenu.jsp and CMMenu.jsp into your JSP file.
- Call the DisplayMenu method from your JSP file. Refer to any of the JSP files provided by the system as an example.

Style Sheet Modifications

JSP UI elements have the style defined in their class attribute. The declaration of all styles is found in the wss.css file within the SelfService directory. The JSP pages reference the style sheet by including a file called IncPreamble.jsp. When this JSP is loaded, it checks whether or not a CMwss.css file exists. If one does, that style sheet is used for the page. If no CM style sheet file exists, the page uses the original wss.css.

To customize the style sheet provided, perform the following steps:

- Make a copy of the wss.css file and call it CMwss.css.
- Make changes to CMwss.css as desired.

WSS Standards

Contents

[Error Message Handling](#)
[Session Object Information](#)

Error Message Handling

The main methods in the object beans (the ones communicating with XAI) have boolean return values. The action bean calls those methods.

If the call to such a method returned true, the program flow continues. A return code of false informs the action bean that an error has occurred.

As a default, the error message that may be received from the Oracle Utilities Customer Care and Billing service or that may be created by the method is stored in a variable by the name of "errorMsg", and the action bean retrieves this variable.

Usually the action bean then redirects the program flow back to the calling JSP page.

Each JSP page includes a JSP file ("ErrorHandler.jsp") at the beginning of its form, which checks whether the transferred error message variable is unlike null, and if so displays a message box with the error message.

An error message returned from the server is displayed in the appropriate language defined for the self-service application. The Oracle Utilities Customer Care and Billing message should be displayed or trapped and mapped to a more user friendly message. Refer to NewPaymentBean.java for an example of overriding a message received by Oracle Utilities Customer Care and Billing.

All messages that are generated by a WSS object or an action bean are also stored in the system message table to take advantage of the multi-language storage of messages. The application displays the message in the appropriate language for the self-service application.

New Error Messages. If your new page requires new messages, define the messages in the system using message category 90000 or greater, which are reserved for implementations.

Session Object Information

Data you want to transfer from an action bean to a JSP page may be put either into the http request object or into the http session object. The difference is in the scope.

If the information will only be valid for the specific request, then store the information in the request object. The session object stays active until the session is ended or timed out, and therefore any information you want to keep accessible in your application can be stored in the session object.

In our WSS application, the ValidUserBean is stored in the session object. On logon its information (account id, user name, entity name, currency symbol) is loaded and stays accessible by all WSS JSP pages.

The Conversion Tool

This document describes the Oracle Utilities Customer Care and Billing conversion tool.

Contents

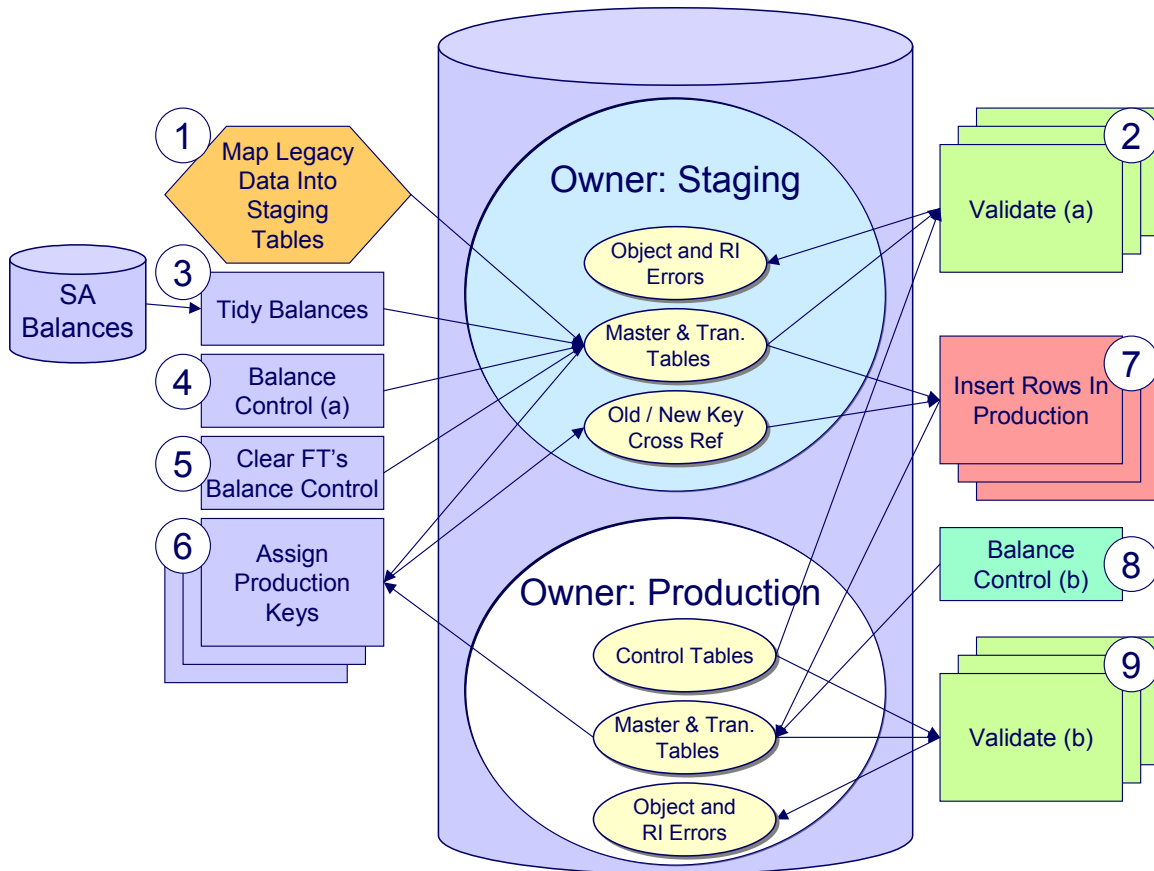
- [Introduction](#)
- [Conversion Tool Steps](#)
- [The Validation User Interface](#)
- [The Staging Tables](#)
- [Appendix A - Entity Relationship Diagramming Standards](#)
- [Appendix B - Multiple Owners In A Single Database](#)
- [Appendix C - Known Oddities](#)

Introduction

When you're ready to convert data from your legacy system into Oracle Utilities Customer Care and Billing, you will have analyzed your CIS processing requirements according to your business and organizational needs and set up the control tables accordingly.

Refer to the ***Administration Guide*** for a complete discussion of the various control tables and the order in which they must be set up.

After the control tables are set up, you are ready to load data into the system from your legacy system. This conversion effort involves several steps as illustrated in the following diagram:



The following points briefly outline each of the above tasks:

- Map Legacy Data Into Staging.** During this step, your legacy master data (e.g., account, person, premise, meter) and transaction data (e.g., bills, payments, meter reads) is migrated into the system. Notice that you are not migrating this data directly into production. Rather, your rows are loaded into tables that are identical to the production tables; they just have a different owner. Refer to [Appendix B – Multiple Owners In A Single Database](#) for information about table ownership.

Different Databases For Staging and Production. The above diagram illustrates how the system is configured to support the conversion effort in the standard installation, i.e., the staging tables are in the same database as the production tables (each with a different owner). However, it is possible for the staging tables to be in a separate database. This option requires additional effort on your part (because you would have to copy the control tables from production into your staging database). Please refer to [Appendix B – Multiple Owners In A Single Database](#) for information about this alternative.

Mapping legacy data into the system is probably the most challenging part of the conversion process because the system is a normalized database (and most legacy applications are not).

- Validate (a).** During the validation (a) step, the system validates the data you loaded into the staging tables. Two types of validation programs exist:

- **Object Validation Programs.** Each of the system's master data objects (e.g., person, account, premise, meter, etc.) is validated using the same logic that is used to validate data added by users in your production system.
- **Referential Integrity Validation Programs.** After you have successfully validated the master data objects, the referential integrity validation programs are executed to validate transaction data and to highlight "orphaned" rows. These programs check the validity of the foreign keys on all rows on all tables.

Control tables from production. It's important to notice that the validation programs validate your staging data using the control tables that have been set up in production. Refer to [Appendix B – Multiple Owners In A Single Database](#) for a description of how this works.

- **Tidy Balances.** During this step, the system creates adjustments that cause each SA's current and payoff balances to equal the desired balances. The desired balances are supplied on a flat file prepared by you.
- **Balance Control (a).** During this step, you run the balance control program and then verify that the balances that it generates are consistent with the balances in your legacy system.
- **Clear FT's Balance Control.** In the previous step, the system creates a balance control and links it to the FT's. If the balance control's balances are consistent with the amount of receivables being transferred into the system, you should run the Clear FT's Balance Control program. This program simply resets the Balance Control column on the FT so that the FT's can be included in a balance control (see the last step below) after they have been transferred to production.
- **Assign Production Keys.** During this step, the system allocates random, clustered keys to the rows in the staging database.
- **Insert Rows Into Production.** During this step, the system populates your production tables with rows from the staging. When the rows are inserted, their prime keys are reassigned using the data populated in the previous step.
- **Balance Control (b).** During this step, you run the balance control program against production. You do this to verify the balances in production are consistent with the values of receivables converted from your legacy application.
- **Validate (b).** During this step, you rerun the object validation programs, but this time against production. We recommend rerunning these programs to confirm that the insertion programs have executed successfully. We recommend running these programs in random sample mode (e.g., validate every 1000th object) rather than conducting a full validation in order to save time. However, if you have time, you should run these programs in full validation mode (to validate every object).

Conversion Tool Steps

The following sections provide more details about the steps in the conversion process.

Contents

[Map Legacy Data Into Staging Tables](#)
[Validate Information In The Staging Tables](#)
[Tidy Balances](#)

Balance Control (a)
Clear FT Balance Control
Allocate Production Keys
Insert Production Data
Run Balance Control Against Production
Validate Production

Map Legacy Data Into Staging Tables

This section provides some high level discussion about mapping legacy data to the system's staging tables. Refer to [The Staging Tables](#) for details about the staging tables in the system.

Recommendation. You can use any method you prefer to load Oracle Utilities Customer Care and Billing data from your legacy application. However, we recommend that you investigate your database's mass load utility (as opposed to using insert statements) as the mechanism to load the staging tables. In addition, we strongly recommend that you disable the indexes on these tables before populating these tables and then enable the indexes after populating these tables.

A Note About Keys

The prime keys of the tables in the staging database are either system-assigned random numbers or they aren't. Those tables that don't have system-assigned random numbers have keys that are a concatenation of the parent's prime-key plus one or more additional fields.

Every table whose prime key is a system-assigned random number has a related table that manages its keys; we refer to these secondary tables as "key tables". The following points provide more information about the key tables:

- Key tables are used by programs that allocate new keys. For example, before a new account ID is allocated, the key assignment program checks the account key table to see if it exists.
- Key tables exist to support archiving and ConfigLab requirements.
 - When an object is [archived](#), its row is removed from the primary table, but its key remains on the key table. This is done so that the key isn't reused in production, as we want to be able to reinstate an archived object.
 - From a [ConfigLab](#) perspective, these key tables are used to prevent a key from being reused in production for an object added in a ConfigLab. For example, a user might add an interval profile in a ConfigLab and we don't want its key to be allocated to an interval profile added in production.
- Key tables only have two columns:
 - The key of the object.
 - An environment ID. The environment ID identifies the database in which the object resides.
- Key tables are named the same as their primary table with a suffix of "_K". For example:
 - The key table for CI_ACCT is CI_ACCT_K
 - The key table for CI_PREM is CI_PREM_K

The name of every table's key table is defined under the Generated Keys column in the Table Names sections in [The Staging Tables](#).

- When you populate rows in tables with system-assigned keys, you must also populate a row in the related key table. For example, if you insert a row into CI_ACCT, you must also insert a row into CI_ACCT_K. The environment ID of these rows must be the same as the environment ID on this database's [installation record](#).
- When you populate rows in tables that reference this record as a foreign key, you must use the appropriate key to ensure the proper data relationships. For example, if you insert a row in CI_SA for the above account, the ACCT_ID column must contain the temporary account key.
- When you insert rows into your staging database, the keys do not have to be random, system-assigned numbers. They just have to be unique. A later process, [Allocate Production Keys](#), will allocate random, system-assigned keys prior to production being populated.

Validate Information In The Staging Tables

During the first validation step, the system validates the data you loaded into the staging tables. Two types of validation programs exist:

- **Object Validation Programs.** The object validation programs validate each of the system's master data objects (e.g., person, account, premise, meter, etc.) and a limited number of transaction data objects (e.g., field activity, field order, etc.). Please note that these programs call the same programs that are used to validate data added by users in your production system.
- **Referential Integrity Validation Programs.** After the master data objects have been validated, the referential integrity validation programs are executed to validate transaction data and to highlight "orphaned" rows. These programs simply check the validity of the foreign keys on all rows on all tables.

The contents of this section describe how to execute the validation programs.

Contents

- [Object Validation Programs](#)
- [Submitting Object Validation Programs](#)
- [Referential Integrity Validation Programs](#)
- [Submitting Referential Integrity Validation Programs](#)
- [Recommendations To Speed Up Validation Programs](#)

Object Validation Programs

Each of the objects described under [Master Data](#) must be validated using the respective object validation program indicated in its Table Names section.

In a limited number of cases object validation is available for [Transaction Data](#) objects, where customers may convert transaction data that is still pending. For example, if you are converting **pending** field activities, you want to ensure that the data is valid. For these cases you may also be converting historic records. For example, in addition to the pending field activities you are converting completed field activities to keep a historic view. You may not want to perform validation on completed records. As a result the background processes provided for transaction data allow you to limit the validation to records in a give status.

We strongly recommend validating each object in the following steps:

- Execute each object's validation program in random-sample mode to highlight pervasive errors. When you execute a validation in random-sample mode, you are actually telling it to validate every X records (where X is a parameter that you supply to the job). Refer to [Submitting Object Validation Programs](#) for more information about the parameters supplied to these background processes.
- You can view errors highlighted by validation programs using the [Validation Error Summary](#) transaction.
- Correct the errors using SQL. Note, you can use the base package's transactions (e.g., Person Maintenance, Premise Maintenance, etc.) to correct an error if the error isn't so egregious that it prevents the object from being displayed on the browser.
- After all pervasive errors have been corrected; re-execute each object's validation program in all-instances mode to highlight elusive, one-off errors. Refer to [Submitting Object Validation Programs](#) for more information about the parameters supplied to these background processes.

Take note! Whenever an object validation program is run, it is necessary to delete all previously recorded errors associated with its tables from the validation error table before it inserts new errors.

After the various object validation programs run cleanly, run the referential integrity validation programs as described in the next section.

Submitting Object Validation Programs

The object validation programs that are described in the [staging tables](#) table names matrices are classic background processes as they can also be run against production data. You submit these processes in the same way you submit any background process in production. Refer to [Object Validation Processes](#) for information about these processes and their parameters.

Referential Integrity Validation Programs

It's important to understand that only master data objects (e.g., persons, accounts, meters, premises, etc.) are validated by the object validation programs discussed above. This means that only master data objects will have their foreign keys checked for validity by the object validation programs. You must run the referential integrity programs to validate all other data.

The referential integrity validation programs highlight:

- Orphaned rows because orphan rows, by definition, don't reference an object.
- Invalid foreign keys on transaction data.

Validating Transaction Data. You may wonder why transaction data is not subject to the object validation routines. This is because: a) the production system only needs validation logic for master data because transaction data is not entered by users, and b) most conversions necessitate loading skeletal transaction data because the legacy system typically doesn't contain enough information to create accurate transactions in the system.

Each of the tables described under [Transaction Data](#) must be validated using the respective referential integrity validation program indicated in its Table Names section. We strongly recommend validating each table in the following steps:

- Execute each table's referential integrity validation program. Refer to [Submitting Referential Integrity Validation Programs](#) for more information about the parameters supplied to these background processes.
- You can view errors highlighted by this process using the [FK Validation Summary](#) transaction.
- Correct the errors using SQL (you cannot use the application to correct these types of errors).
- Rerun the referential integrity programs until no errors are produced.

Take note! Whenever you run a referential integrity validation program, it deletes all errors associated with its table from the referential integrity error table.

In order to highlight orphaned rows in the master data, run the referential integrity validation programs against all tables described under [Master Data](#) using the procedure described above.

When ALL referential integrity programs indicate the staging database is clean, you may now proceed to the next step – [tidy balances](#).

Submitting Referential Integrity Validation Programs

The referential integrity validation programs described under [Master Data](#) and [Transaction Data](#) (in the Table Names matrices) are submitted using a batch driver program, **CIPVRNVB**, and this program is executed in the staging database. Please note that the referential integrity validation programs may also be run in the production environment on occasion, to determine the integrity of data in the production database.

Refer to [Referential Integrity Validation Processes](#) for information about these processes and their parameters.

You should supply the following parameters to this program:

- **Batch code.** The batch code associated with the appropriate table's referential integrity validation program. Refer to each table listed under [Master Data](#) and [Transaction Data](#) (in the Table Names matrices) for each referential integrity batch code / program.
- **Batch thread number.** Thread number is not used and should be left blank.
- **Batch thread count.** Thread count is not used and should be left blank.
- **Batch rerun number.** Rerun number is not used and should be left blank.
- **Batch business date.** Business date is the date supplied to the referential integrity validation programs and the date under which statistics will be logged.
- **Total number of commits.** Total number of commits is not used and should be left blank.
- **Maximum minutes between cursor re-initiation.** Maximum minutes between cursor re-initiation is not used and should be left blank.
- **User ID.** User ID is only used to log statistics for the execution of the batch job.
- **Password.** Password is not used.
- **Language Code.** Language code is used to access language-specific control table values. For example, error messages are presented in this language code.

- **Trace program at start (Y/N), trace program exit (Y/N), trace SQL (Y/N) and output trace (Y/N).** These switches are only used during QA and benchmarking. If trace program start is set to Y, a message is displayed whenever a program is started. If trace program at exist is set to Y, a message is displayed whenever a program is exited. If trace SQL is set to Y, a message is displayed whenever an SQL statement is executed.

Recommendations To Speed Up Validation Programs

The following points describe ways to accelerate the execution of the validation programs:

- Ensure that statistics are recalculated after data has been inserted into the staging tables. For Oracle users, we strongly recommend using the Oracle-provided PL/SQL package to generate statistics rather than the analyze command.
- [Object validation programs](#) should be run multi threaded.
- Execute shorter running validation processes (e.g., less records) first so that the error data can be analyzed while other processes are busy running.
- [Referential integrity validation programs](#) run fairly quickly without much tuning. However, additional benefits are gained by running several programs at the same time.
- Remember that the [object validation programs](#) can be run in “validate every nth row”. We recommend running these programs using a largish value for this parameter until the pervasive problems have been rectified.

Tidy Balances

This background process creates adjustments that cause each SA's current and payoff balances to equal its balance in the legacy system (note: the batch control ID of **CNV-BAL** is used for this process).

Submitting this process. You submit this process in the staging database. Refer to [Tidy Balances](#) for a description of this process and its parameters.

You supply the desired balances to this background process in a flat file in the following format:

Field	Size	Description
SA ID	X10	The unique identifier of the service agreement
Payoff Balance Sign	X1	Positive or Negative value indicates debit or credit balance respectively
Payoff Balance	N15.2	The SA's payoff balance (how much the customer really owes).
Current Balance New Charge Sign	X1	Positive or Negative value indicates debit or credit balance respectively
Current Balance – New Charge	N15.2	The amount of the SA's current balance that is considered a new charge, i.e., it hasn't started aging yet.
Current Balance Amount 1 Sign	X1	Positive or Negative value indicates debit or credit balance respectively
Current Balance – Amount 1	N15.2	The amount of the SA's current balance that is X days old (X is defined in the next field)
Age of Current Balance –	N3	The number of days old the prior field is (if you keep your debt in

Amount 1		"buckets" as opposed to knowing the exact number of days it has aged, you will have to choose an exact age). Set this value to zero if the value of amount 1 should be considered a "new charge" (i.e., it should only start aging when it is swept onto the next bill produced for the SA's account)
Current Balance Amount 2 Sign	X1	Positive or Negative value indicates debit or credit balance respectively
Current Balance – Amount 2	N15.2	The amount of the SA's current balance that is X days old (X is defined in the next field)
Age of Current Balance – Amount 2	N3	The number of days old the prior field is (if you keep your debt in "buckets" as opposed to knowing the exact number of days it has aged, you will have to choose an exact age)
Current Balance Amount 3 Sign	X1	Positive or Negative value indicates debit or credit balance respectively
Current Balance – Amount 3	N15.2	The amount of the SA's current balance that is X days old (X is defined in the next field)
Age of Current Balance – Amount 3	N3	The number of days old the prior field is (if you keep your debt in "buckets" as opposed to knowing the exact number of days it has aged, you will have to choose an exact age)
Current Balance Amount 4 Sign	X1	Positive or Negative value indicates debit or credit balance respectively
Current Balance – Amount 4	N15.2	The amount of the SA's current balance that is X days old (X is defined in the next field)
Age of Current Balance – Amount 4	N3	The number of days old the prior field is (if you keep your debt in "buckets" as opposed to knowing the exact number of days it has aged, you will have to choose an exact age)
Current Balance Amount 5 Sign	X1	Positive or Negative value indicates debit or credit balance respectively
Current Balance – Amount 5	N15.2	The amount of the SA's current balance that is X days old (X is defined in the next field)
Age of Current Balance – Amount 5	N3	The number of days old the prior field is (if you keep your debt in "buckets" as opposed to knowing the exact number of days it has aged, you will have to choose an exact age)

Submitting this process. You submit this process in the staging database. Refer to [Tidy Balances](#) for a description of this process and its parameters.

Balance Control (a)

During this step, you run the balance control programs and then verify that the balances that it generates are consistent with the balances in your legacy system.

Submitting this process. You submit this process in the staging database. Refer to [The Big Picture of Balance Control](#) for more information about the balance control processes. Refer to [Balance Control](#) for information about the page used to view the balances generated by this process.

Clear FT Balance Control

In the previous step, the system created a balance control and links it to the FT's. If the balance control's balances are consistent with the amount of receivables being transferred into the system, you should run the Clear FT's Balance Control program. This program simply resets the Balance Control column on the FT so that the FT's can be included in a balance control (see the last step below) after they have been transferred to production. Note: the batch control ID of **CNV-BCG** is used to request this process.

Submitting this process. You submit this process in the staging database. Refer to [Reset Balances](#) for a description of this process and its parameters.

Allocate Production Keys

The topics in this section describe the background processes used to assign production keys to the staging data.

Contents

- [The Big Picture of Key Assignment](#)
- [Submitting Key Assignment Programs](#)
- [Recommendations To Speed Up Key Generation Programs](#)

The Big Picture of Key Assignment

It's important to understand that the system does not overwrite the prime-keys on the rows in the staging database, as this is a very expensive IO transaction. Rather, a series of tables exist that hold each row's old key and the new key that will be assigned to it when the row is [transferred into the production database](#). We refer to these tables as the "old key / new key" tables. The old key / new key tables are named the same as their primary table, but rather than being prefixed by "CI", they are prefixed by "CK". For example, the old key / new key table for CI_ACCT is called CK_ACCT.

The insertion programs that transfer the rows into the production database use the new key for the main record of the key along with any other record where this key is a foreign key. Note that the capability of assigning the new key to a foreign key applies to

- "True" foreign keys, i.e. where the key is a column in another table. For example, CI_SA has a column for ACCT_ID.
- FK reference characteristics. These are characteristics that define, through an FK reference, the table and the key that this characteristic represents.

The insertion programs are not able to assign "new keys" to foreign keys defined in an XML structure field (CLOB).

The key assignment programs listed under [Master Data](#) and [Transaction Data](#) (in the table names sections) are responsible for populating the old key / new key tables (i.e., you don't have to populate these tables). Because the population of the rows in these tables is IO intensive, we have supplied detailed instructions that will accelerate the execution time of these programs.

Why are keys reassigned? The conversion process allocates new prime keys to take advantage of the system's parallel processing and data-clustering techniques in the production system (these techniques are dependent on randomly assigned, clustered keys).

Iterative conversions. Rather than perform a "big bang" conversion (one where all customers are populated at once), some implementations have the opportunity to go live on subsets of their customer base. If this describes your implementation, please be aware that the system takes into account the existing prime keys in the production database before it allocates a new key value. This means when you convert the next subset of customers, you can be assured of getting clean keys.

Program Dependencies. The programs used to assign production keys are listed in the Table Names matrices. Most of these programs have no dependencies (i.e., they can be executed in any order you please). The exceptions to this statement are noted in [Program Dependencies](#).

Submitting Key Assignment Programs

The key assignment programs described under [Master Data](#) and [Transaction Data](#) (in the Table Names matrices) are submitted using a batch driver program, **CIPVRNKB**, and this program is executed in the staging database. You should supply the following parameters to this program:

- **Batch code.** The batch code associated with the appropriate table's key assignment program. Refer to each table listed under [Master Data](#) and [Transaction Data](#) (in the Table Names matrices) for each key assignment batch code / program.
- **Batch thread number.** Thread number is not used and should be left blank.
- **Batch thread count.** Thread count is not used and should be left blank.
- **Batch rerun number.** Rerun number is not used and should be left blank.
- **Batch business date.** Business date is the date supplied to the key assignment programs and the date under which statistics will be logged.
- **Total number of commits.** Total number of commits is not used and should be left blank.
- **Maximum minutes between cursor re-initiation.** Maximum minutes between cursor re-initiation is not used and should be left blank.
- **User ID.** User ID is only used to log statistics for the execution of the batch job.
- **Password.** Password is not used.
- **Language Code.** Language code is used to access language-specific control table values. For example, error messages are presented in this language code.

- **Trace program at start (Y/N), trace program exit (Y/N), trace SQL (Y/N) and output trace (Y/N).** These switches are only used during QA and benchmarking. If trace program start is set to Y, a message is displayed whenever a program is started. If trace program at exist is set to Y, a message is displayed whenever a program is exited. If trace SQL is set to Y, a message is displayed whenever an SQL statement is executed.
- **Mode.** The proper use of this parameter will greatly speed up the key assignment step as described under [Recommendations To Speed Up Key Generation](#). This parameter has three values:
 - If you supply a mode with a value of **I** (initial key generation), the system allocates new keys to the rows in the staging tables (i.e., it populate the respective old key / new key table).
 - If you supply a mode with a value of **D** (resolve duplicate keys), the system reassigns keys that are duplicates.
 - If you supply a mode with a value of **B** (both generate keys and resolve duplicates), the system performs both of the above steps. This is the default value if this parameter is not supplied.
 - Please see [Recommendations To Speed Up Key Generation](#) for how to use this parameter to speed up the execution of these processes.

Parallel Key Generation. Note well, no key generation program should be run (either in mode **I** or **B**) while another program is being run unless that program is in the same tier (see [Program Dependencies](#) for a description of the tiers).

- **Start Row Number.** This parameter is only used if you are performing conversions where data already exists in the tables in the production database (subsequent conversions). In an Oracle database the key assignment routines create the initial values of keys by manipulation of the Oracle row number, starting from 1. After any conversion run, a subsequent conversion run will start with that row number again at 1, and the possibility of duplicate keys being assigned will be higher. The purpose of this parameter is to increase the value of row number by the given value, and minimize the chance of duplicate key assignment.

Recommendations To Speed Up Key Generation Programs

The following points describe ways to accelerate the execution of the key generation programs.

Naming convention. The convention "CK_<table_name>" is used to denote the old key / new key tables described under [The Big Picture of Key Assignment](#).

- Make the size of your rollback segments large. The exact size is dependent on the number of rows involved in your conversion. Our research has shown that processing 7 million rows generates roughly 3GB of rollback information.
 - Setup the rollback segment(s) to about 10GB with auto extend to a maximum size of 20GB to determine the high water mark
 - A next extent value on the order of 100M should be used.
- Make sure to turn off all small rollback segments (otherwise Oracle will use them rather than the large rollback segments described above).

- After the key assignment programs execute, you can reclaim the space by:
 - Keep a low value for the “minimum extent” parameter for the rollback.
 - Shrink the rollback segments and the underlying data files at the end of the large batch jobs.
- Compute statistics on the CK_<table_name> tables after every 50% increase in table size. Key generation is performed in tiers or steps because of the inheritance dependency between some tables and their keys. Although key generation for the tier currently being processed is performed by means of set-based SQL, computation of statistics between tiers will allow the database to compute the optimum access path to the keys being inherited from the **previous** tier's generation run.
For Oracle users, we strongly recommend using the Oracle-provided PL/SQL package to generate statistics rather than the analyze command.
- Optimal use of the **Mode** parameter under [Submitting Key Assignment Programs](#).
 - Before any key assignments, alter both the “old key” CX_ID index and the “new key” CI_ID index on the CK_<table_name> tables to be unusable.
 - Run all [key assignment tiers](#), submitting each job with MODE = “I”.
 - Rebuild the CX_ID and CI_ID indexes on the CK_<table_name>. Rebuilding the indexes using both the PARALLEL and NOLOGGING parameters will speed the index creation process in an Oracle DB. Statistics should be computed for these indexes.
 - Run all key assignment tiers that were previously run in MODE = ‘I’, submitting each job with MODE = “D”. This will reassign all duplicate keys.

Insert Production Data

The topics in this section describe the background processes used to populate the production database with the information in the staging database.

Contents

[The Big Picture Of Insertion Programs](#)
[Submitting Insertion Programs](#)
[Recommendations To Speed Up Insertion Programs](#)

The Big Picture Of Insertion Programs

All insertion programs are independent and may run concurrently. Also note, all insertion programs can be run in many parallel threads as described in the next section (in order to speed execution).

Submitting Insertion Programs

The insertion programs described under [Master Data](#) and [Transaction Data](#) (in the Table Names matrices) are submitted using a batch driver program, **CIPVRNIB**, and this program is executed in the staging database. You should supply the following parameters to this program:

- **Batch code.** The batch code associated with the appropriate table's insertion program. Refer to each table listed under [Master Data](#) and [Transaction Data](#) (in the Table Names matrices) for each insertion batch code / program.

- **Batch thread number.** Thread number contains the relative thread number of the process. For example, if you want to insert accounts into production in 20 parallel threads, each of the 20 execution instances receives its relative thread number (1 through 20). Refer to [Parallel Background Processes](#) for more information.
- **Batch thread count.** Thread count contains the total number of parallel threads that have been scheduled. For example, if the account insertion process has been set up to run in 20 parallel threads, each of the 20 execution instances receives a thread count of 20. Refer to [Parallel Background Processes](#) for more information.
- **Batch rerun number.** Rerun number is not used and should be left blank.
- **Batch business date.** Business date is the date supplied to the insertion programs and the date under which statistics will be logged.
- **Total number of commits.** This is the number of commits IN TOTAL that you want to perform. For example, if you have 1,000,000 accounts and you supply a value of **100**, then a commit will be executed for approximately every 10,000 accounts.
- **Maximum minutes between cursor re-initiation.** This should only be populated if you want to override the default value of **15**.
- **User ID.** User ID is only used to log statistics for the execution of the batch job.
- **Password.** Password is not used.
- **Language Code.** Language code is used to access language-specific control table values. For example, error messages are presented in this language code.
- **Trace program at start (Y/N), trace program exit (Y/N), trace SQL (Y/N) and output trace (Y/N).** These switches are only used during QA and benchmarking. If trace program start is set to Y, a message is displayed whenever a program is started. If trace program at exist is set to Y, a message is displayed whenever a program is exited. If trace SQL is set to Y, a message is displayed whenever an SQL statement is executed.

Recommendations To Speed Up Insertion Programs

The following points describe ways to accelerate the execution of the insertion programs:

- Before running the first insertion program:
 - Rebuild the index on the prime key on the old key / new key table (i.e., those tables prefixed with "CK").
 - Re-analyze the statistics on the old key / new key table (i.e., those tables prefixed with "CK"). For Oracle users, we strongly recommend using the Oracle-provided PL/SQL package to generate statistics rather than the analyze command.
 - Alter all indexes on the production tables being inserted into to be unusable.
- After the insertion programs have populated production data, rebuild the indexes and compute statistics for these tables. For Oracle users, we strongly recommend using the Oracle-provided PL/SQL package to generate statistics rather than the analyze command.

Run Balance Control Against Production

During this step, you rerun the balance control program, but this time against production. You do this to verify the balances in production are consistent with the values of receivables converted from your legacy application.

Submitting this process. You submit this process in the production database. Refer to [The Big Picture of Balance Control](#) for more information about the balance control processes. Refer to [Balance Control](#) for information about the page used to view the balances generated by this process.

Validate Production

During this step, you rerun the [object validation programs](#), but this time in production. We recommend rerunning these programs to confirm that the insertion programs have executed successfully. We recommend running these programs in random sample mode (e.g., validate every 1000th object) rather than conducting a full validation in order to save time. However, if you have time, you should run these programs in full validation mode (to validate every object). Please refer to the various “Table Names” sections above for the respective names of the programs to run.

The Validation User Interface

The topics in this section describe the various pages that assist in the conversion effort.

Contents

- [Validation Error Summary](#)
- [Validation Error Detail](#)
- [FK Validation Summary](#)
- [FK Validation Detail](#)

Validation Error Summary

Navigate to the **Admin** menu and open **Validation Error Summary** to view validation errors associated with the objects defined in [Master Data](#).

Description of Page

You can use **Table Name** to restrict errors to a specific object. If this field is left blank, all errors on all objects will be displayed.

The grid contains a separate row for each type of error. The following information is displayed:

- **Table Name** is the name of the main table associated with the object.
- **Message Category** and **Message Number** define the type of error. These fields are the unique identifier of the message that describes the error (the verbiage of this message is displayed in the **Message Text** column).
- **Count** contains the number of records with this error. Press the Go To button to see the individual records with the error.

Validation Error Detail

This page is used to view validation errors of a given type associated with one of the objects defined in [Master Data](#). This transaction is not intended to be invoked from the **Admin** menu. Rather, drill into the validation details from [Validation Error Summary](#).

Description of Page

Use **Table Name**, **Message Category**, and **Message Number** to define the object and the type of error you wish to display. The grid contains a separate row for each object with the given type of error. The following information is displayed:

- **Table Name** is the name of the main table associated with the object.
- **Record Identifier** is the unique identifier of the object with the error (e.g., the person ID, the account ID, the premise ID, etc.). Press the Go To button to transfer to the maintenance page associated with the object.
- **Message Category** and **Message Number** define the type of error. These fields are the unique identifier of the message that describes the error (the verbiage of this message is displayed in the **Message Text** column).

FK Validation Summary

Navigate to the **Admin** menu and open **FK Validation Summary** to view foreign key validation errors associated with the objects defined in [Master Data](#).

Description of Page

You can use **Table Name** to restrict errors to a specific object. If this field is left blank, all errors on all objects will be displayed.

The grid contains a separate row for each type of error. The following information is displayed:

- **Table Name** is the name of the main table associated with the object.
- **Count** contains the number of records on this table that have this error.
- **Foreign Key Field Names 1 to 6** contain the names of the foreign keys contained on this table that have been found to be in error.
- **Foreign Key Values 1 to 6** contain the values within the foreign key fields that are found to be in error.

FK Validation Detail

This page is used to view foreign key validation errors of a given type associated with one of the objects defined in [Master Data](#). This transaction is not intended to be invoked from the **Admin** menu. Rather, drill into the validation details from [FK Validation Summary](#).

Description of Page

Use **Table Name** to specify the table you wish to view. The names and values of the foreign key fields on the table are displayed. The grid that follows contains the primary key values of this table's records that are in error. The following information is displayed:

- **Table Name** is the name of the main table.
- **FK Fields 1 to 6** are the names of the foreign keys contained in this table. Displayed alongside the key names are the values within these fields. These identify records on other tables to which this table's record is related. For example, the CI_PREM_GEO record identified by its displayed primary keys should be related to a Premise record with the Premise ID shown – it appears in this list only if there is something amiss with this relationship.

The Staging Tables

This section describes the objects into which your legacy data is mapped. For each object, we provide the following:

- A data model.
- An indication of which tables have system-assigned keys.
- The physical table names.
- The name of the batch control to submit to validate the object.
- The name of the program (and related batch control) that validates each table for referential integrity.
- The name of the program (and related batch control) that performs key assignment for each table.
- The name of the program (and related batch control) that inserts the table's rows into production from staging.
- Suggestions to assist in the conversion process.

Recommendation. We recommend you read this document on a browser (or using Word under windows) so you can take advantage of the [Color Coding](#).

Column details do not appear in this document. When you're ready to examine an object's tables, use the hyperlinks in the respective Table Names section to transfer to the [data dictionary](#). The data dictionary will show you the required columns, the foreign keys (and their related tables), the source code of the program that validates the contents of the table, and a host of other information that will assist the conversion process.

Look Up and Control Tables. In the data models that appear below, you will find a variety of entities that are classified as either a control table or a lookup table. Please refer to [Color Coding](#) for more information about how to recognize such an entity.

Contents

[A Note About Programs in the Table Names Matrices](#)
[Master Data](#)
[Transaction Data](#)
[Program Dependencies](#)

A Note About Programs in the Table Names Matrices

For each object described in the master data and transaction data sections, there is a "table names" section that includes a matrix listing the name of each table that is part of the maintenance object. Included in the matrix is information about the programs provided to perform object validation, referential integrity validation, key assignment and insertion. The following are some points about these programs:

- One object validation program exists for the entire set of tables for the maintenance object. The **Object Validation Batch Control** column indicates the batch control used to submit the object validation. Refer to [Submitting Object Validation Programs](#) for more information. Drilling down on the hypertext allows you to see more information about the batch control, including the program associated with it.
- A referential integrity validation program exists for every table whose key includes a parent key of another object. As described in [Submitting Referential Integrity Validation Programs](#), these programs are submitted using a driver supplied by the system where the batch code for the appropriate table is provided. (The driver then executes the program whose name matches the batch code). The **Referential Integrity Validation Batch Control** column indicates the table's batch control / program name.
- One key assignment program exists for the parent table for the maintenance object. As described in [Submitting Key Assignment Programs](#), these programs are submitted using a driver supplied by the system where the batch code for the appropriate table is provided. (The driver then executes the program whose name matches the batch code). The **Key Assignment Batch Control** column indicates the table's batch control / program name.
- An insertion program exists for every table for the maintenance object. As described in [Submitting Insertion Programs](#), these programs are submitted using a driver supplied by the system where the batch code for the appropriate table is provided. (The driver then executes the program whose name matches the batch code). The **Insertion Batch Control** column indicates the table's batch control / program name.

Master Data

This section describes the various “master data” objects (e.g., person, account, meter, etc.) that must be created before you can convert transaction data.

Key Assignment Dictates The Order Of Conversion. The following contents are listed in the order in which the objects should be converted in order to maintain referential integrity.

Contents

- [Person](#)
- [Account](#)
- [Item](#)
- [Landlord](#)
- [Meter](#)
- [Meter Configuration](#)
- [Premise](#)
- [Service Point](#)
- [Service Agreement](#)
- [SA Interval Billing](#)
- [Contract Options](#)
- [Service Credit Membership](#)
- [Declaration](#)

Person

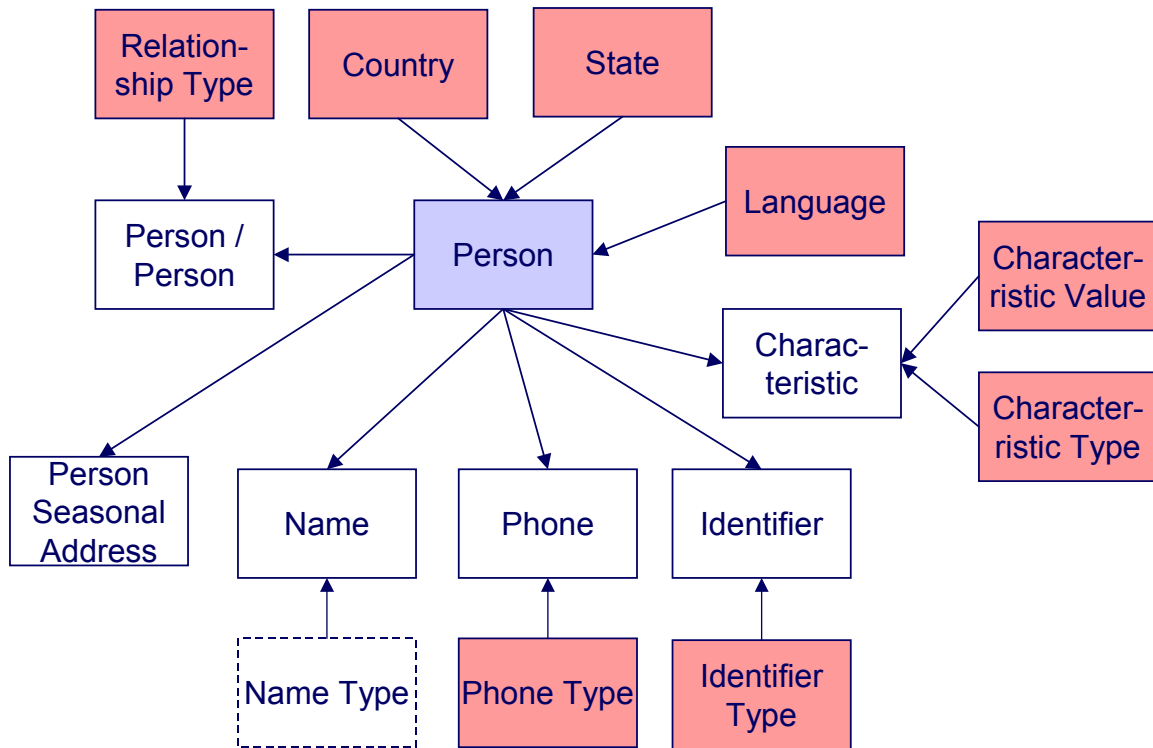
Each customer must have a person and an account object. This section describes the person object. Refer to [Account](#) for details about the account object.

Contents

[Person Data Model](#)
[Person Table Names](#)
[Person Suggestions](#)

Person Data Model

The following data model illustrates the person object.

**Person Table Names**

Data Model Name	Table Name	Generated Keys	Object Validation Batch Control	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Person	CI_PER	Yes CI_PER_K	VAL-PER		CIPVPERK	CIPVPERI
Name	CI_PER_NAME	No. The key is PER_ID plus a sequence number.		CIPVPNMV		CIPVPNMI
Person / Person	CI_PER_PER	No. The key is PER_ID1, PER_ID2, relationship type and start date.		CIPVPPEV		CIPVPPEI
Phone	CI_PER_PHONE	No. The key is PER_ID plus phone type.		CIPVPPHV		CIPVPPHI

Identifier	CI_PER_ID	No. The key is PER_ID plus identifier type.		CIPVPIDV		CIPVPIDI
Characteristic	CI_PER_CHAR	No. The key is PER_ID plus an edate and a char type.		CIPVPRCV		CIPVPRCI
Seasonal Address	CI_PER_ADDR_SEAS	No. The key is PER_ID plus a sequence number.		CIPVPSAV		CIPVPSAI

Person Suggestions

A person must have at least one row on the name table and at least one of the names must be marked as being the primary name.

A person must have at least one row on the identity table and at least one of the identities must be marked as being the primary ID.

The country and state are only necessary if the person has an override mailing address.

Account

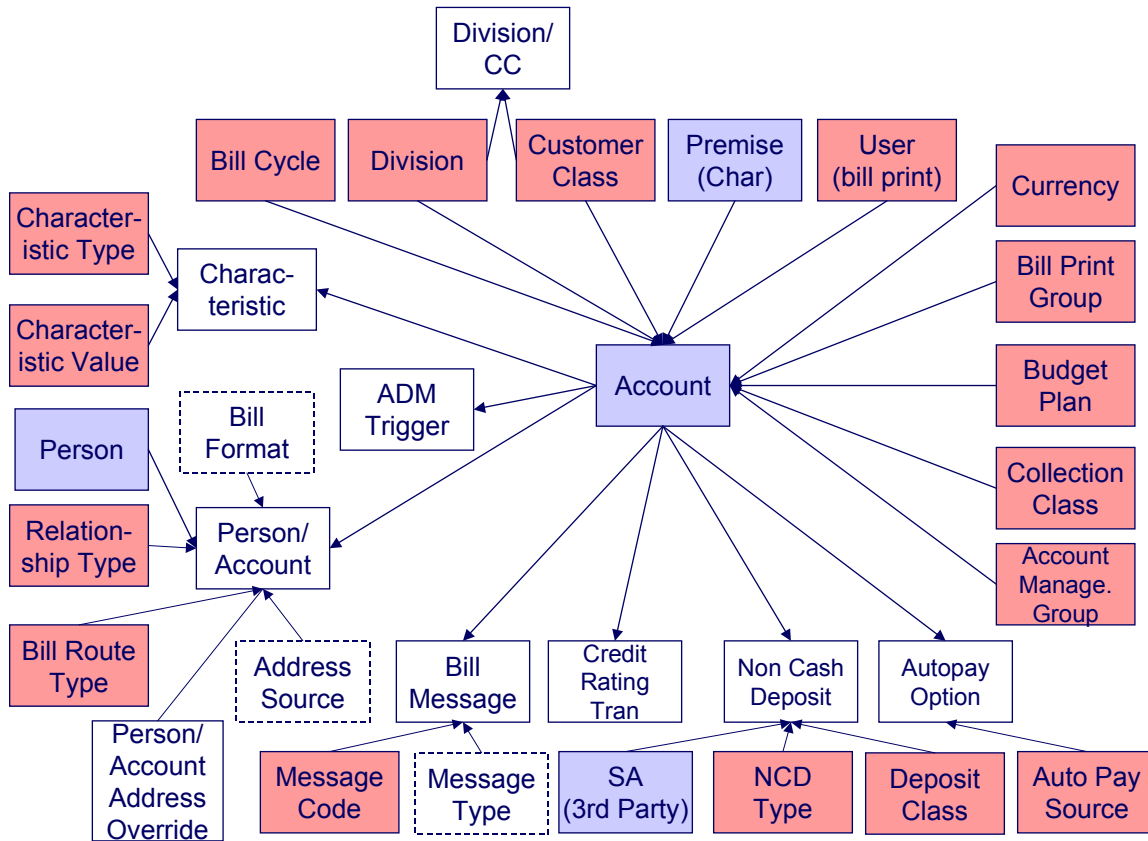
Each customer must have a person and an account object. This section describes the account object, refer to [Person](#) for details about the person object.

Contents

- [Account Data Model](#)
- [Account Table Names](#)
- [Account Suggestions](#)

Account Data Model

The following data model illustrates the account object.



Account Table Names

Data Model Name	Table Name	Generated Keys	Object Validation Batch Control	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Account	CI_ACCT	Yes CI_ACCT_K	VAL-ACCT		CIPVACCK	CIPVACCI
Bill Message	CI_ACCT_MSG	No. The key is account ID plus bill message code.		CIPVMSGV		CIPVMSGI
Autopay Option	CI_ACCT_APAY	Yes CI_ACCT_APAY_K		CIPVAAPV	CIPVAAPK Has dependencies	CIPVAAPI
Characteristic	CI_ACCT_CHAR	No. The key is ACCT_ID plus an edate and a char type.		CIPVACHV		CIPVACHI
Person/Account	CI_ACCT_PER	No. The key is account ID plus person ID.		CIPVACPV		CIPVACPI
Person/Account	CI_PER_ADDR_OV	No. The key is		CIPVPAOV		CIPVPAOI

nt Address Override	RD	Account ID plus Person ID				
Non Cash Deposit	CI_NCD	No. The key is account ID plus seq number		CIPVNCDV		CIPVNCDI
Credit Rating Tran	CI_CR_RAT_HIST	Yes CI_CR_RAT_HIST_K		CIPVCRTV	CIPVCRRK Has dependencies	CIPVCRTI
ADM Trigger	CI_ADM_RVW_SC_H	No. The key is account ID plus date		CIPVARSV		CIPVARSI

Account Suggestions

An account must have at least one row on the account / person table and at least one account / person must be marked as being the main customer. Please see column notes for the account / person table for inter-field validation in respect of the various switches (e.g., if main customer switch is on, then the person must also be financially responsible).

We recommend storing an ADM trigger ([CI_ADM_RVW_SCH](#)) for every account where the trigger date is the conversion date. This will cause the account to be reviewed by the [account debt monitor](#) when it next runs. We have supplied a dedicated batch process for this purpose that simply inserts a row in this table with the review date set equal to the current date. This will ensure that all converted accounts are reviewed after they are inserted into production. This program is named CIPVADMB and goes by the batch control ID of **CNV-ADM**.

If your legacy system has the equivalent of a credit rating or a cash only score, you should create credit rating transactions. The values you create need to be consistent with the base and threshold credit rating and cash only points on the installation record. Refer to the business process guide – customer information – how are credit rating transactions created for more information.

Item

Each badged piece of equipment must have an item. Examples of items include badged street lamps, current transforms, backflow devices, voltage regulators, etc.

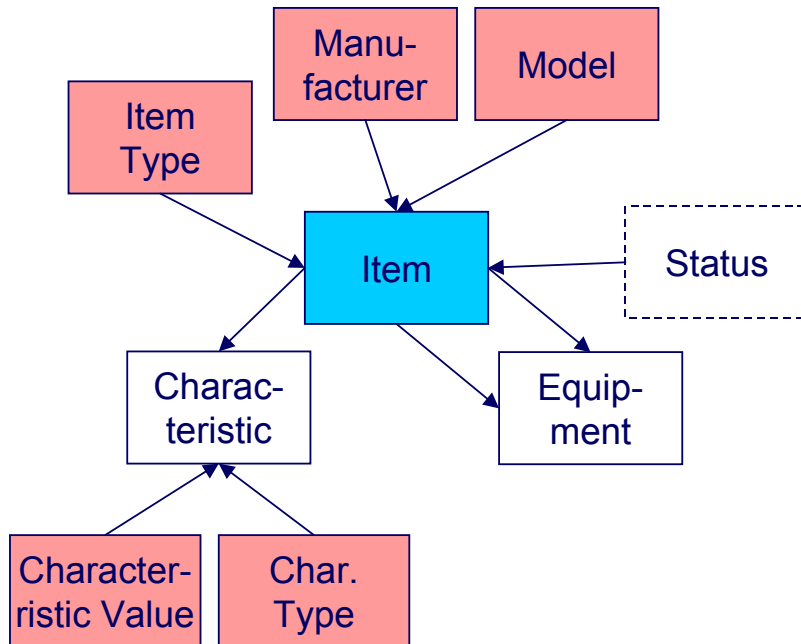
If the item is currently installed at a service point, you must link the item to the service point by inserting a row on the [SP Item History and On Off Event](#) tables.

Contents

- [Item Data Model](#)
- [Item Table Names](#)
- [Item Suggestions](#)

Item Data Model

The following data model illustrates the item object.



Item Table Names

Data Model Name	Table Name	Generated Keys	Object Validation Batch Control	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Item	CI_ITEM	Yes CI_ITEM_K	VAL-ITEM		CIPVITMK	CIPVITMI
Characteristic	CI_ITEM_CHAR	No. The key is ACCT_ID plus an edate and a char type.		CIPVITCV		CIPVITCI
Equipment	CI_ITEM_EQ	No. The key is item ID plus equipment (item) ID.		CIPVIEQV		CIPVIEQI

Item Suggestions

N/A.

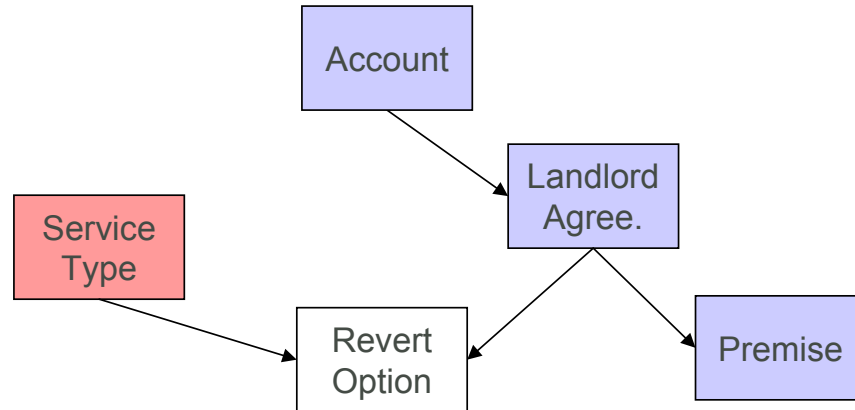
Landlord

Contents

- [Landlord Data Model](#)
- [Landlord Table Names](#)
- [Landlord Suggestions](#)

Landlord Data Model

The following data model illustrates the landlord object.



Landlord Table Names

Data Model Name	Table Name	Generated Keys	Object Validation Batch Control	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Landlord	CI_LANDLORD	Yes CI_LANDLORD_K	VAL-LL		CIPVLNDK	CIPVLNDI
Revert Option	CI_LL_DETAIL	No. The key is LL_ID plus service type		CIPVLLDV		CIPVLLDI

Landlord Suggestions

N/A.

Meter

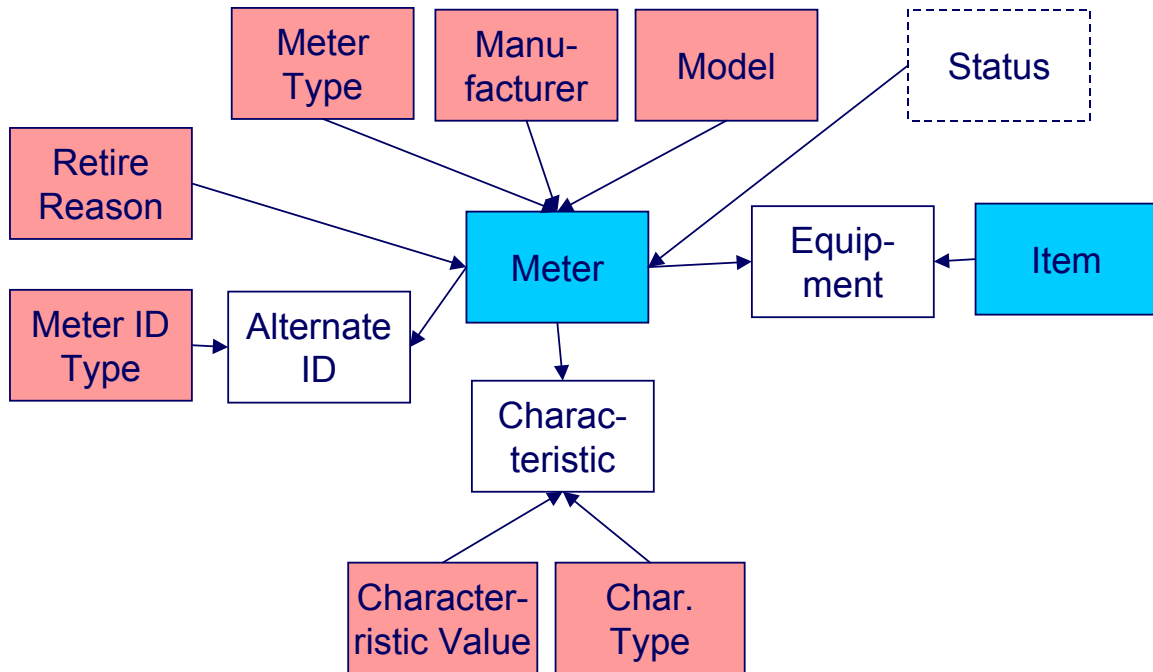
Every meter must have a meter object and every meter object must have a meter configuration. This section describes the meter object. Refer to [Meter Configuration](#) for information about the meter configuration object.

Contents

- [Meter Data Model](#)
- [Meter Table Names](#)
- [Meter Suggestions](#)

Meter Data Model

The following data model illustrates the meter object.



Meter Table Names

Data Model Name	Table Name	Generated Keys	Object Validation Batch Control	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Meter	CI_MTR	Yes CI_MTR_K	VAL-MTR		CIPVMTRK Has dependencies	CIPVMTRI
Characteristic	CI_MTR_CHAR	No. The key is MTR_ID plus an edate and a char type.		CIPVMTCV		CIPVMTCI
Equipment	CI_MTR_EQ	No. The key is meter ID plus equipment (meter) ID.		CIPVMEQV		CIPVMEQI
Alternate ID	CI_MTR_ID	No. The key is MTR_ID plus meter id type.		CIPVMIDV		CIPVMIDI

Meter Suggestions

N/A.

Meter Configuration

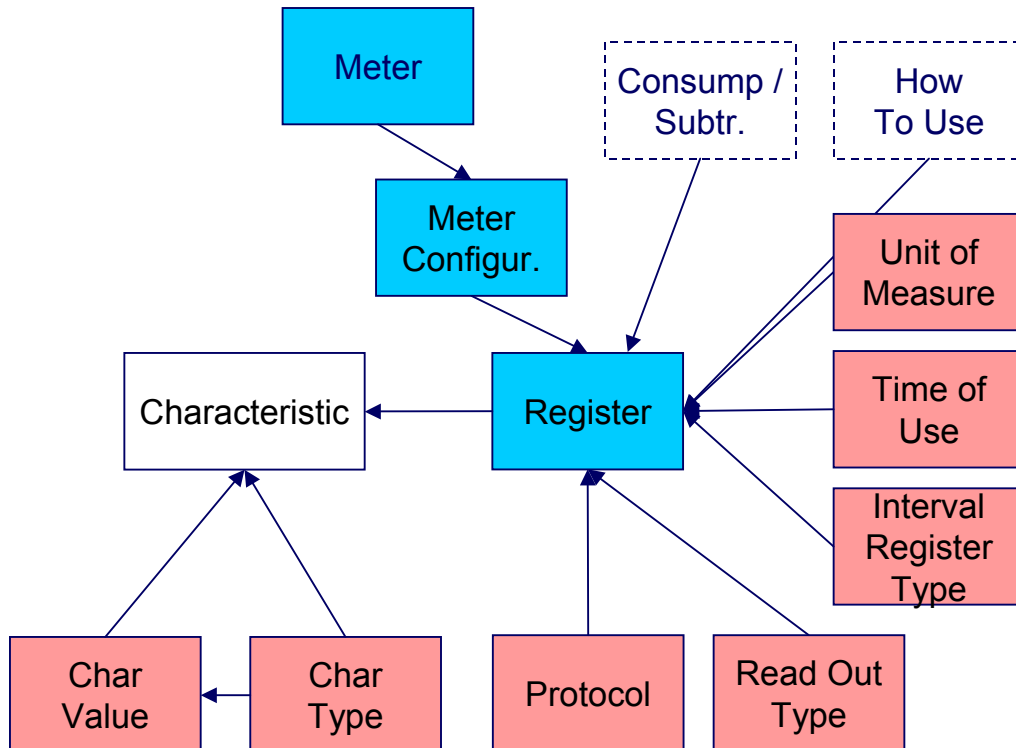
Every meter must have a meter object and every meter object must have a meter configuration. This section describes the meter configuration object. Refer to [Meter](#) for information about the meter object.

Contents

- [Meter Configuration Data Model](#)
- [Meter Configuration Table Names](#)
- [Meter Configuration Suggestions](#)

Meter Configuration Data Model

The following data model illustrates the meter configuration object.



Meter Configuration Table Names

Data Model Name	Table Name	Generated Keys	Object Validation Batch Control	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Meter Configuration	CI_MTR_CONFIG	Yes CI_MTR_CONFIG_K	VAL-CFG		CIPVMTGK Has dependencies	CIPVMTGI
Register	CI_REG	Yes CI_REG_K		CIPVREGV	CIPVREGK Has dependencies	CIPVREGI

Characteristic	CI_REG_CHAR	No. The key is REG_ID plus an edate and a char type.		CIPVRGCV		CIPVRGCI
----------------	-----------------------------	--	--	----------	--	----------

Meter Configuration Suggestions

N/A.

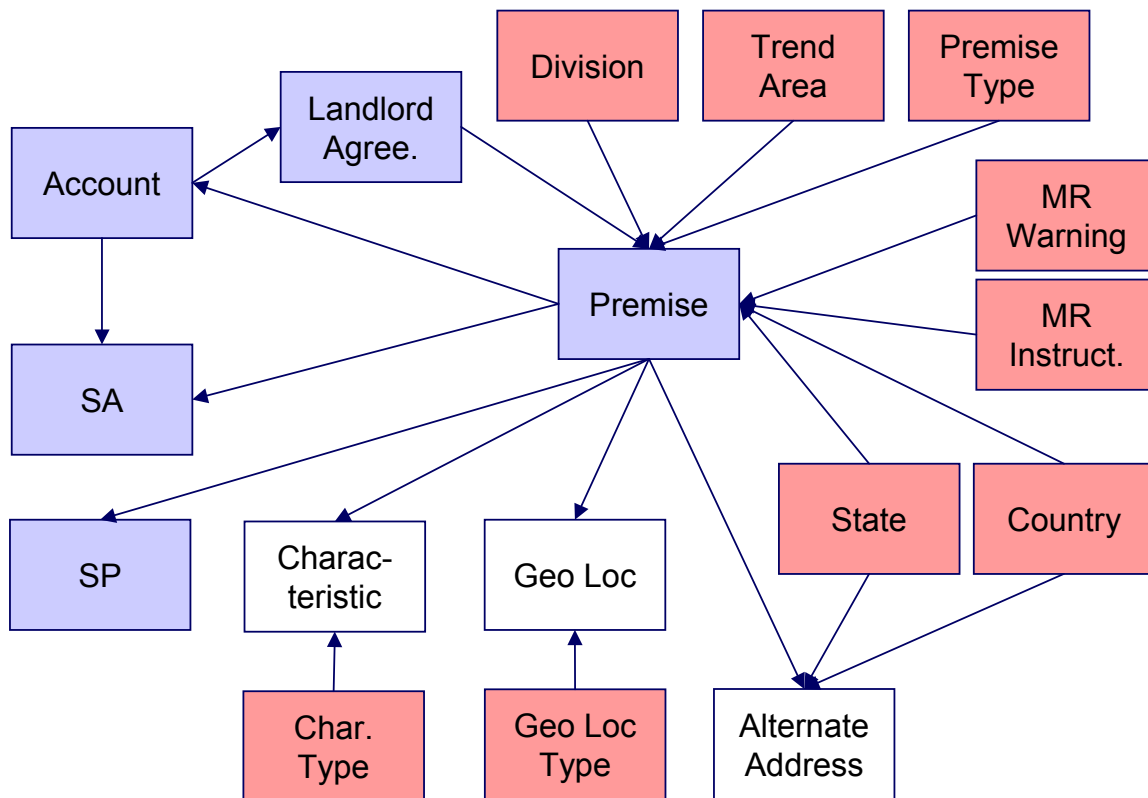
Premise

Contents

- [Premise Data Model](#)
- [Premise Table Names](#)
- [Premise Suggestions](#)

Premise Data Model

The following data model illustrates the premise object.



Premise Table Names

Data Model Name	Table Name	Generated Keys	Object Validation Batch Control	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Premise	CI_PREM	Yes CI_PREM_K	VAL-PREM		CIPVPRMK	CIPVPRMI

					Has dependencies	
Characteristic	CI_PREM_CHAR	No. The key is PREM_ID plus an edate and a char type.		CIPVPCHV		CIPVPCHI
Geo Loc	CI_PREM_GEO	No. The key is PREM_ID plus geo loc type.		CIPVPGOV		CIPVPGOI
Alternate Address	CI_PRM_ALT_ADD R	Yes CI_PRM_ALT_AD DR_K		CIPVAPAV	CIPVAPAK Has dependencies	CIPVAPAI

Premise Suggestions

N/A.

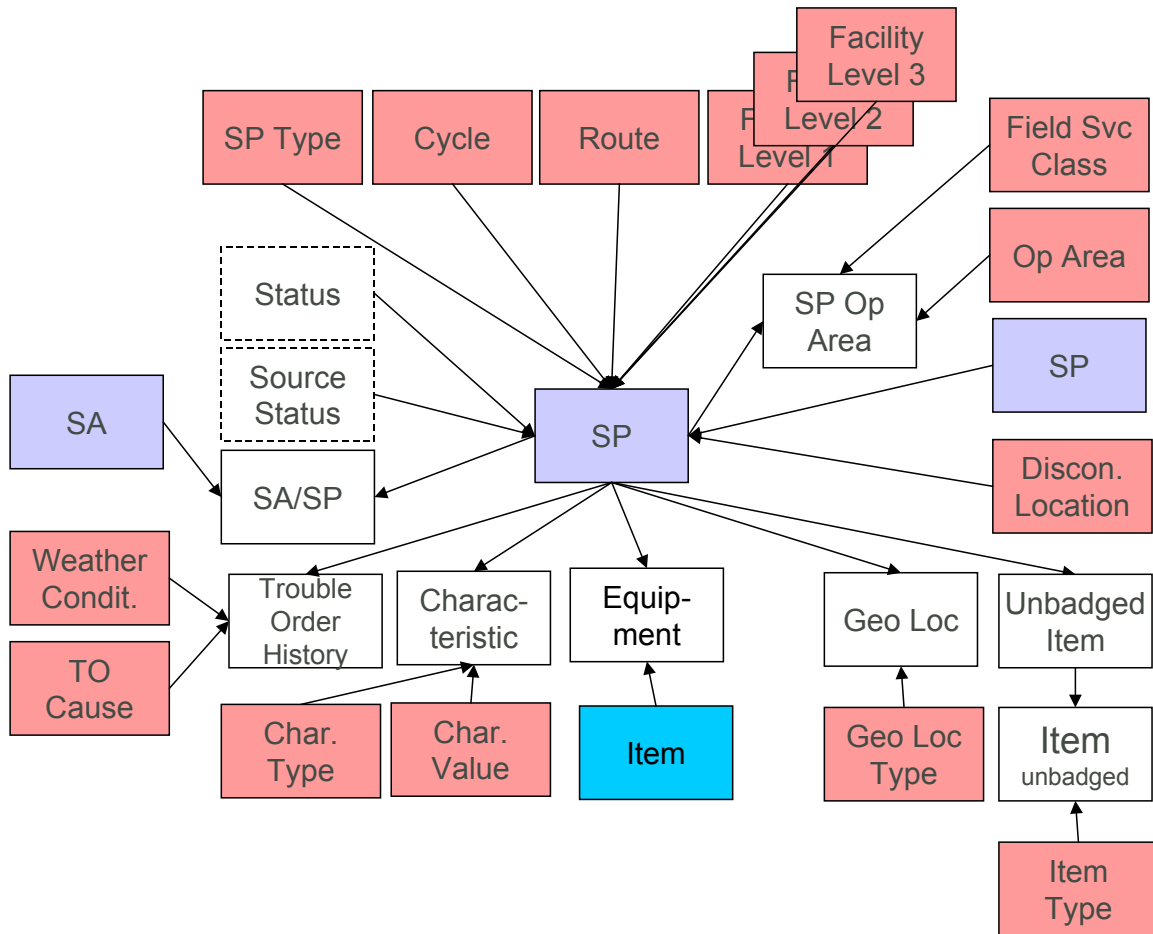
Service Point

Contents

- [Service Point Data Model](#)
- [Service Point Table Names](#)
- [Service Point Suggestions](#)

Service Point Data Model

The following data model illustrates the service point object.



Service Point Table Names

Data Model Name	Table Name	Generated Keys	Object Validation Batch Control	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Service Point	CI_SP	Yes CI_SP_K	VAL-SP		CIPVSPPK Has dependencies	CIPVSPPI
Characteristic	CI_SP_CHAR	No. The key is SP_ID plus an edate and a char type.		CIPVSPCV		CIPVSPCI
Equipment	CI_SP_EQ	No. The key is service point ID plus equipment (service point) ID.		CIPVSEQV		CIPVSEI
Geo Loc	CI_SP_GEO	No. The key is service point ID		CIPVSPGV		CIPVSPGI

		plus geo type.				
Unbadged Item	CI_SP_MULT_ITEM	No. The key is SP_ID plus edate.		CIPVSPMV		CIPVSPMI
Item	CI_MULT_ITEM	No. The key is SP_ID, edate and item type.		CIPVSMIV		CIPVSMII
SP Op Area	CI_SP_OP_AREA	No. The key is SP_ID plus field service classification code		CIPVSPOV		CIPVSPOI
SA/SP	CI_SA_SP (note, this table really belongs to the SA object, it is included here for completeness)	Yes CI_SA_SP_K	CIPVSVAB	CIPVSAPV	CIPVSSPK Has dependencies	CIPVSAPI

Service Point Suggestions

N/A.

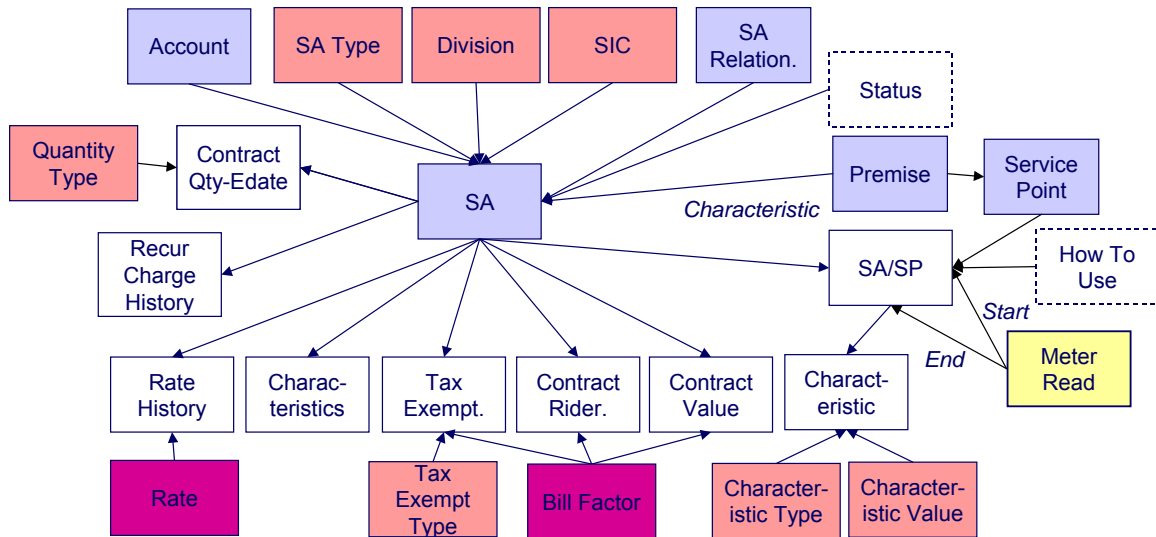
Service Agreement

Contents

- [Service Agreement Data Model](#)
- [Service Agreement Table Names](#)
- [Service Agreement Suggestions](#)

Service Agreement Data Model

The following data model illustrates the service agreement object.



Service Agreement Table Names

Data Model Name	Table Name	Generated Keys	Object Validation Batch Control	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Service Agreement	CI_SA	Yes CI_SA_K	VAL-SA		CIPVSVAK Has dependencies	CIPVSVAI
Characteristic	CI_SA_CHAR	No. The key is SA_ID plus an edate and a char type.		CIPVSACV		CIPVSACI
Contract Quantity Edate	CI_SA_CONT_QTY	No. The key is service agreement ID plus quantity type plus edate.		CIPVSAQV		CIPVSAQI
Message	CI_SA_MSG	No. The key is service agreement ID plus Bill message code.		CIPVSMGV		CIPVSMGI
Recurring Charge	CI_SA_RCHG_HIST	No. The key is service agreement ID plus edate.		CIPVSARV		CIPVSARI
SA Relationship	CI_SA_REL	Yes CI_SA_REL_K	VAL-SARL	CIPVSRLV	CIPVSRLK Has dependencies	CIPVSRLI
Rate History	CI_SA_RS_HIST	No. The key is service agreement ID plus edate.		CIPVSAHV		CIPVSAHI
SA/SP	CI_SA_SP	Yes CI_SA_SP_K		CIPVSAPV	CIPVSSPK	CIPVSAPI

					Has dependencies	
SA/SP Characteristic	CI_SA_SP_CHAR	No. The key is SA/SP Id plus char type plus effective date.		CIPVSSCV		CIPVSSCI
Tax Exempt	CI_SA_CONTERM – this table is also used for the next 2 entities, the key contains CONTERM_TYPE_FLG that controls the entity	No. This key is service agreement ID plus CONTERM_TYPE_FLG plus BF_CD plus START_DT		CIPVSAOV		CIPVSAOI
Contract Rider	CI_SA_CONTERM – this table is also used for the previous and next entities, the key contains CONTERM_TYPE_FLG that controls the entity	No. This key is service agreement ID plus CONTERM_TYPE_FLG plus BF_CD plus START_DT		CIPVSAOV		CIPVSAOI
Contract Value	CI_SA_CONTERM – this table is also used for the previous 2 entities, the key contains CONTERM_TYPE_FLG that controls the entity	No. This key is service agreement ID plus CONTERM_TYPE_FLG plus BF_CD plus START_DT		CIPVSAOV		CIPVSAOI

Service Agreement Suggestions

N/A.

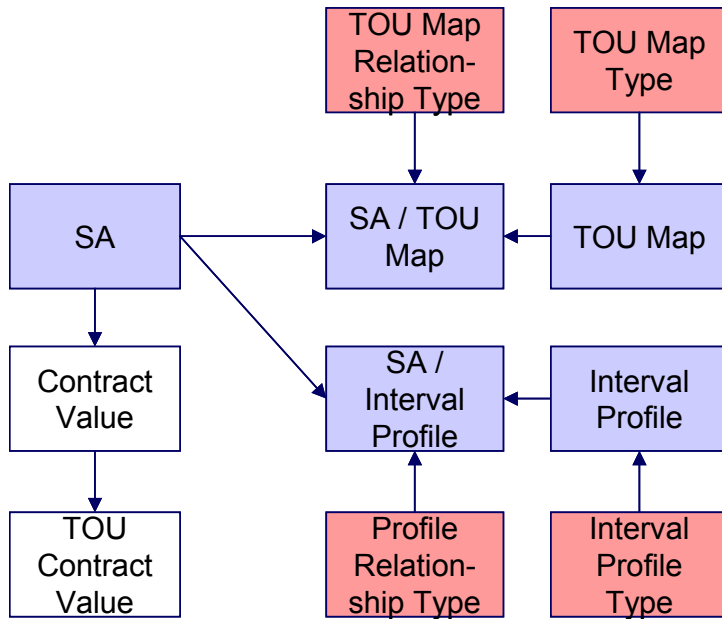
SA Interval Billing

Contents

- [SA Interval Billing Data Model](#)
- [SA Interval Billing Table Names](#)
- [SA Interval Billing Suggestions](#)

SA Interval Billing Data Model

The following data model illustrates the Interval Billing objects related to the service agreement.



SA Interval Billing Table Names

Data Model Name	Table Name	Generated Keys	Object Validation Batch Control	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
TOU Map	CI_TOU_MAP	Yes CI_TOU_MAP_K	VAL-TMAP		CIPVTMAK Has dependencies	CIPVTMAI
TOU Map Lang	CI_TOU_MAP_L	No. The key is TOU_MAP_ID plus language code.		CIPVTMLV		CIPVTMLI
Interval Profile	CI_INTV_PF	Yes CI_INTV_PF_K	VAL-INPF		CIPVINPK Has dependencies	CIPVINPI
Interval Profile Lang	CI_INTV_PF_L	No. The key is INTV_PF_ID plus language code		CIPVINLV		CIPVINLI
SA / Interval Profile	CI_SA_INTV_PF	No. The key is service agreement ID plus INTV_PF_REL_TY P_CD plus START_DTTM plus INTV_PF_ID		CIPVSIFV		CIPVSIPI
SA / TOU Map	CI_SA_TOU_MAP	No. The key is service agreement ID plus		CIPVSTMV		CIPVSTMI

		TMAP_REL_TYPE _CD plus START_DTTM plus TOU_MAP_ID				
TOU Contract Value	CI_TOU_CONT_VAL	No. The key is service agreement ID plus CONTERM_TYPE _FLG plus START_DT plus BF_CD plus TOU_GRP_CD plus TOU_CD		CIPVTCVV		CIPVTCVI

SA Interval Billing Suggestions

N/A.

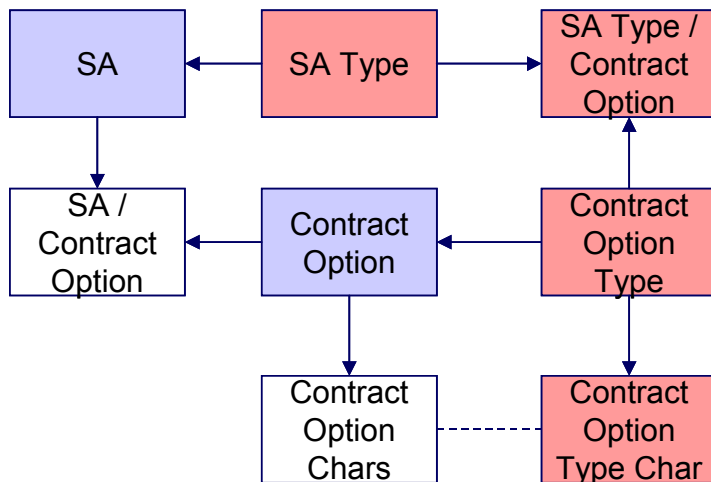
Contract Options

Contents

- [Contract Options Data Model](#)
- [Contract Options Table Names](#)
- [Contract Options Suggestions](#)

Contract Options Data Model

The following data model illustrates the contract options objects.



Contract Options Table Names

Data Model Name	Table Name	Generated Keys	Object Validation Batch Control	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Contract Option	CI_COP	Yes CI_COP_K	VAL-COP		CIPVCOPK	CIPVCOPI

					Has dependencies	
Contract Option Language	CI_COP_L	No. The key is CONT_OPT_ID plus language code		CIPVCOLV		CIPVCOLI
Contract Option Characteristics	CI_COP_CHAR	No. The key is CONT_OPT_ID plus CHAR_TYPE_CD plus EFFDT		CIPVCCFV		CIPVCCAI
SA / Contract Option	CI_SA_COP	Yes CI_SA_COP_K		CIPVSCPv	CIPVSCPv Has dependencies	CIPVSCPI

Contract Options Suggestions

N/A.

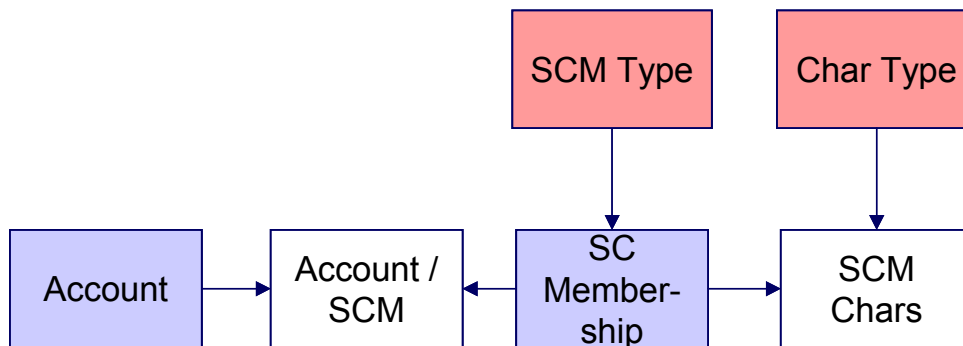
Service Credit Membership

Contents

- [Service Credit Membership Data Model](#)
- [Service Credit Membership Table Names](#)
- [Service Credit Membership Suggestions](#)

Service Credit Membership Data Model

The following data model illustrates the Service Credit Membership objects.



Service Credit Membership Table Names

Data Model Name	Table Name	Generated Keys	Object Validation Batch Control	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Service Credit Membership	CI_SCM	Yes CI_SCM_K	VAL-SCM	CIPVSCBV	CIPVSCMK	CIPVSCMI
Service Credit	CI_SCM_ACCT	No. The key is		CIPVSCAV		CIPVSCAI

Membership / Account		SCM_ID plus ACCT_ID.				
Service Credit Membership Characteristics	CI_SCM_CHAR	No. The key is SCM_ID plus CHAR_TYPE_CD plus EFFDT.		CIPVSCCV		CIPVSCCI

Service Credit Membership Suggestions

N/A.

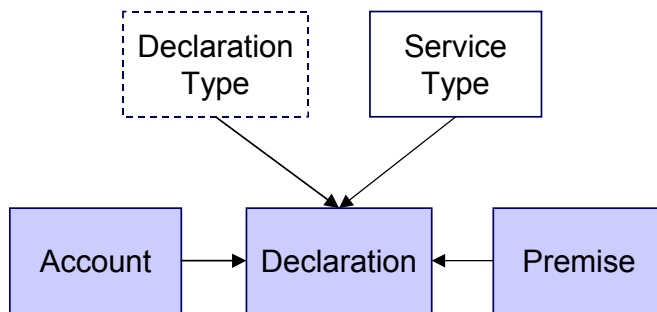
Declaration

Contents

- [Declaration Data Model](#)
- [Declaration Table Names](#)
- [Declaration Suggestions](#)

Declaration Data Model

The following data model illustrates the Declaration object.



Declaration Table Names

Data Model Name	Table Name	Generated Keys	Object Validation Batch Control	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Declaration	CI_DCL	Yes CI_DCL_K	VAL-DCL		CIPVDCRK	CIPVDCRI

Declaration Suggestions

N/A.

Transaction Data

This section describes the tables in which your transaction data (e.g., bills, payments, meter reads, customer contacts, etc.) resides.

Contents

- [SP / Meter History and On/Off Event](#)
- [SP / Item History and On/Off Event](#)

Bill
Payment
Adjustment
Customer Contact
Meter Read
Field Order
Field Activity
Interval Data
Contract Option Events
Register Interval Data
Bill Factor Value
Bill Factor Interval Prices
Bill Factor TOU Pricing
Service Credit Event
Non-Billed Budgets
Billable Charge
Collection Process
Severance Process
Write Off Process
Workflow Process
Device Test
Collection Agency Referral
Trend

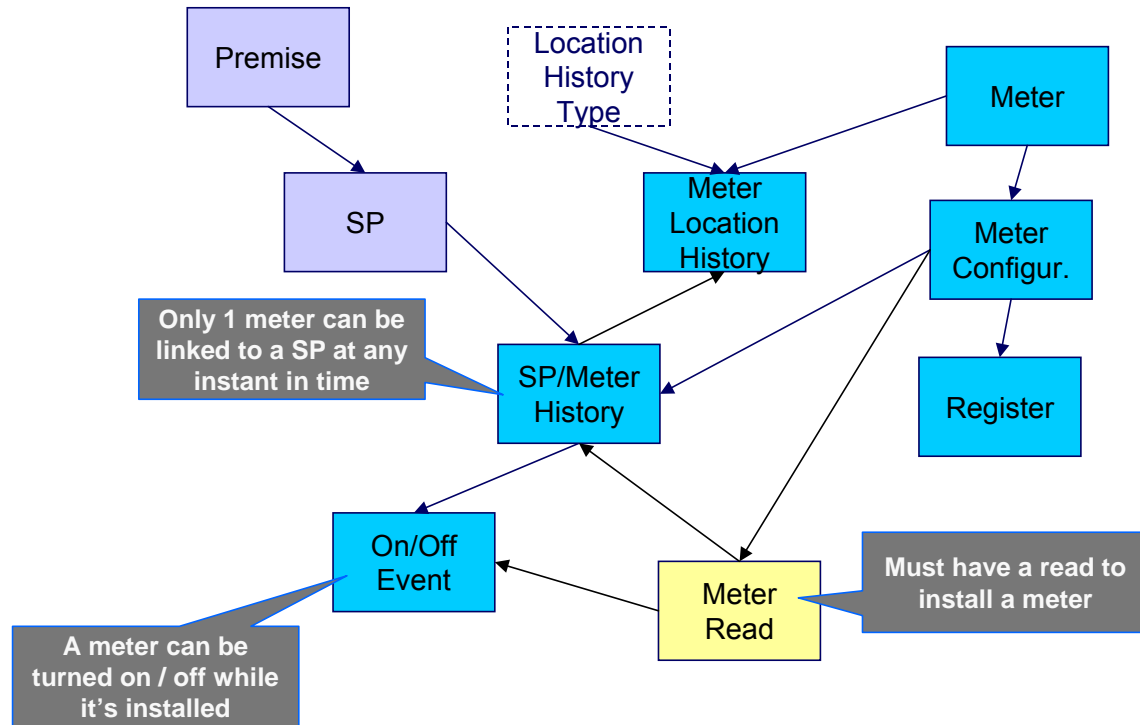
SP / Meter History and On/Off Event

Contents

SP / Meter Data Model
SP / Meter Table Names
SP / Meter Suggestions

SP / Meter Data Model

The following data model illustrates the service point / meter installation object.



SP / Meter Table Names

Data Model Name	Table Name	Generated Keys	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
SP/Meter History	CI_SP_MTR_HIST	Yes CI_SP_MTR_HIST_K	CIPVSMHV	CIPVSMHK Has dependencies	CIPVSMHI
On/Off Event	CI_SP_MTR_EVT	No. The key is meter history ID plus sequence number.	CIPVSMEV		CIPVSMEI
Meter Location History	CI_MTR_LOC_HIS	Yes. CI_MTR_LOC_HIS_K	CIPVMLHV	CIPVMLHK	CIPVMLHI

SP / Meter Suggestions

In order to link a meter to a service point, you must

- Link the meter's meter configuration to the service point by inserting a record on the CI_SP_MTR_HIST table.
- In addition, you must also create a CI_SP_MTR_EVT record. Note the following about this record:
 - The value of the SP_MTR_EVT_FLG should be I (for installation).

- The value of the MTR_ON_OFF_FLG should be 1 (on).
- It must reference a read whose read date is the installation date. The reading can be a dummy value unless this customer has not been billed since the install date (i.e., the installation has taken place recently). In this situation, this read must be the true start read for the customer. Note, this read should also be linked to the SA/SP record associated with the SA that's linked to the SP as its start read.

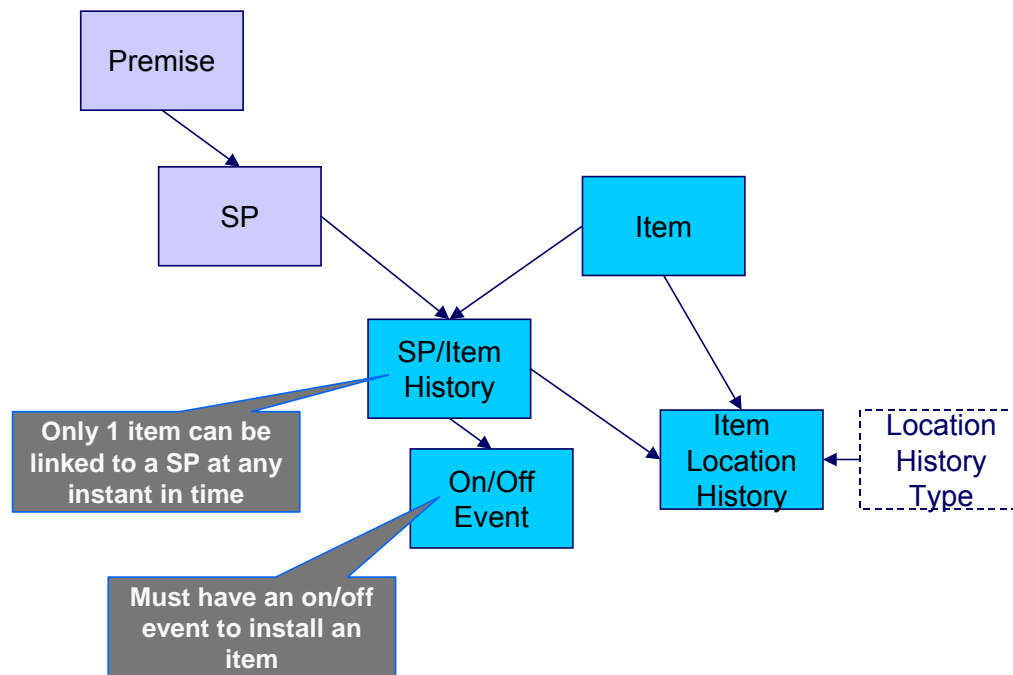
SP / Item History and On/Off Event

Contents

- [SP / Item Data Model](#)
- [SP / Item Table Names](#)
- [SP / Item Suggestions](#)

SP / Item Data Model

The following data model illustrates the service point / item installation object.



SP / Item Table Names

Data Model Name	Table Name	Generated Keys	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
SP/Item History	CI_SP_ITEM_HIST	Yes CI_SP_ITEM_HIST_K	CIPVSIHV	CIPVSIHK Has dependencies	CIPVSIHI
On/Off Event	CI_SP_ITEM_EVT	No. The key is meter history ID	CIPVSIIEV		CIPVSIIEI

		plus sequence no.			
Item Location History	CI_ITEM_LOC_HIS	Yes. CI_ITEM_LOC_HI S_K	CIPVILHV	CIPVILHK	CIPVILHI

SP / Item Suggestions

In order to link an item to a service point, you must

- Link the item to the service point by inserting a record on the CI_SP_ITEM_HIST table.
- In addition, you must also create a CI_SP_ITEM_EVT record. Note the following about this record:
 - The value of the SP_ITEM_EVT_FLG should be I (for installation).
 - The value of the ITEM_ON_OFF_FLG should be 1 (on).

Bill

Contents

[Bill Data Model](#)
[Bill Table Names](#)
[Bill Suggestions](#)

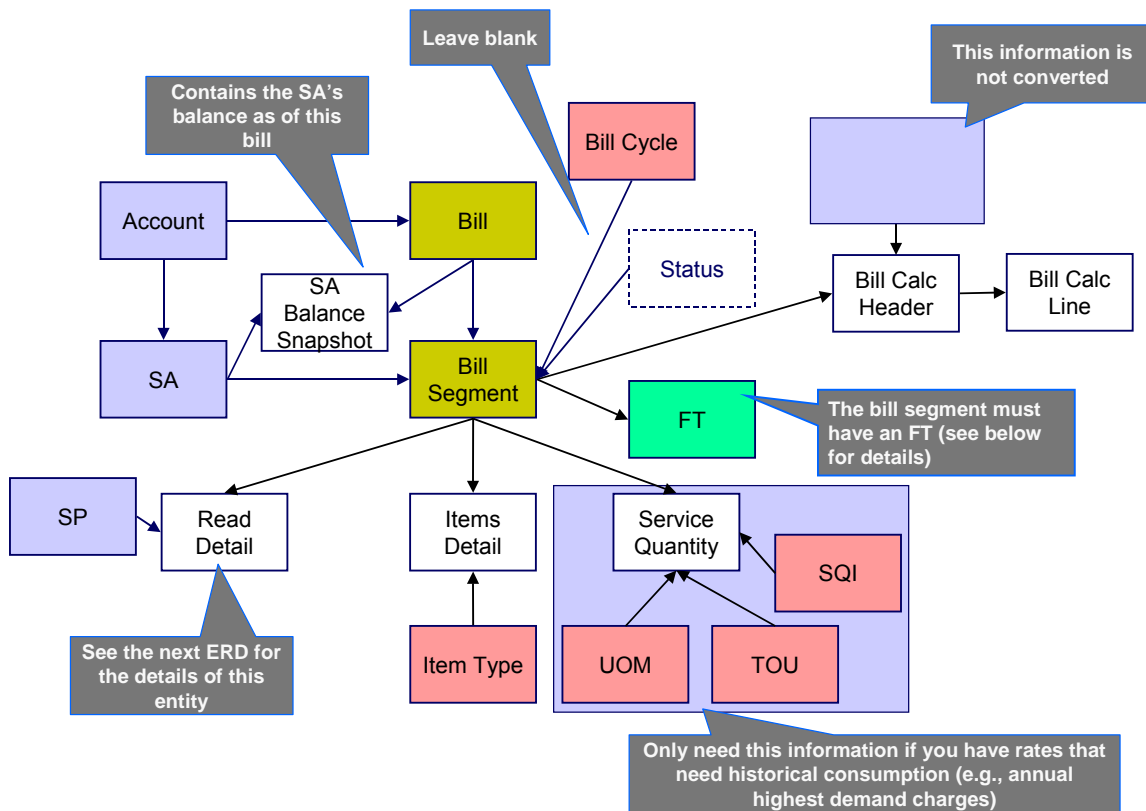
Bill Data Model

Contents

[Bill - Main](#)
[Bill - Read Details](#)
[Bill - FT](#)
[Bill Characteristics](#)
[Bill Messages, Bill Routing and Bill Review Schedule](#)

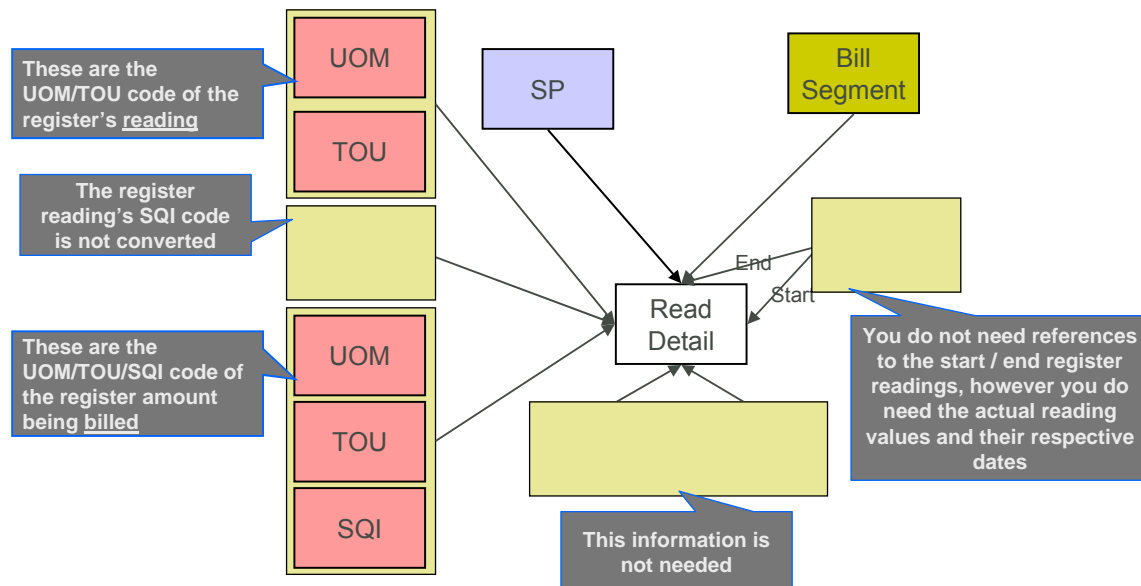
Bill - Main

The following data model illustrates the bill object.



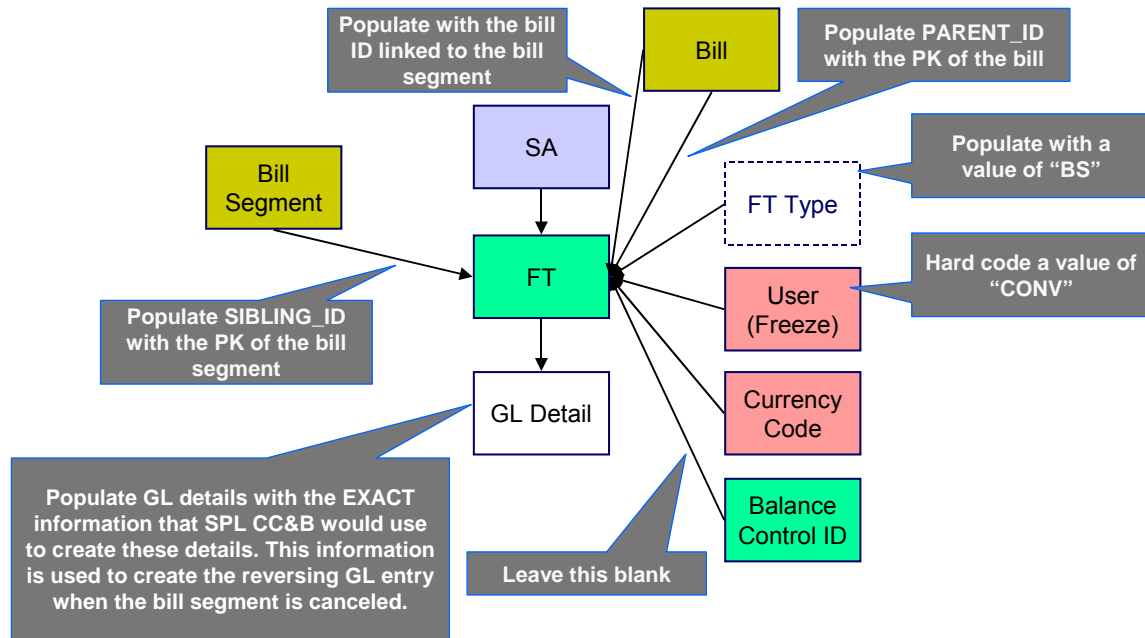
Bill - Read Details

The following data model illustrates the FK references on the read detail entity (a bill segment has one or more read details if the bill segment is associated with metered service).



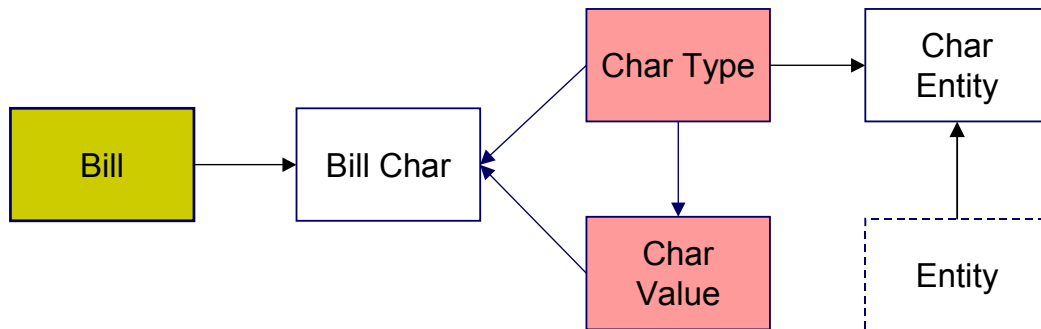
Bill - FT

The following data model illustrates the FT that must be associated with a bill segment.



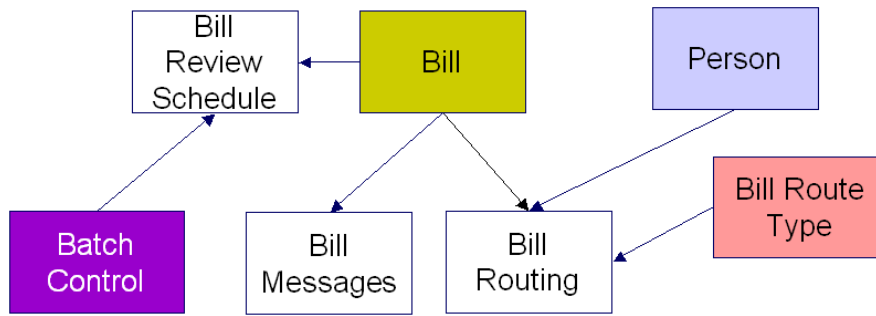
Bill Characteristics

The following data model illustrates Bill Characteristics.



Bill Messages, Bill Routing and Bill Review Schedule

The following data model illustrates Bill Messages, Bill Routing and Bill Review Schedule.



Bill Table Names

Data Model Name	Table Name	Generated Keys	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Bill	CI_BILL	Yes CI_BILL_K	CIPVBLLV	CIPVBILK Has dependencies	CIPVBLLI
SA Balance Snapshot	CI_BILL_SA	No. The key is bill ID plus SA ID.	CIPVBSAV		CIPVBSAI
Bill Segment	CI_BSEG	Yes CI_BSEG_K	CIPVSEGV	CIPVBSGK Has dependencies	CIPVSEGI
Calc Header	CI_BSEG_CALC	No. The key is bill segment ID and a sequence number	CIPVBSCV		CIPVBSCI
Calc Lines	CI_BSEG_CALC_LN	No. The key is bill segment ID, the header sequence number, and a sequence number	CIPVBSLV		CIPVBSLI
Read Detail	CI_BSEG_READ	No. The key is bill segment ID, SP ID and a sequence number	CIPVSRRV		CIPVSRRI
Item Detail	CI_BSEG_ITEM	No. The key is bill segment id and a sequence number	CIPVBSIV		CIPVBSII
Service Quantity	CI_BSEG_SQ	No. The key is bill segment ID, uom code, tou code and SQI code	CIPVSQTV		CIPVSQTI
FT (financial transaction)	CI_FT	Yes CI_FT_K	CIPVFTFV	CIPVFTXK Has	CIPVFTFI

				dependencies	
FT GL (FT general ledger)	CI_FT_GL	No. The key is FT ID and a GL sequence number	CIPVFTGV		CIPVFTGI
Characteristics	CI_BILL_CHAR	No. The key is bill ID, char type code and a sequence number	CIPVBCHV		CIPVBCHI
Bill Messages	CI_BILL_MSGS	No. The key is bill ID and bill message code.	CIPVBLMV		CIPVBLMI
Bill Routing	CI_BILL_ROUTING	No. The key is bill ID and a sequence number	CIPVBLRV		CIPVBLRI
Bill Review Schedule	CI_BILL_RVW_SC H	No. The key is bill ID, bill review date and batch code.	CIPVBRVV		CIPVBRVI

Bill Suggestions

Most companies have found it impossible to load bill segment item, bill calc header and lines with sufficient information and therefore these tables are not populated. See the comments in the above ERD's for more information.

Please populate the columns on the FT that's associated with the bill segment as follows:

- CUR_AMT should be set equal to the bill segment amount
- PAY_AMT should be set equal to the bill segment amount
- CRE_DTTM should be set equal to the bill segment end date / time
- FREEZE_SW should be "Y"
- FREEZE_DTTM should be set equal to the bill segment end date / time
- ARS_DT should be set equal to the bill segment end date
- CORRECTION_SW should be "N"
- REDUNDANT_SW should be "N"
- NEW_DEBIT_SW should be "N"
- NOT_IN_ARS_SW should be set to "N"
- SHOW_ON_BILL_SW should be set to "N"
- ACCOUNTING_DT should be set to the current date
- SCHED_DISTRIB_DT should be left blank
- CURRENCY_CD should be the currency on the installation record
- BAL_CTL_GRP_ID should be left blank
- XFERRED_OUT_SW should be set to "Y"

- PARENT_ID should be set to the bill ID
- SIBLING_ID should be set to the bill segment ID
- Do NOT create any GL details for the FT. If GL details are converted, ensure they are populated with the EXACT information SPL CC&B would use to create them. This information is used to create the reversing GL entry when the bill segment is canceled.

Payment

Contents

[Payment Data Model](#)
[Payment Table Names](#)
[Payment Suggestions](#)

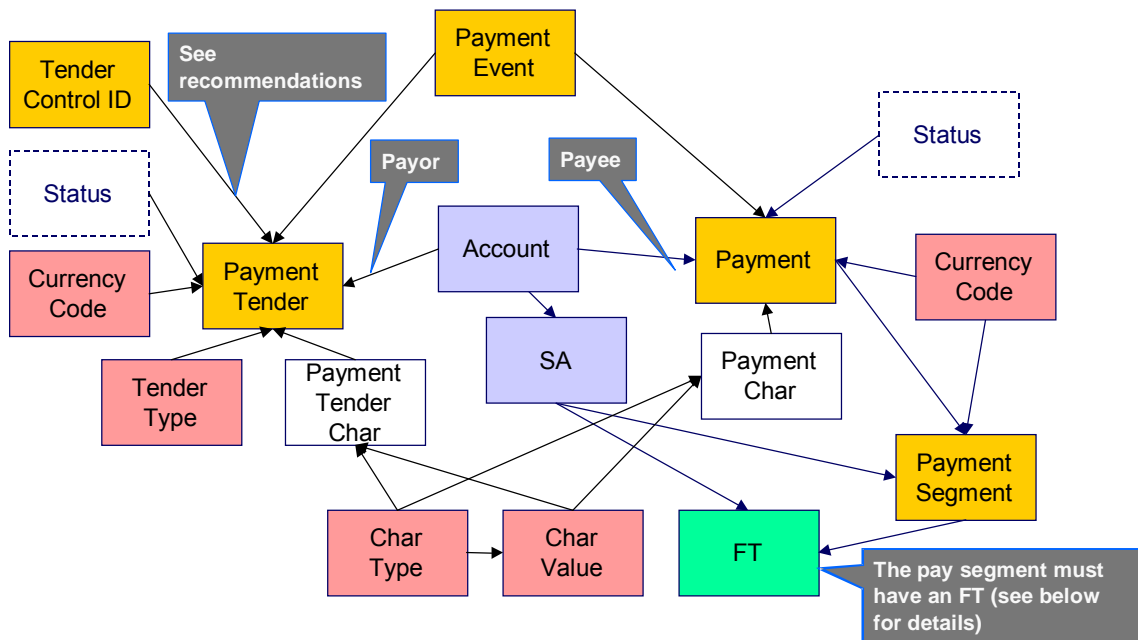
Payment Data Model

Contents

[Payment - Main](#)
[Payment - FT](#)

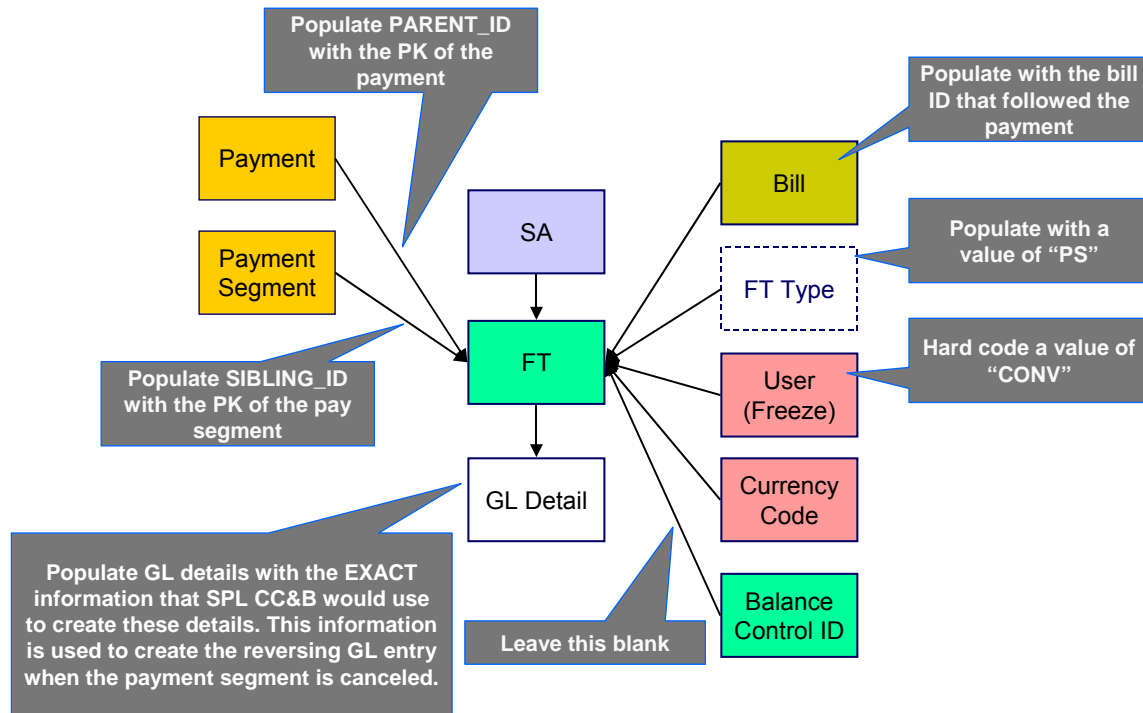
Payment - Main

The following data model illustrates the payment object.



Payment - FT

The following data model illustrates the FT that must be associated with a payment segment.



Payment Table Names

Data Model Name	Table Name	Generated Keys	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Payment	CI_PAY	Yes CI_PAY_K	CIPVPAYV	CIPVPAYK Has dependencies	CIPVPAYI
Payment Event	CI_PAY_EVENT	Yes CI_PAY_EVENT_K		CIPVPYEK Has dependencies	CIPVPYEI
Payment Event Characteristic	CI_PAY_EVT_CHARACTER	No. The key is PAY_EVENT_ID and a char type.	CIPVBLCV		CIPVBLCI
Payment Tender	CI_PAY_TNDR	Yes CI_PAY_TNDR_K	CIPVTNDV	CIPVTNDK Has dependencies	CIPVTNDI
Payment Segment	CI_PAY_SEG	Yes CI_PAY_SEG_K	CIPVPSGV	CIPVPSGK Has dependencies	CIPVPSGI
FT (financial transaction)	CI_FT	Yes CI_FT_K	CIPVFTFV	CIPVFTXK Has dependencies	CIPVFTFI

FT GL (FT general ledger)	CI_FT_GL	No. The key is FT id and a GL sequence number	CIPVFTGV		CIPVFTGI
Payment Tender Characteristic	CI_PAY_TNDR_CH AR	No. The key is PAY_TENDER_ID, plus a sequence number and a char type	CIPVTNCV		CIPVTNCI
Payment Characteristic	CI_PAY_CHAR	No. The key is PAY_ID, plus a sequence number and a char type	CIPVPYCV		CIPVPYCI

Payment Suggestions

We recommend that you use the system to create a single deposit control and link to it a single tender control using the PRODUCTION tables. The tender control should reference a tender source of "conversion". Use the prime key of the tender control as the foreign key on the tenders that you insert into the STAGING tables. This means you will have an invalid foreign key relationship on CI_PAY_TNDR (it will reference a tender control that doesn't exist).

After converting the payments:

- Re-access the tender control in production and enter the appropriate amounts (per tender type) to balance the tender control.
- Re-access the deposit control in production and enter the appropriate amounts to balance the deposit control.

Please populate the columns on the FT that's associated with the payment segment as follows:

- CUR_AMT should be set equal to the payment segment amount
- PAY_AMT should be set equal to the payment segment amount
- CRE_DTTM should be set equal to the payment segment date / time
- FREEZE_SW should be "Y"
- FREEZE_DTTM should be set equal to the payment segment date / time
- ARS_DT should be set equal to the payment segment date
- CORRECTION_SW should be "N"
- REDUNDANT_SW should be "N"
- NEW_DEBIT_SW should be "N"
- NOT_IN_ARS_SW should be set to "N"
- SHOW_ON_BILL_SW should be set to "N" on all payments other than payments that have been received since the last bill. For recent payments that you want to show on the next bill, this switch must be "Y"
- ACCOUNTING_DT should be set to the current date
- SCHED_DISTRIB_DT should be left blank

- CURRENCY_CD should be the currency on the installation record
- BAL_CTL_GRP_ID should be left blank
- XFERRED_OUT_SW should be set to "Y"
- PARENT_ID should be set to the payment ID
- SIBLING_ID should be set to the payment segment ID
- Do NOT create any GL details for the FT. If GL details are converted, ensure they are populated with the EXACT information SPL CC&B would use to create them. This information is used to create the reversing GL entry when the payment segment is canceled.

Adjustment

Contents

[Adjustment Data Model](#)
[Adjustment Table Names](#)
[Adjustment Suggestions](#)

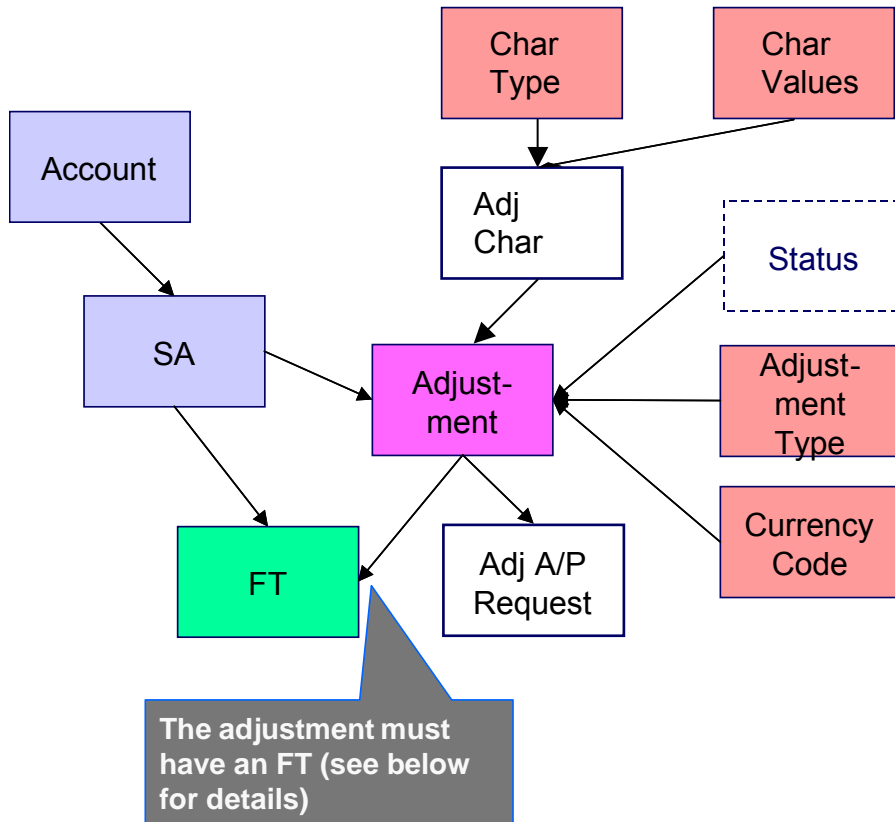
Adjustment Data Model

Contents

[Adjustment - Main](#)
[Adjustment - FT](#)

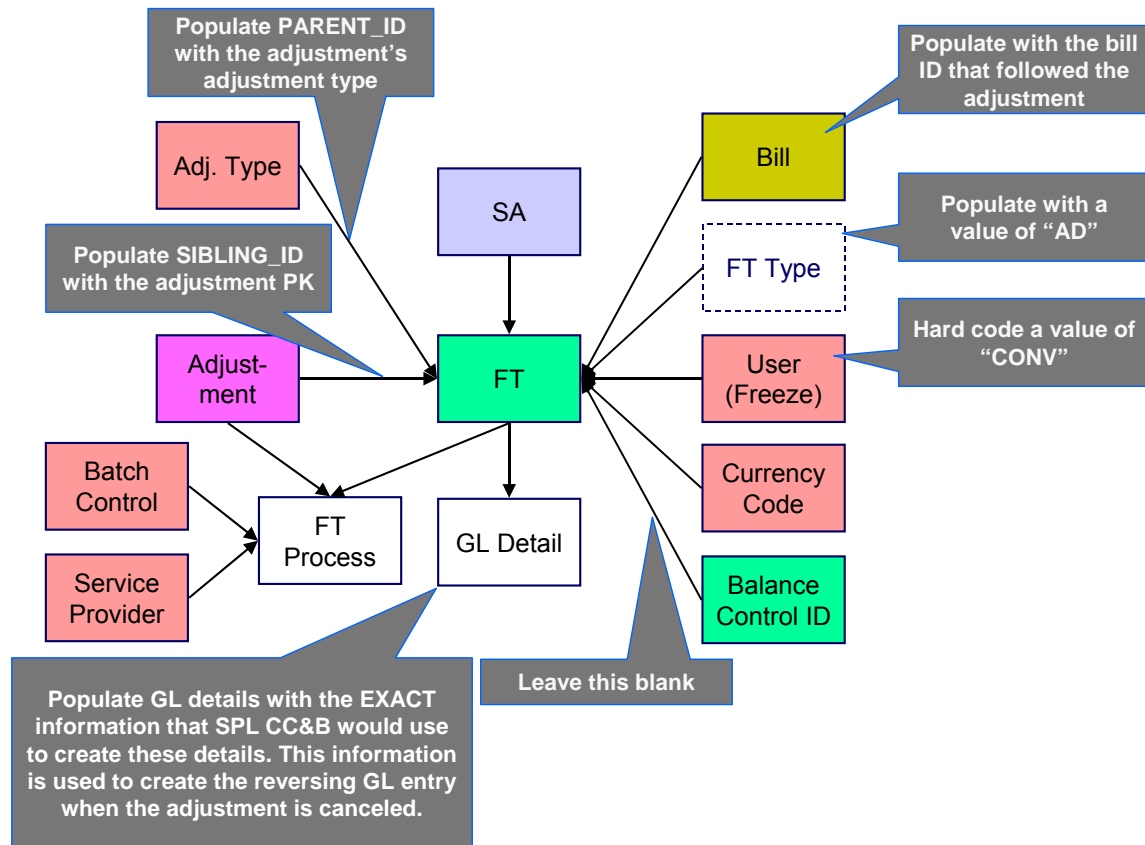
Adjustment - Main

The following data model illustrates the adjustment object.



Adjustment - FT

The following data model illustrates the FT that must be associated with an adjustment segment.



Adjustment Table Names

Data Model Name	Table Name	Generated Keys	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Adjustment	CI_ADJ	Yes CI_ADJ_K	CIPVADJV	CIPVADJK Has dependencies	CIPVADJI
Adjustment Characteristic	CI_ADJ_CHAR	No. The key is ADJ_ID and a char type.	CIPVADCV		CIPVADCI
Adjustment A/P Request	CI_ADJ_APREQ	Yes CI_ADJ_APREQ_K	CIPVAPRV	CIPVAPRK Has dependencies	CIPVAPRI
FT (financial transaction)	CI_FT	Yes CI_FT_K	CIPVFTFV	CIPVFTXK Has dependencies	CIPVFTFI
FT GL (FT general)	CI_FT_GL	No. The key is FT ID and a GL	CIPVFTGV		CIPVFTGI

ledger)		sequence number			
FT Process	CI_FT_PROC	No. The key is FT id and a sequence number	CIPVFTPV		CIPVFTPI

Adjustment Suggestions

Please populate the columns on the FT that's associated with the adjustment as follows:

- CUR_AMT should be set equal to the adjustment amount
- PAY_AMT should be set equal to the adjustment amount
- CRE_DTTM should be set equal to the adjustment date / time
- FREEZE_SW should be "Y"
- FREEZE_DTTM should be set equal to the adjustment date / time
- ARS_DT should be set equal to the adjustment date
- CORRECTION_SW should be "N"
- REDUNDANT_SW should be "N"
- NEW_DEBIT_SW should be "N"
- NOT_IN_ARS_SW should be set to "N"
- SHOW_ON_BILL_SW should be set to "N" on all adjustments other than adjustments that have been generated since the last bill. For recent adjustments that you want to show on the next bill, this switch must be "Y"
- ACCOUNTING_DT should be set to the current date
- SCHED_DISTRIB_DT should be left blank
- CURRENCY_CD should be the currency on the installation record
- BAL_CTL_GRP_ID should be left blank
- XFERRED_OUT_SW should be set to "Y"
- PARENT_ID should be set to the adjustment's adjustment type
- SIBLING_ID should be set to the adjustment ID
- Do NOT create any GL details for the FT. If GL details are converted, ensure they are populated with the EXACT information SPL CC&B would use to create them. This information is used to create the reversing GL entry when the adjustment is canceled.

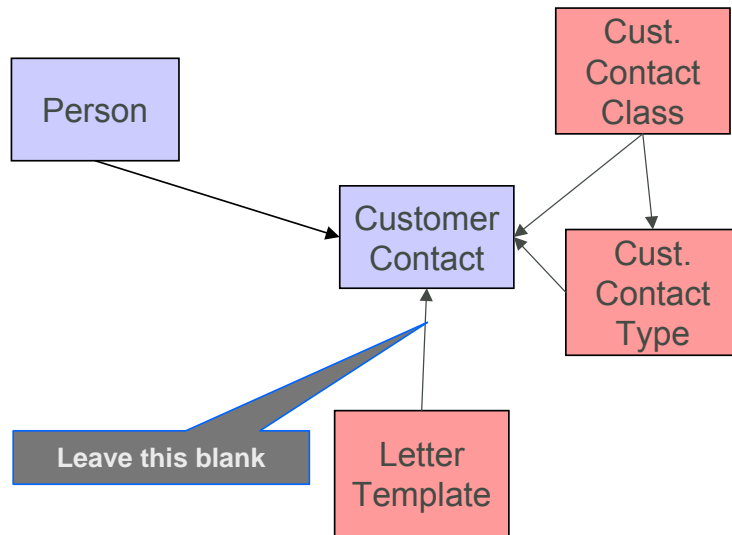
Customer Contact

Contents

- [Customer Contact Data Model](#)
- [Customer Contact Table Names](#)
- [Customer Contact Suggestions](#)

Customer Contact Data Model

The following data model illustrates the Customer Contact object.



Customer Contact Table Names

Data Model Name	Table Name	Generated Keys	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Customer Contact	CI_CC	Yes CI_CC_K	CIPVCSCV	CIPVCCTK Has dependencies	CIPVCSCI

Customer Contact Suggestions

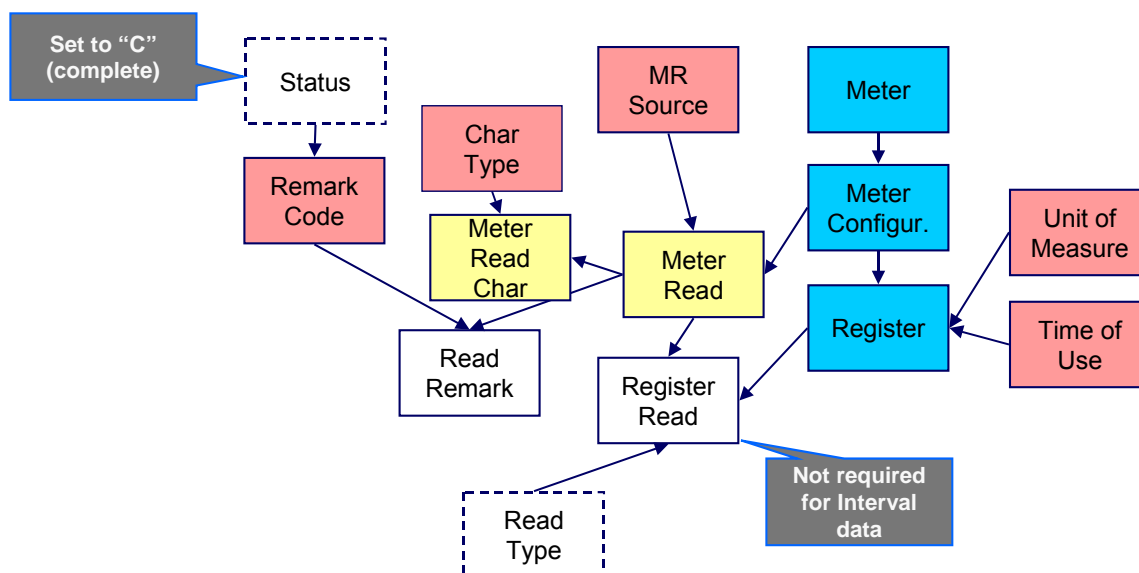
N/A

Meter Read**Contents**

- [Meter Read Data Model](#)
- [Meter Read Table Names](#)
- [Meter Read Suggestions](#)

Meter Read Data Model

The following data model illustrates the meter read object.



Meter Read Table Names

Data Model Name	Table Name	Generated Keys	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Meter Read	CI_MR	Yes CI_MR_K	CIPVMRDV	CIPVMRDK Has dependencies	CIPVMRDI
Register Read	CI_REG_READ	Yes CI_REG_READ_K	CIPVRGRV	CIPVRRDK Has dependencies	CIPVRGRI
Read Remark	CI_MR_REM	No. Key is meter read plus read ID.	CIPVMRMV		CIPVMRMI
Meter Read Characteristics	CI_MR_CHAR	No. The key is MR_ID plus a sequence number and a char type.	CIPVMRCV		CIPVMRCI

Meter Read Suggestions

N/A

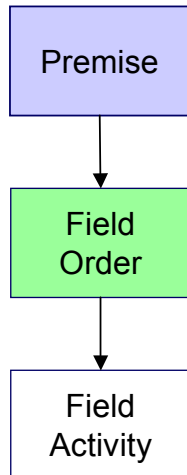
Field Order

Contents

[Field Order Data Model](#)
[Field Order Table Names](#)
[Field Order Suggestions](#)

Field Order Data Model

The following data model illustrates the Field Order object.



Field Order Table Names

Data Model Name	Table Name	Generated Keys	Object Validation Batch Control	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Field Order	CI_FO	Yes CI_FO_K	VAL-FO	CIPVFORV	CIPVFORK Has dependencies	CIPVFORI

Field Order Suggestions

N/A

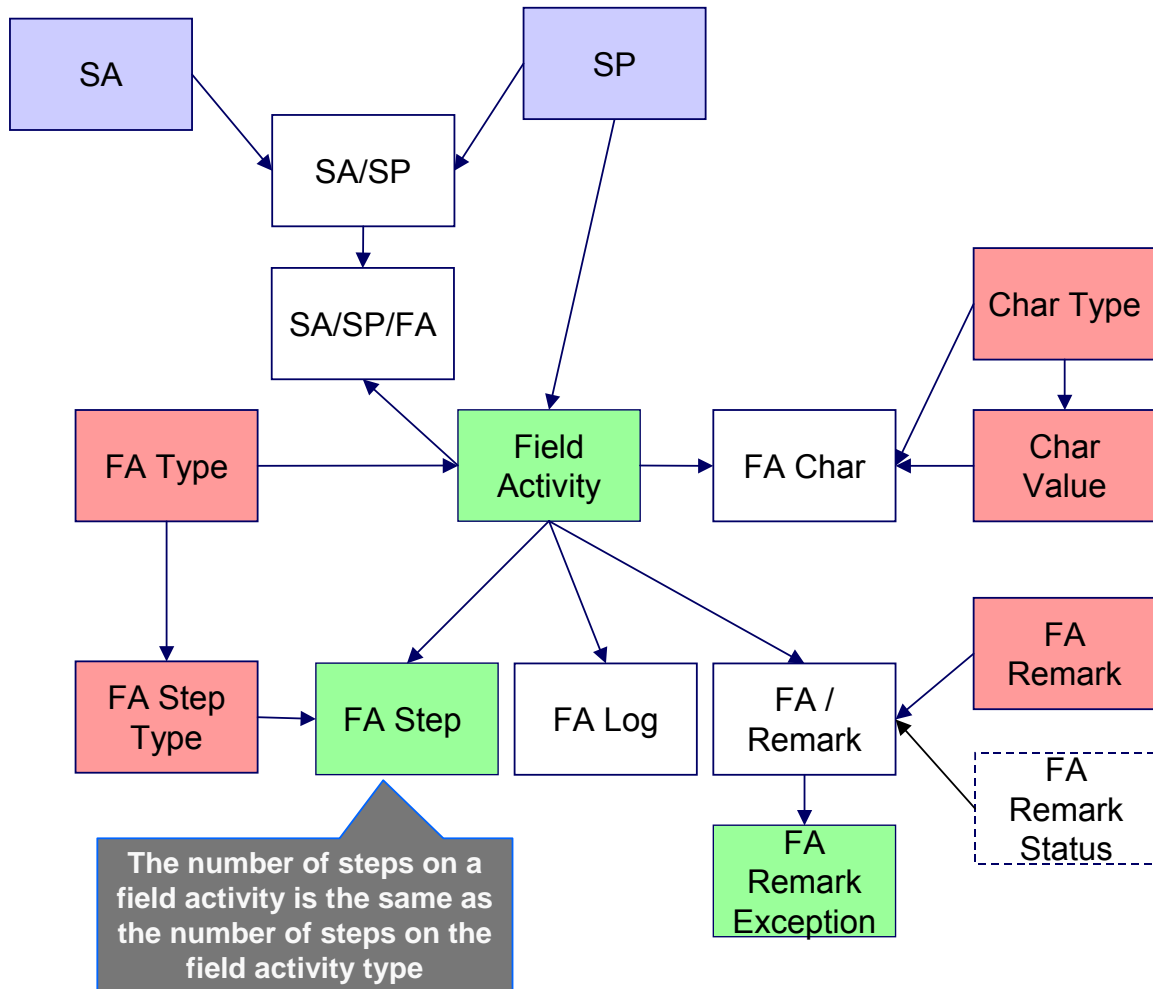
Field Activity

Contents

[Field Activity Data Model](#)
[Field Activity Table Names](#)
[Field Activity Suggestions](#)

Field Activity Data Model

The following data model illustrates the field activity object.



Field Activity Table Names

Data Model Name	Table Name	Generated Keys	Object Validation Batch Control	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Field Order	CI_FO	Yes CI_FO_K	VAL-FO	CIPVFORV	CIPVFORK Has dependencies	CIPVFORI
Field Activity	CI_FA	Yes CI_FA_K	VAL-FA		CIPVFACK Has dependencies	CIPVFACI
Step	CI_FA_STEP	No. The key is FA_ID plus sequence number.		CIPVFSTV		CIPVFSTI
Characteristic	CI_FA_CHAR	No. The key is FA_ID plus		CIPVFAHV		CIPVFAHI

		CHAR_TYPE_CD plus sequence number.				
Remarks	CI_FA_REM	No. The key is FA_ID plus FA_REM_CD.		CIPVFARV		CIPVFARI
Log	CI_FA_LOG	No. The key is FA_ID plus sequence number.		CIPVFALV		CIPVFALI
SA/SP/FA	CI_SA_SP_FA	No. The key is SA/SP id and FA id.		CIPVSSFV		CIPVSSFI

Field Activity Suggestions

N/A

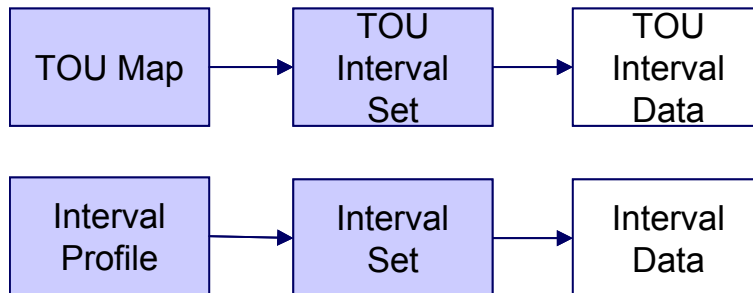
Interval Data

Contents

[Interval Data Data Model](#)
[Interval Data Table Names](#)
[Interval Data Suggestions](#)

Interval Data Data Model

The following data model illustrates the SA Interval Billing object.



Interval Data Table Names

Data Model Name	Table Name	Generated Keys	Object Validation Batch Control	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
TOU Interval Set	CI_TOU_DATA_SET	Yes CI_TOU_DATA_SET_K	VAL-TDS		CIPVTDSK Has dependencies	CIPVTDSI
TOU Interval	CI_TOU_DATA	No. The key is		CIPVTOFV		CIPVTODI

Data		TOU_DATA_SET_ID plus TOU_DATA_DTTM				
Interval Set	CI_INTV_DATA_SET	Yes CI_INTV_DATA_SET_K	VAL-IDS		CIPVIDSK Has dependencies	CIPVIDSI
Interval Data	CI_INTV_DATA	No. The key is INTV_DATA_SET_ID plus INTV_DATA_DTTM		CIPVITFV		CIPVITDI

Interval Data Suggestions

N/A

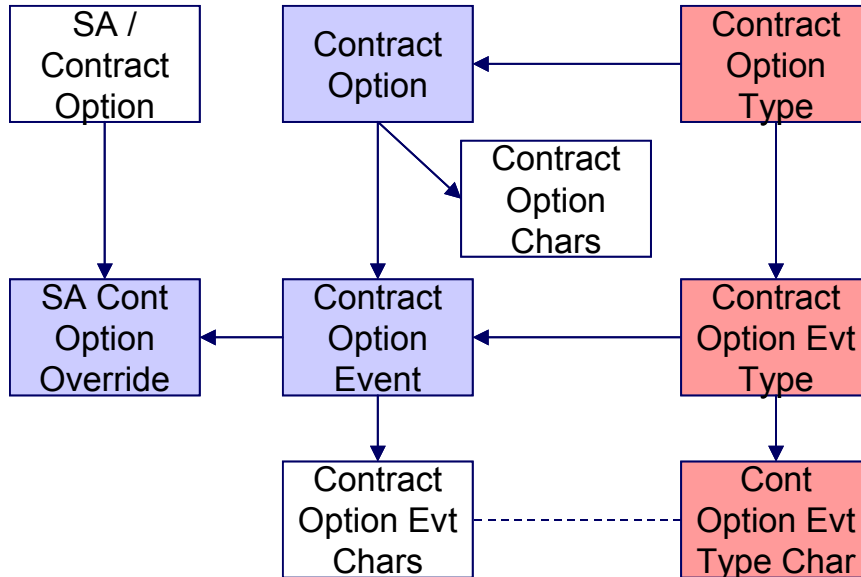
Contract Option Events

Contents

- [Contract Option Events Data Model](#)
- [Contract Option Events Table Names](#)
- [Contract Option Events Suggestions](#)

Contract Option Events Data Model

The following data model illustrates the Contract Options objects.



Contract Option Events Table Names

Data Model Name	Table Name	Generated Keys	Object Validation Batch Control	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control

Contract Option Event	CI_COP_EVT	Yes CI_COP_EVT_K	VAL-CEVT		CIPVCEVK Has dependencies	CIPVCEVI
Contract Option Event Characteristics	CI_COP_EVT_CHAR	No. The key is CONT_OPT_EVT_ID plus CHAR_TYPE_CD		CIPVCVCV		CIPVCVCI
SA Contract Option Override	CI_SA_COP_OVRD	No. The key is SA_CONT_OPT_ID plus CONT_OPT_EVT_ID plus OVRD_DTTM		CIPVSCOV		CIPVSCOI

Contract Option Events Suggestions

N/A

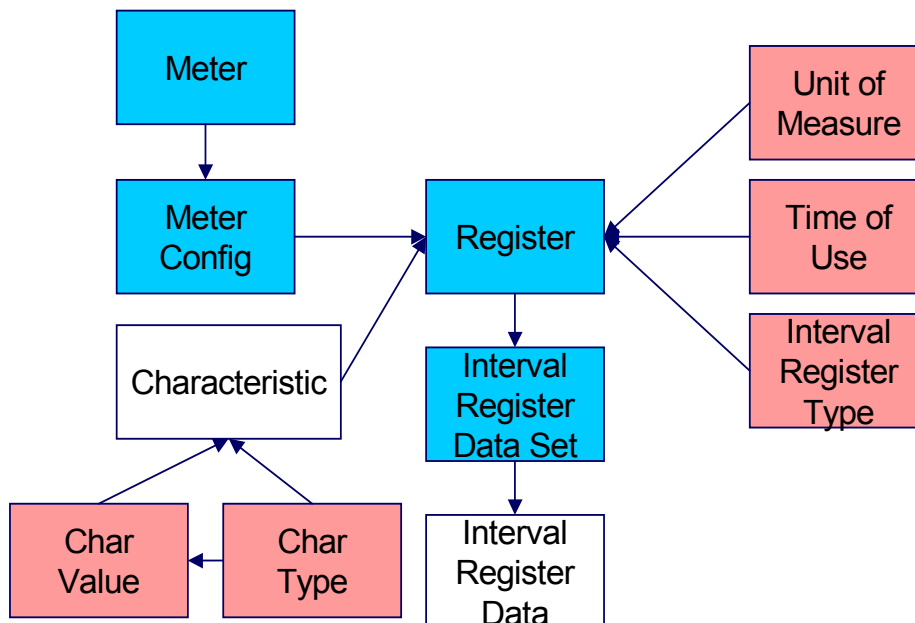
Register Interval Data

Contents

- [Register Interval Data Model](#)
- [Register Interval Data Table Names](#)
- [Register Interval Data Suggestions](#)

Register Interval Data Model

The following data model illustrates the register interval data object.



Register Interval Data Table Names

Data	Table Name	Generated	Object	Referential	Key	Insertion
------	------------	-----------	--------	-------------	-----	-----------

Model Name		Keys	Validation Batch Control	Integrity Validation Batch Control	Assignment Batch Control	Batch Control
Interval Register Data Set	CI_REG_DATA_SE I	Yes CI_REG_DATA_S ET_K	VAL-IRDS		CIPVIRSK Has dependencies	CIPVIRSI
Interval Register Data	CI_REG_DATA	No. The key is REG_DATA_SET_ ID plus REG_DATA_DTT M		CIPVREFV		CIPVREDI

Register Interval Data Suggestions

N/A

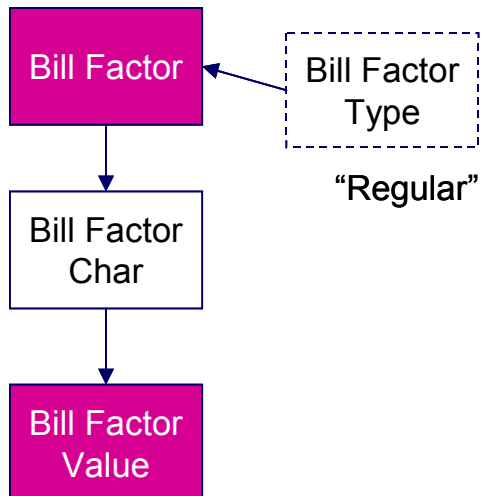
Bill Factor Value

Contents

- [Bill Factor Value Data Model](#)
- [Bill Factor Value Table Names](#)
- [Bill Factor Value Suggestions](#)

Bill Factor Value Data Model

The following data model illustrates the bill factor objects.



Bill Factor Value Table Names

Data Model Name	Table Name	Generated Keys	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Bill Factor	CI_BF_VAL	No. The key is	CIPVBFVV		CIPVBFVI

Value		BF_CD plus CHAR_TYPE_CD plus CHAR_VAL plus TOU_GRP_CD plus EFFDT			(Not threadable)
-------	--	---	--	--	---------------------

Bill Factor Value Suggestions

N/A

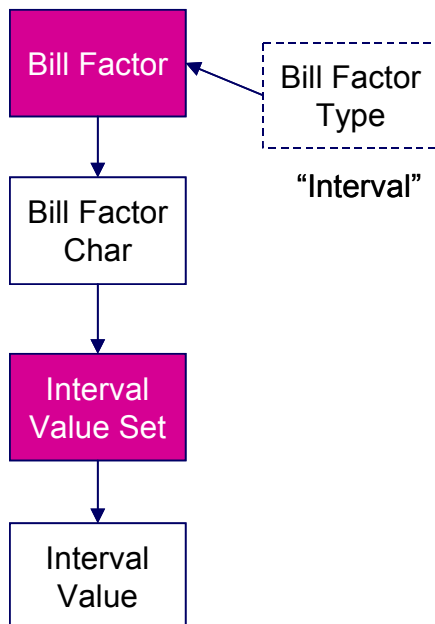
Bill Factor Interval Prices

Contents

- [Bill Factor Interval Prices Data Model](#)
- [Bill Factor Interval Prices Table Names](#)
- [Bill Factor Interval Prices Suggestions](#)

Bill Factor Interval Prices Data Model

The following data model illustrates the bill factor interval prices objects.



Bill Factor Interval Prices Table Names

Data Model Name	Table Name	Generated Keys	Object Validation Batch Control	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Interval Value Set	CI_INTV_VAL_SE I	Yes CI_INTV_VAL_SE I_K	VAL-IVS		CIPVIVSK	CIPVIVSI

Interval Value	CI_INTV_VAL	No. The key is INTV_VAL_SET_I D plus INTV_VAL_DTTM		CIPVITFV		CIPVITVI
----------------	-----------------------------	---	--	----------	--	----------

Bill Factor Interval Prices Suggestions

N/A

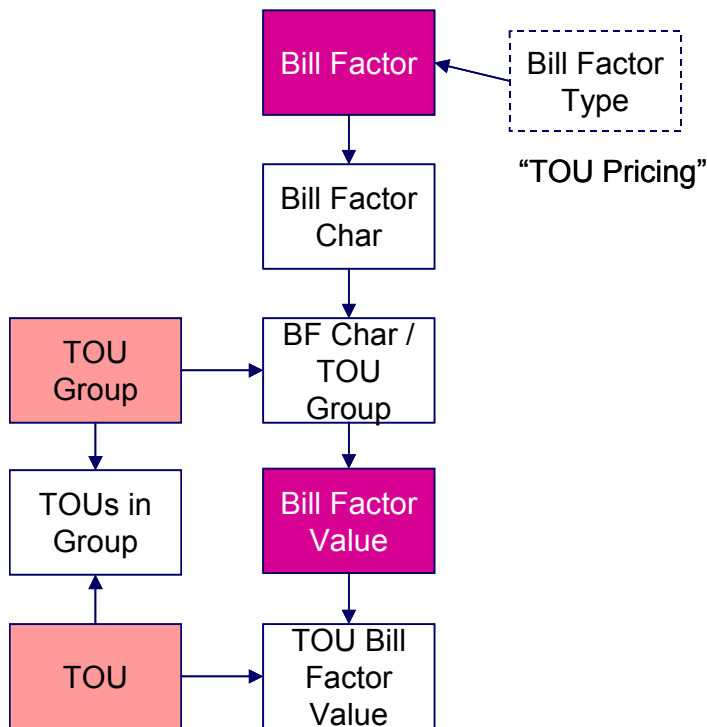
Bill Factor TOU Pricing

Contents

- [Bill Factor TOU Pricing Data Model](#)
- [Bill Factor TOU Pricing Table Names](#)
- [Bill Factor TOU Pricing Suggestions](#)

Bill Factor TOU Pricing Data Model

The following data model illustrates the bill factor TOU Pricing objects.



Bill Factor TOU Pricing Table Names

Data Model Name	Table Name	Generated Keys	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
TOU Bill Factor Value	CI_TOU_BF_VAL	No. The key is BF_CD plus CHAR_TYPE_CD	CIPVTBVV		CIPVTBVI (Not

		plus CHAR_VAL plus EFFDT plus TOU_GRP_CD plus TOU_CD			threadable)
--	--	---	--	--	-------------

Bill Factor TOU Pricing Suggestions

N/A

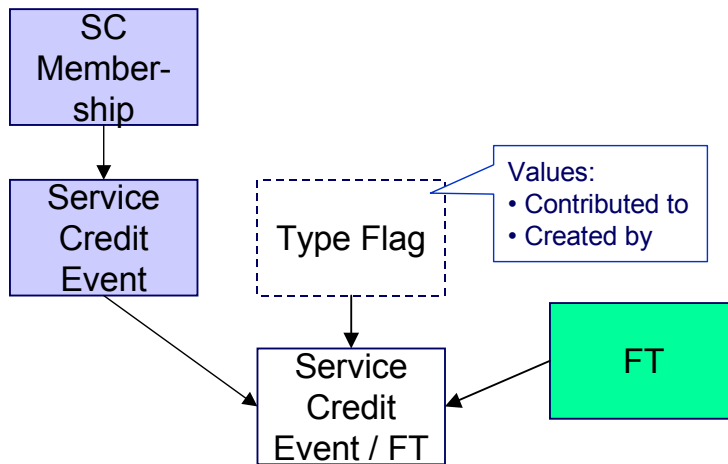
Service Credit Event

Contents

- [Service Credit Event Data Model](#)
- [Service Credit Event Table Names](#)
- [Service Credit Event Suggestions](#)

Service Credit Event Data Model

The following data model illustrates the Service Credit Event objects.



Service Credit Event Table Names

Data Model Name	Table Name	Generated Keys	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Service Credit Event	CI_SC_EVT	Yes CI_SC_EVT_K	CIPVSCVV	CIPVSCVK Has dependencies	CIPVSCVI
Service Credit Event / FT	CI_SC_EVT_FT	Yes CI_SC_EVT_FT_K	CIPVSCFV	CIPVSCFK Has dependencies	CIPVSCFI

Service Credit Event Suggestions

Loading the Service Credit Event FT data is optional. This data need only be converted if it exists in the legacy system and it is deemed necessary to include it.

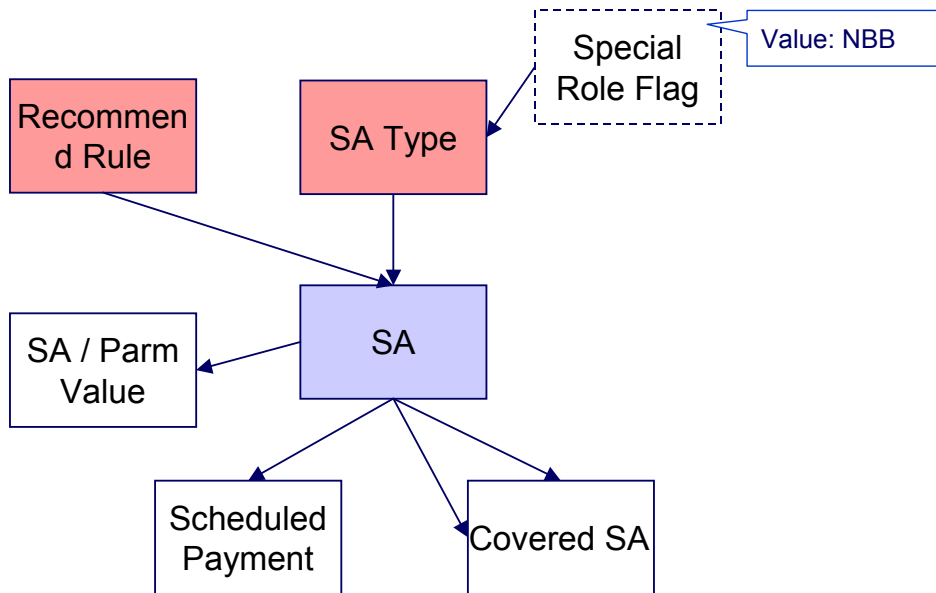
Non-Billed Budgets

Contents

- [Non-Billed Budgets Data Model](#)
- [Non-Billed Budgets Table Names](#)
- [Non-Billed Budgets Suggestions](#)

Non-Billed Budgets Data Model

The following data model illustrates the Non-Billed Budgets objects.



Non-Billed Budgets Table Names

Data Model Name	Table Name	Generated Keys	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
NBB / Service Agreement	CI_NB_SA	No. The key is SA_ID plus CVRD_SA_ID	CIPVNBSV		CIPVNBSI
NBB Scheduled Payments	CI_NB_SCHED_PAY	Yes CI_NB_SCHED_PAY_K	CIPVNSPV	CIPVNSPK Has dependencies	CIPVNSPI
NBB / SA Parameters	CI_SA_NB_PARM	No. The key is SA_ID plus Sequence Number.	CIPVNPMV		CIPVNPMI

Non-Billed Budgets Suggestions

N/A

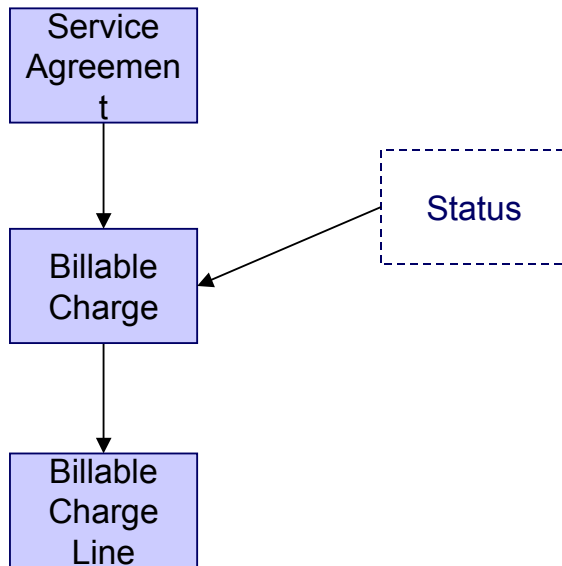
Billable Charge

Contents

- [Billable Charge Data Model](#)
- [Billable Charge Table Names](#)
- [Billable Charge Suggestions](#)

Billable Charge Data Model

The following data model illustrates the Billable Charge objects.



Billable Charge Table Names

Data Model Name	Table Name	Generated Keys	Object Validation Batch Control	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Billable Charge	CI_BILL_CHG	Yes. CI_BILL_CHG_K	VAL-BCHG	CIPVBCGV	CIPVBCGK	CIPVBCGI
Billable Charge Line	CI_B_CHG_LINE	No. The key is billable charge id and a sequence number		CIPVBCLV		CIPVBCLI

Billable Charge Suggestions

N/A

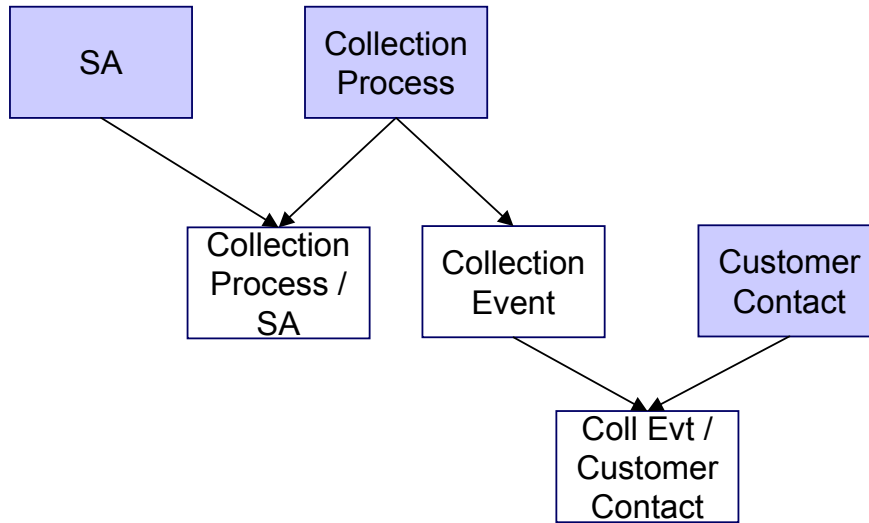
Collection Process

Contents

- [Collection Process Data Model](#)
- [Collection Process Table Names](#)
- [Collection Process Suggestions](#)

Collection Process Data Model

The following data model illustrates the Collection Process objects.



Collection Process Table Names

Data Model Name	Table Name	Generated Keys	Object Validation Batch Control	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Collection Process	CI_COLL_PROC	Yes. CI_COLL_PROC_K	VAL-COLL	CIPVCLPV	CIPVCLPK	CIPVCLPI
Collection Event	CI_COLL_EVT	No. The key is collection process id and a sequence number.		CIPVCVNV		CIPVCVNI
Collection Process / Service Agreement	CI_COLL_PROC_SA	No. The key is collection process id and service agreement id.		CIPVCLSV		CIPVCLSI
Collection Event Customer Contact	CI_COLL_EVT_CC	No. The key is collection process id, a sequence number and the customer contact id.		CIPVCECV		CIPVCECI

Collection Process Suggestions

N/A

Severance Process

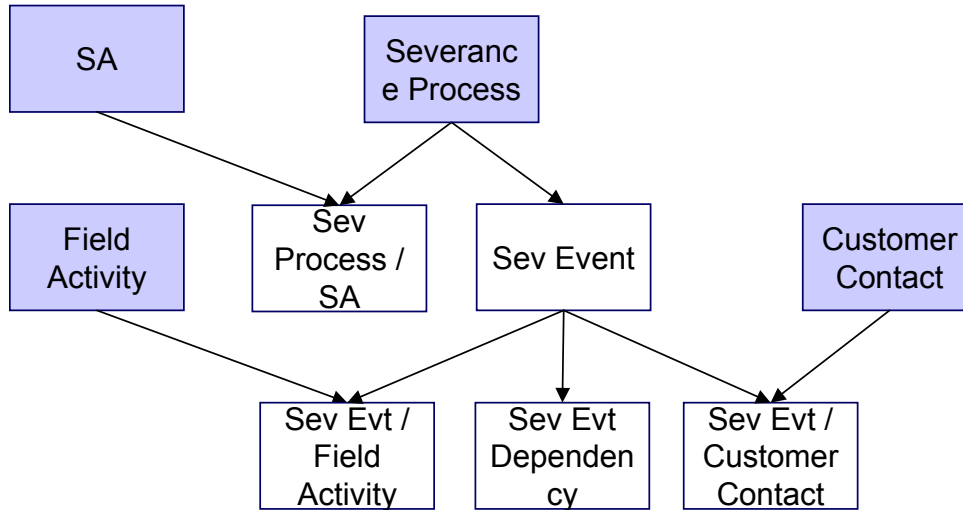
Contents

[Severance Process Data Model](#)

Severance Process Table Names Severance Process Suggestions

Severance Process Data Model

The following data model illustrates the Severance Process objects.



Severance Process Table Names

Data Model Name	Table Name	Generated Keys	Object Validation Batch Control	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Severance Process	CI_SEV_PROC	Yes. CI_SEV_PROC_K	VAL-SEVP	CIPVSEPV	CIPVSEPK	CIPVSEPI
Severance Event	CI_SEV_EVT	No. The key is severance process id and a sequence number.		CIPVSEVV		CIPVSEVI
Severance Event Customer Contact	CI_SEV_EVT_CC	No. The key is severance process id, a sequence number and the customer contact id.		CIPVSECV		CIPVSECI
Severance Event / Field Activity	CI_SEV_EVT_FA	No. The key is severance process id, a sequence number and field activity id.		CIPVSEFV		CIPVSEFI
Severance Event Dependency	CI_SEV_EVT_DEP	No. The key is severance process id, event sequence number and a sequence number.		CIPVSEDV		CIPVSEDI

Severance Process Suggestions

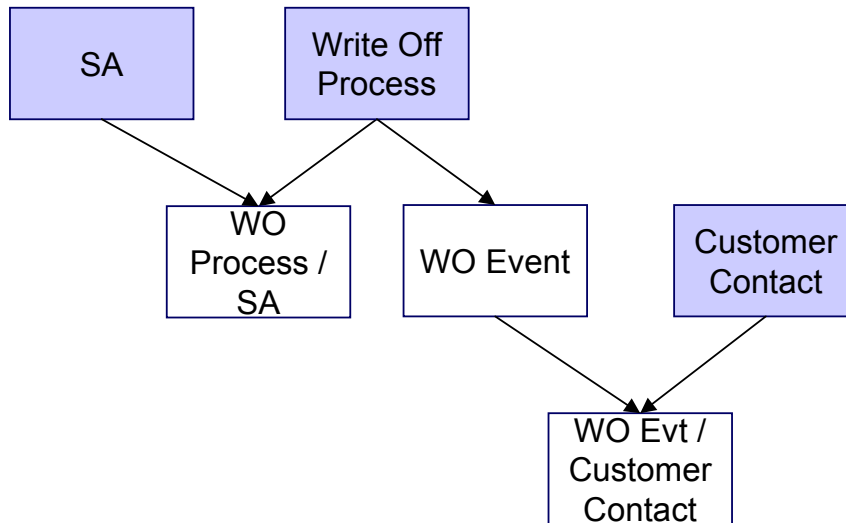
N/A

Write Off Process**Contents**

- [Write Off Process Data Model](#)
- [Write Off Process Table Names](#)
- [Write Off Process Suggestions](#)

Write Off Process Data Model

The following data model illustrates the Write Off Process objects.

**Write Off Process Table Names**

Data Model Name	Table Name	Generated Keys	Object Validation Batch Control	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Write Off Process	CI_WO_PROC	Yes. CI_WO_PROC_K	VAL-WOP	CIPVWOPV	CIPVWOPK	CIPVWOPI
Write Off Process / Service Agreement	CI_WO_PROC_SA	No. The key is write-off process id and service agreement id.		CIPVWOSV		CIPVWOSI
Write Off Event	CI_WO_EVT	No. The key is write-off process id and a sequence number.		CIPVWOVV		CIPVWOVI
Write Off Event / Customer Contact	CI_WO_EVT_CC	No. The key is write-off process id, a sequence number and the customer		CIPVWOCV		CIPVWOCI

		contact id.				
--	--	-------------	--	--	--	--

Write Off Process Suggestions

N/A

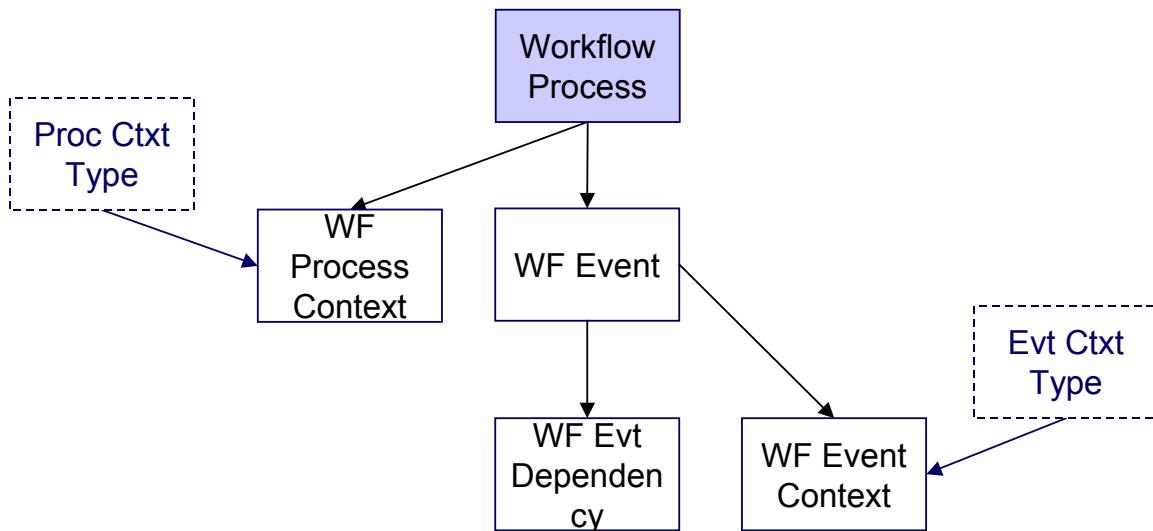
Workflow Process

Contents

- [Workflow Process Data Model](#)
- [Workflow Process Table Names](#)
- [Workflow Process Suggestions](#)

Workflow Process Data Model

The following data model illustrates the Workflow Process objects.



Workflow Process Table Names

Data Model Name	Table Name	Generated Keys	Object Validation Batch Control	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Workflow Process	CI_WF_PROC	Yes. CI_WF_PROC_K	VAL-WFP	CIPVWPRV	CIPVWPRK	CIPVWPRI
Workflow Process Context	CI_WF_PROC_CTX_I	No. The key is workflow process id, workflow process context type and value.		CIPVWPCV		CIPVWPCI
Workflow Event	CI_WF_EVT	No. The key is workflow process id and a sequence number.		CIPVWEVV		CIPVWEVI

Workflow Event Context	CI_WF_EVT_CTX	No. The key is workflow process id, event sequence number, event context type and value.		CIPVWECV		CIPVWECI
Workflow Event Dependency	CI_WF_EVT_DEP	No. The key is workflow process id, event sequence number and a sequence number.		CIPVWEDV		CIPVWEDI

Workflow Process Suggestions

N/A

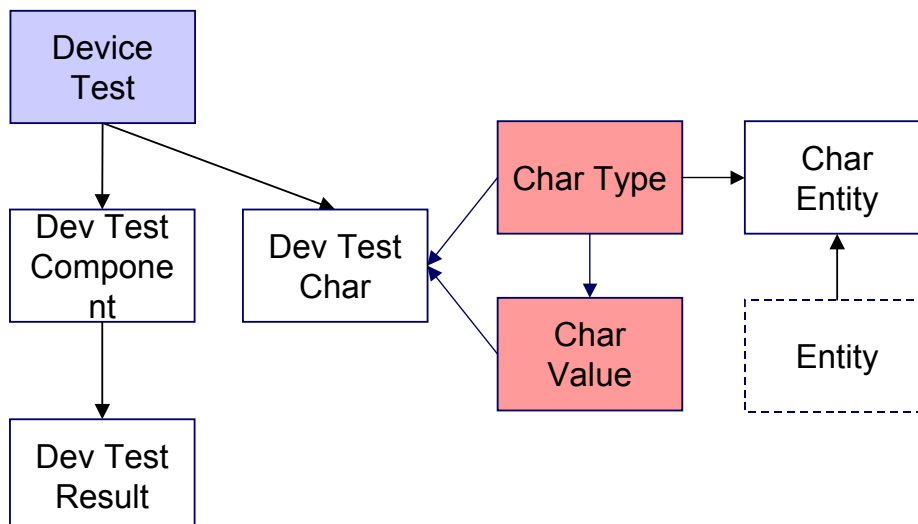
Device Test

Contents

- [Device Test Data Model](#)
- [Device Test Table Names](#)
- [Device Test Suggestions](#)

Device Test Data Model

The following data model illustrates the Device Test object.



Device Test Table Names

Data Model Name	Table Name	Generated Keys	Object Validation Batch Control	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Device Test	CI_DV_TEST	Yes. CI_DV_TEST_K	VAL-DTST	CIPVDTTV	CIPVDTTK	CIPVDTTI

Device Test Characteristics	CI_DV_TEST_CHAR	No. The key is device test id and characteristic type code		CIPVDTCV		CIPVDTCI
Device Test Component	CI_DV_TEST_COMPONENT	No. The key is device test id and a sequence number.		CIPVDTMV		CIPVDTMI
Device Test Result	CI_DV_TEST_RES	No. The key is device test id, sequence number, test component type code and a sequence number		CIPVDTRV		CIPVDTRI

Device Test Suggestions

N/A

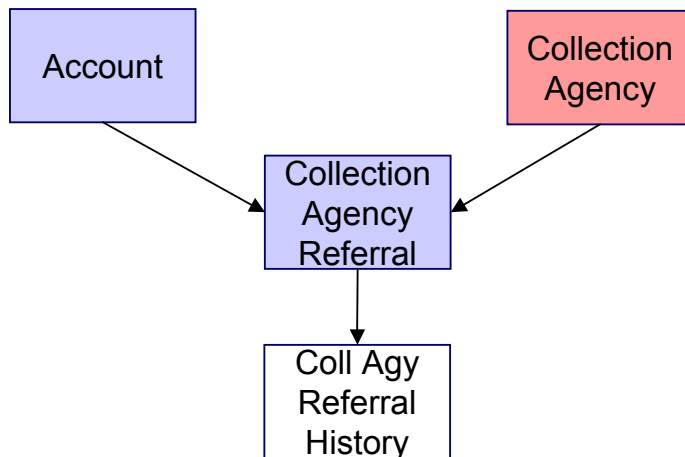
Collection Agency Referral

Contents

- [Collection Agency Referral Data Model](#)
- [Collection Agency Referral Table Names](#)
- [Collection Agency Referral Suggestions](#)

Collection Agency Referral Data Model

The following data model illustrates the Collection Agency Referral object.



Collection Agency Referral Table Names

Data Model Name	Table Name	Generated Keys	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Collection	CI_COLL_AGY_REF	Yes.	CIPVCARV	CIPVCARK	CIPVCARI

Agency Referral		CI_COLL_AGY_REF_K			
Collection Agency Referral History	CI_COLL_AGY_HIS	No. The key is collection agency referral id and characteristic type code	CIPVARHV		CIPVARHI

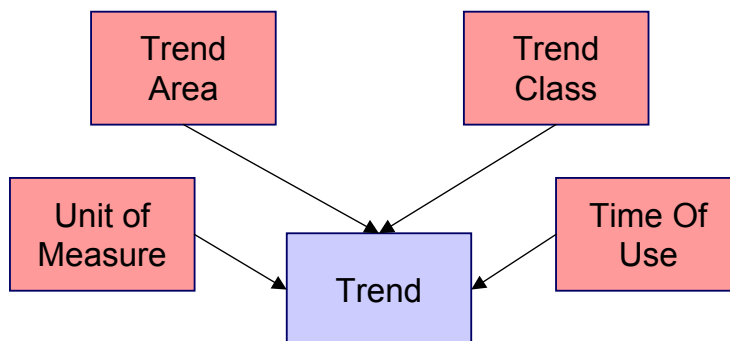
Collection Agency Referral Suggestions

N/A

Trend**Contents**[Trend Data Model](#)[Trend Table Name](#)[Trend Suggestions](#)

Trend Data Model

The following data model illustrates the Trend object.



Trend Table Name

Data Model Name	Table Name	Generated Keys	Referential Integrity Validation Batch Control	Key Assignment Batch Control	Insertion Batch Control
Trend	CI_TREND	No. The key is batch run number, trend area code, trend class code, unit of measure code, time of use code and read date.	CIPVTRNV		CIPVTRNI

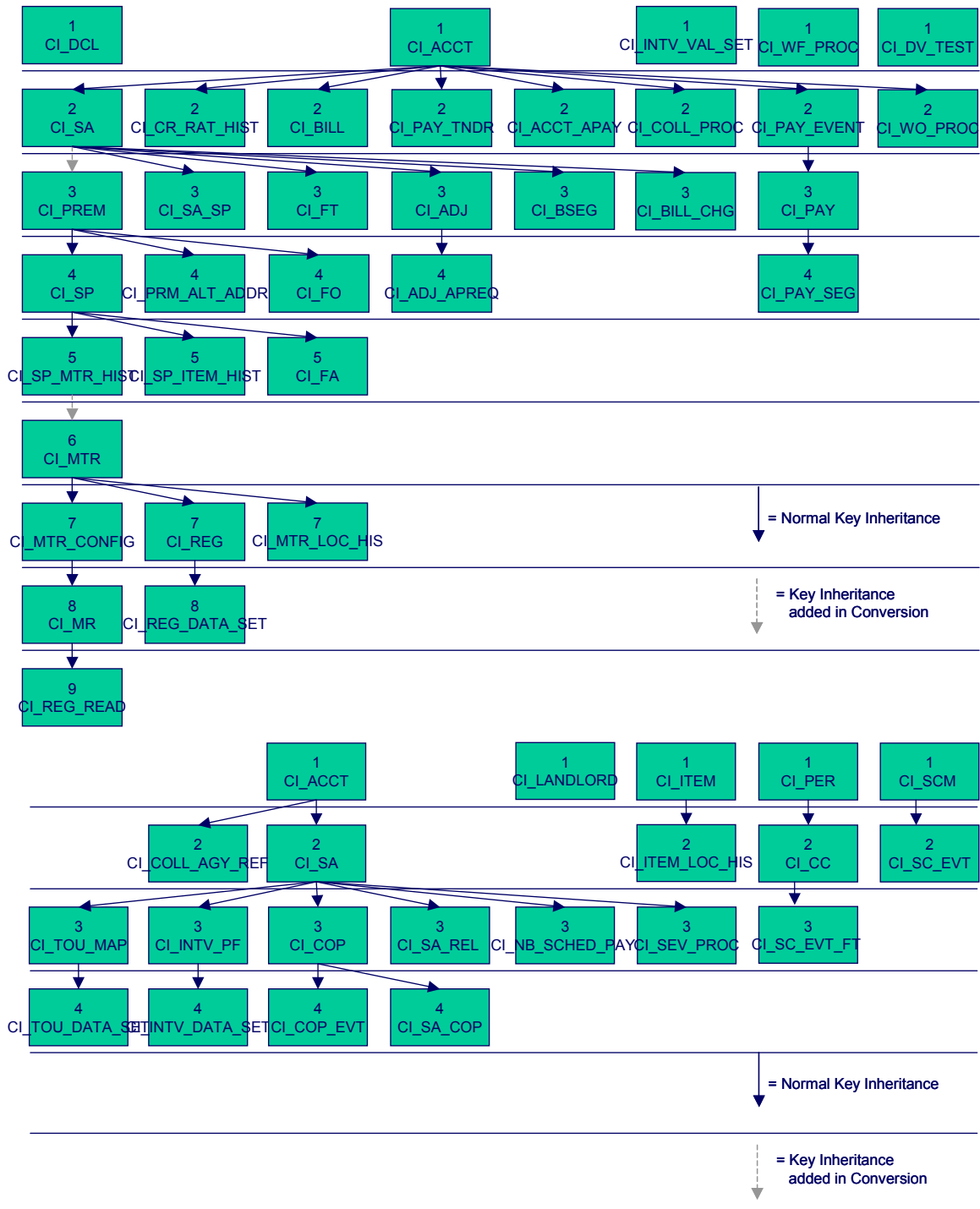
Trend Suggestions

N/A

Program Dependencies

The programs used to assign production keys are listed under [Master Data](#) and [Transaction Data](#) (in the Table Names matrices). Most of these programs have no dependencies (i.e., they can be executed in any order you please). The only exceptions to this statement are illustrated in the following diagram.

The tiers in this diagram contain a box for each table whose key assignment program is dependent on the successful execution of other key assignment programs. The numbers that appear in the boxes describe the order in which these programs must be executed.

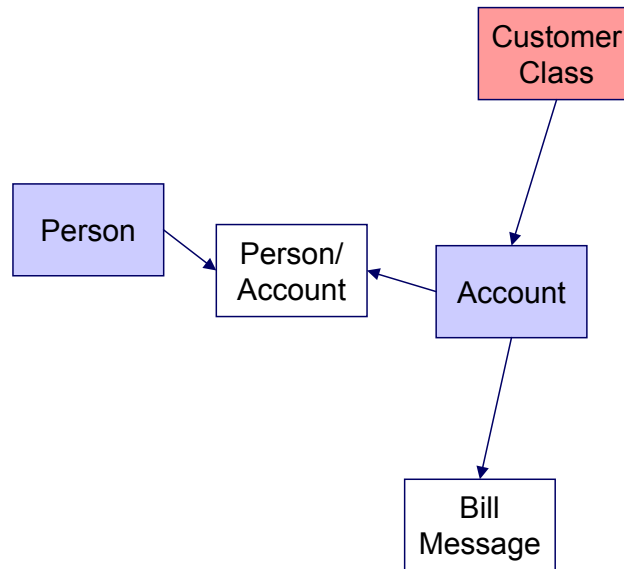


Please refer to the various "Table Names" sections above for the respective names of the programs to allocate each table's keys.

Take note! Prior to running the key generation program for a particular object, it is required that any previously generated keys be cleared from the key allocation tables and the key allocation temporary storage table. It is recommended that the key allocation tables be analyzed between runs to maximize performance.

Appendix A - Entity Relationship Diagramming Standards

Because all data is stored in relational table, you need to be able to read diagrams that illustrate relationships between the various tables. The following entity diagram uses every diagramming notation used in the documentation:



Contents

- Entity
- Color Coding
- Relationships

Entity

Every box on the above diagram represents an entity (i.e., a table). An entity may be a physical entity, such as a Person, or a logical construct, such as an Account.













Color Coding

If you can view this document in color, you will notice that each entity is colored. The color indicates the “subsystem” which governs the entity. Know the governing subsystem is important because:

- The system’s menu structure is subsystem-oriented (i.e., if you know the subsystem, you will know how to use the menus to navigate to the page used to view and update the entity).
- The system’s documentation is subsystem-oriented (i.e., if you know the subsystem, you will know which chapter contains information about the entity).

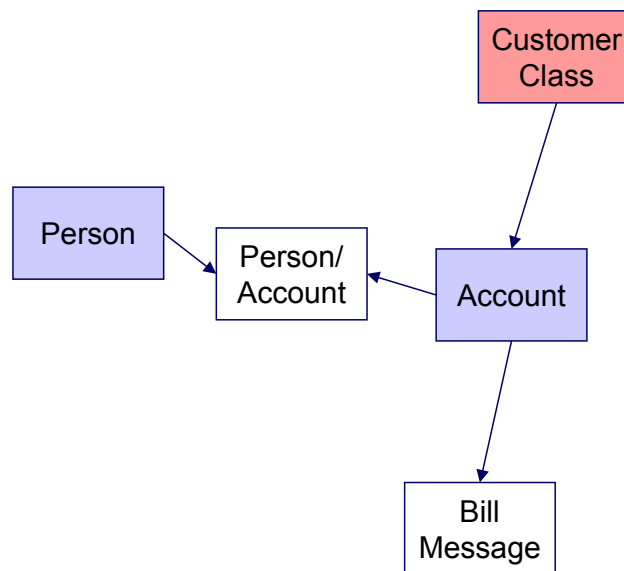
Some entities are not color-coded (i.e., they are white). These entities do not have a dedicated page, as they are part of a parent entity. For example, the Person / Account entity above is related to the Account object and does not have its own page. You must display the parent entity in order to view such an entity. For example, if you want to look at Person / Account information, you must go to the Account page.

The following table describes the colors utilized in the documentation:

Color	Subsystem
	Customer Information
	Admin (Control) Table. These tables are referenced as foreign keys on master and transaction tables. We do not document the names of these tables in this document as the table names are easily accessible using the Table transaction.
	N/A – the entity is maintained in respect of a higher level entity.
	N/A – the values in these types of entities are defined in a special table referred to as the lookup table. In order to determine the valid values for a column that references a lookup table, use the name of the column as the search value on the Look Up user interface.
	Meter Management
	Meter Reading
	Rates
	Billing
	Financial Transaction
	Payment
	Field Order
	Adjustment

Relationships

The solid line connecting the two entities that is terminated by an arrow represents a relationship between two entities. You read the relationship from the entity without the arrow to the entity with the arrow. For example in the following diagram, the line between Customer Class and Account illustrates that a Customer Class may have many Accounts, but an Account may be part of a single customer class.



Appendix B - Multiple Owners In A Single Database

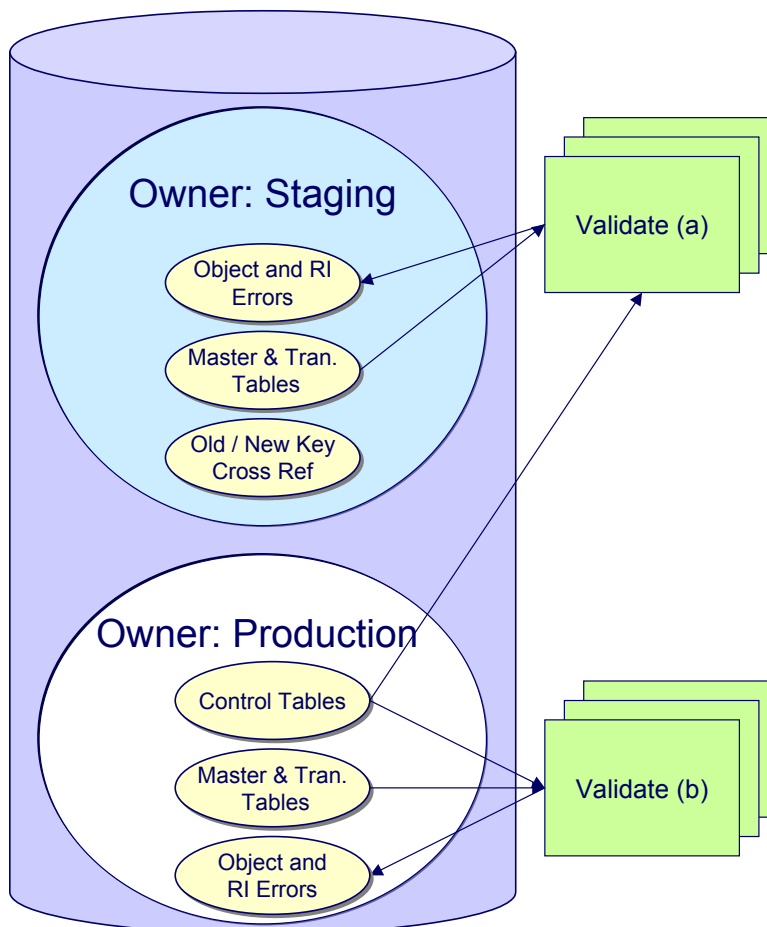
In the schematic referenced in the [Introduction](#), you'll notice that there are two table owners in the system database. We refer to the first owner as “staging” and the second owner as “production”.

The staging owner is linked to the tables into which you insert your pre-validated data. These tables have an owner ID of **CISSTG**.

Multiple staging databases. It is possible to have multiple staging databases. In this situation, each one would have a unique owner ID, e.g., **CISSTG1**, **CISSTG2**, etc.

The production owner is linked to the tables used by your production system. These tables have an owner ID of **CISADM**.

When the validation programs run against your staging data, they validate the staging data against the production control tables (and insert errors into the staging error table). This means that the SQL statements that access / update master and transaction data need to use the staging owner (**CISSTG**). Whereas the SQL statements that access control tables need to use the production owner (**CISADM**).



But notice that when these same programs run against production (Validate (b)), the SQL statements will never access the staging owner. Rather, they all point at the production owner.

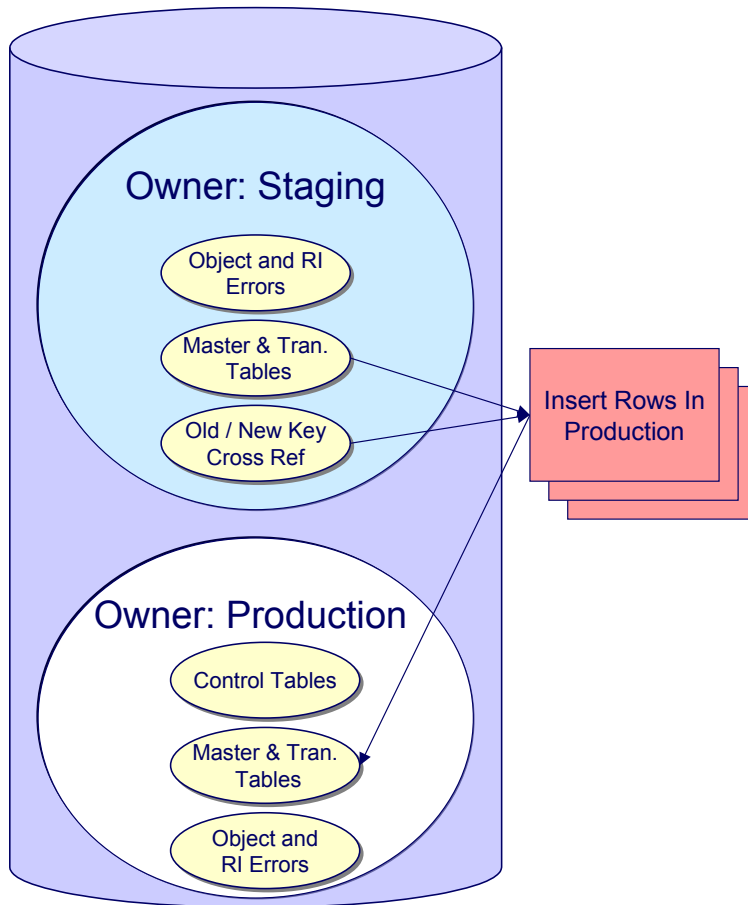
This is accomplished as follows:

- A separate application server must exist for each owner. Each application server points at a specific Tuxedo server. The Tuxedo server references a specific database user ID.
- The database user ID associated with the staging database uses **CISSTG** as the owner for the master and transaction tables, but it uses **CISADM** as the owner of the production control tables.
- The database user ID associated with the production database uses **CISADM** as the owner for the master, transaction, and control tables.

You may wonder why we went to this trouble. There are several reasons:

- We wanted to re-use the validation logic that exists in the programs that validate your production data. In order to achieve this, these programs must sometimes point at the staging owner, and other times they must point at the production owner (and this must be transparent to the programs otherwise two sets of SQL would be necessary).
- We wanted to let you use the application to look at and correct staging data. This can be accomplished by creating an application server that points at your staging database with the ownership characteristics described above.
- We wanted the validation programs to be able to validate your production data (in addition to your staging data). Why would you want to validate production data if only clean data can be added to production? Consider the following scenarios:
 - After an upgrade, you might want to validate your production data to ensure your pre-existing user-exit logic still works.
 - You may want to conduct an experiment of the ramifications of changing your validation logic. To do this, you could make a temporary change to user exit logic (in production) and then run the validation jobs on a random sample basis.
 - You forget to run a validation program before populating production and you want to see the damage. If you follow the instructions in this document, this should never happen. However, accidents happen. And if they do, at least there's a way to determine the ramifications.

While the redirection of owner ID's is a useful technique for the validation programs, it cannot be used by the key assignment and production insert programs? Why, because these programs have to access the same tables but with different owners. For example, the program that inserts rows into the person table must select rows from staging.Person and insert them into production.Person.



This is accomplished as follows:

- Views exist for each table that exists in both databases. These views have hard-coded the database owner **CISADM** (production). For example, there is a view called CX_PER that points at person table in production.
- The key assignment and insertion programs use these views whenever then need to access production data.

Appendix C - Known Oddities

Be aware that the following tables reference master data (e.g., persons, accounts). This means that if you look at them using a user ID that defaults ownership to the staging level, you will not be able to see the related master data (because the person / account doesn't exist in the staging owner's tables).

- Collection Agency. References a person.
- Service Provider. References a person and a service agreement.
- 3rd Party Payor. References an account.
- Tender Source. References a suspense account.

