Oracle
Public Sector Revenue Management

Administration Guide

Release 2.4.0 Service Pack 2

E49975-03

August 2015

Oracle Public Sector Revenue Management Administration Guide

Release 2.4.0 Service Pack 2

E49975-03

August 2015

Documentation build: 8.3.2015 14:40:38 [T1_1438638038000]

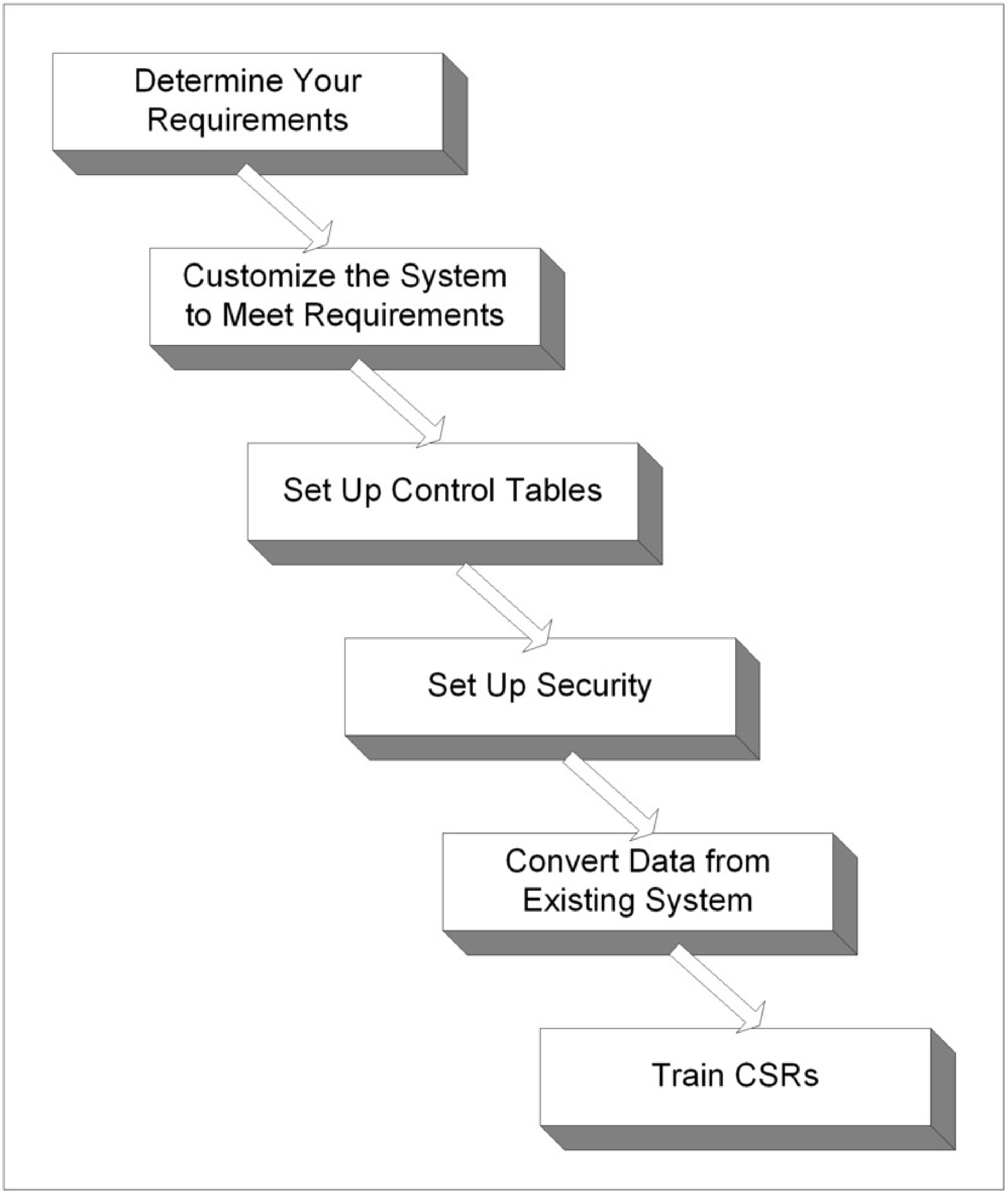
Contents

Chapter 1

Preparing To Implement

Getting ready for production takes a good deal of planning. You have probably already begun analyzing your requirements according to your business and organizational needs. You will need to review your current environment and think about what changes could be made now and in the future. And while you might have decided to simply transfer your current processing structure to Oracle Public Sector Revenue Management, you may also have discovered that the product can provide new options.

Because the system is sophisticated and customizable, there are a number of steps involved in rolling out and using your new system.


Determine Your Requirements
Customize the System to Meet Requirements
Set Up Control Tables
Set Up Security
Convert Data from Existing System
Train CSRs

The topics in this section describe the order in which the control tables should be set up.

Control Table Setup Sequence

To implement the system, you must set up your organization's business rules in "control tables". Setting up these tables is time-consuming because we allow you to tailor many aspects of the system to meet your organization's requirements. We strongly recommend that you take the time to document how you plan to set up all of these tables before you use the following roadmap to enter the control data. Time spent understanding the interrelationships between this data will reap the rewards of a clean system that meets your current and long term needs.

While we describe the transactions and options in more detail in other sections of this manual, use the following chart (and the remaining sections of this chapter) as your roadmap. Here we list the order in which you perform tasks and the pages you'll use to set up your system. The order is important because some information must exist before other information can be defined (i.e., many dependencies exist).

NOTE: Auto setup. The Auto Setup column in the following table contains suggestions to save you time. It also indicates if a control table contains information when the system is installed.

NOTE: You don't have to set up every control table. You need only set up those control tables that govern functions that are applicable to your organization.

NOTE: Algorithm references. In the sections below when it is mentioned that an algorithm is needed, note that the base product may provide base algorithms that can be used as is if they satisfy the implementation’s needs. Additionally, the product may provide an algorithm type but no algorithms in cases where the algorithm requires parameters that the implementation may need to configure. Finally, the implementation may determine that a new Algorithm Type and its Algorithm are needed to support the desired business functionality.

Function	***Menu***	***Auto Setup***
Global Context		
Algorithm	Admin Menu, Algorithm. You will need to set up an algorithm that populates global context values. The global context is used by various zones in the system to display relevant data. This algorithm is plugged-in on the *installation record*.	
Accounting Related Tables		
Country & State	Admin Menu, Country	
Currency Codes	Admin Menu, Currency Code	**USD** is automatically populated
Accounting Calendar	Admin Menu, Accounting Calendar	
GL Division	Admin Menu, General Ledger Division	
Security Related Tables		
Application Service	Admin Menu, Application Service	All base package transactions are automatically populated
Security Type	Admin Menu, Security Type	
User Group	Admin Menu, User Group Note, you won't be able to set up users at this point	One user group, **ALL-SERVICES,** is automatically set up. It references all other application services and a single user called **SYSUSER**. Note: You may be able to *import user groups if your organization has already defined them using LDAP*.
Language	Admin Menu, Language	**ENG** is automatically populated
Display Profile	Admin Menu, Display Profile	Two display profiles are automatically set up: **NORTHAM** displays currencies and dates in

Function	***Menu***	***Auto Setup***
		a classic American format; **EURO** displays information in a classic European format
Data Access Role	Admin Menu, Data Access Role	**CI_DFLT** is automatically populated. It references a single user called **SYSUSER** and a single access group called **CI_ ACCGRP**
Access Group	Admin Menu, Access Group	**CI_ACCGRP** is automatically populated. It references the data access role **CI_DFLT**
User	Admin Menu, User	**SYSUSER** is automatically set up. Note, you may be able to *import your users if your organization has already defined them using LDAP*.
Return to User Group	You must return to your user groups and define all of their users	
Account Type Related Tables		
Account Type	Admin Menu, Account Type. At this point, you'll only be able to set up your account type codes. You will return to these account types throughout the setup process to populate additional information.	
Financial Transaction Related Tables		
Work Calendar	Admin Menu, Work Calendar	
Division	Admin Menu, Division	
Revenue Class	Admin Menu, Revenue Class	
Algorithm	Admin Menu, Algorithm. You will need to set up the algorithm that constructs a distribution code's corresponding GL account when it is interfaced to the general ledger	
Distribution Code	Admin Menu, Distribution Code	
Bank & Bank Accounts	Admin Menu, Bank	
Algorithm	Admin Menu, Algorithm. You will need to set up the algorithm that constructs a payment segment's financial transaction	
Payment Segment Type	Admin Menu, Payment Segment Type	
Algorithm	Admin Menu, Algorithm. You will need to set up the algorithm that constructs an adjustment's financial transaction	
Payment event distribution related control tables need to be set up if your organization uses the distribution rules method to create payment events. Refer to *Setting Up The System To Use Distribution Rules* for a description of the tables that must be set up to enable this functionality.		
Algorithm	Admin Menu, Algorithm. Several plug-in spots are available to perform additional logic when processing adjustments. For example, if you have the system calculate adjustments, you must set up an adjustment generation algorithm. Refer to *Adjustment Type* for other available plug-in spots that may be used by your implementation.	
Algorithm	Admin Menu, Algorithm. You may want to set up an algorithm that formats the Adjustment information that is displayed throughout the system for a specific Adjustment Type. This algorithm is plugged-in on the *Adjustment Type*.	
Algorithm	Admin Menu, Algorithm. You may want to set up an algorithm that formats the Adjustment information that is displayed throughout the system. This algorithm is plugged-in on the *installation record*.	
Debt Category	Admin Menu, Debt Category	

Function	*Menu*	*Auto Setup*
Debt Category Priority	Admin Menu, Debt Category Priority	
Adjustment Type	Admin Menu, Adjustment Type	
Adjustment Type Profile	Admin Menu, Adjustment Type Profile	
Approval Profile	Admin Menu, Approval Profile. Note, an approval profile references a To Do type and one or more To Do Roles; these must be set up before you can set up the approval profile. After the approval profile(s) are set up, they must be referenced on the adjustment types that they govern.	
Cancel Reason - Payment	Admin Menu, Payment Cancel Reason	
Cancel Reason - Adjustment	Admin Menu, Adjustment Cancel Reason	
Tender Type	Admin Menu, Tender Type	
Tender Source	Admin Menu, Tender Source	
A/P Request Type	Admin Menu, A/P Request Type	
Installation	Admin Menu, Installation Options - Framework and Admin Menu, Installation Options. Many fields on the installation record impact the financial transaction environment. Refer to the description of the *Financial Transaction* tab for more information.	
Algorithm	Admin Menu, Algorithm. You will need to set up an algorithm that distributes payments.	
Algorithm	Admin Menu, Algorithm. You will need to set up an algorithm that handles overpayment situations.	
Algorithm	Admin Menu, Algorithm. You may need to set up an algorithm if you want the system to levy a non-sufficient funds charge if a payment is canceled due to non-sufficient funds.	
Algorithm	Admin Menu, Algorithm. You will need to set up an algorithm that formats the payment information that is displayed throughout the system. This algorithm is plugged-in on the *installation record*.	
Algorithm	Admin Menu, Algorithm. You will need to set up an algorithm that defaults the amount when a payment is manually added. This algorithm also calculates the amount of an automatic payment for a bill for an account with an active auto pay option. This algorithm is plugged-in on the *installation record*.	
Algorithm	Admin Menu, Algorithm. Refer to *Account Type* for other available plug-in spots that may be used by your implementation to perform additional logic when processing payments.	
Return to Account Type	Admin Menu, Account Type. You will need to plug-in the algorithms defined above on your account types.	
Address Environment		
Address Type	Admin Menu, Extendable Lookup, search for the Address Type lookup. Define valid address types for person addresses and tax role addresses.	
Address Inactive Reasons	Admin Menu, Extendable Lookup, search for the Address Inactive Reason lookup. Define valid address inactive reasons to use when changing the status of an address to inactive.	
Algorithm	Admin Menu, Algorithm. You will need an algorithm that finds an address if your	

Function	*Menu*	*Auto Setup*
	implementation receives requests from external sources to define or update a taxpayer’s address. This plug-in determines if the address already exists in the centralized address repository. This algorithm is plugged-in on the *installation record*.	
Taxpayer Environment		
Account Management Group	Admin Menu, Account Management Group. Note: You will probably have to set up To Do Type and To Do Roles before you can set up account management groups. Refer to *Assigning A To Do Role* for more information on how account management groups may be used to define an entry's role.	
Account Relationship	Admin Menu, Account Relationship Type	
Alert Type	Admin Menu, Alert Type	
Algorithm	Admin Menu, Algorithm. If you have software capable of reconstructing an image of a letter in a PDF (for the purpose of online display), you will need to create an algorithm that formats the extract records that are sent to your letter image software.	
Letter Template	Admin Menu, Letter Template	
Customer Contact Class	Admin Menu, Customer Contact Class	
Customer Contact Type	Admin Menu, Customer Contact Type	
Algorithm	Admin Menu, Algorithm. You may need to set up the algorithms that determine if person ID's are in a predefined format.	
Identifier Type	Admin Menu, Identifier Type	
Industry Codes	Admin Menu, Industry Code	
Algorithm	Admin Menu, Algorithm. You may need to set up the algorithms that determine if phone numbers are in a predefined format.	
Phone Type	Admin Menu, Phone Type	
Person Relationship Type	Admin Menu, Person Relationship Type.	
Person Type	Admin Menu, Person Type	
Algorithm	Admin Menu, Algorithm. You will need to set up an algorithm that formats the person information that is displayed throughout the system. This algorithm is plugged-in on the *installation record*.	
Algorithm	Admin Menu, Algorithm. You can override the system's standard account information string by setting up an algorithm that produces this string of information. This algorithm is plugged-in on the *installation record*.	
Installation	Admin Menu, Installation Options. Many fields on the installation record impact the Customer Environment. Refer to the description of the *Main*, *Person*, and *Account* tabs for more information.	
Automatic Payment (EFT) Environment		
Algorithm	Admin Menu, Algorithm. You will need to set up an algorithm to create automatic payments. This algorithm is plugged-in on the *installation record*.	
Tender Source	Admin Menu, Tender Source Note: Earlier, you created tender sources for the remittance processor and your cash drawers. At this point, you'll need to add	

Function	Menu	Auto Setup
	at least one tender source for automatic payments. Why? Because automatic payments get linked to a tender control (which, in turn, gets linked to a tender source) when they are interfaced out of the system.	
Algorithm	Admin Menu, Algorithm. You will need to set up the appropriate automatic payment date calculation algorithm to populate the extract, GL interface and payment dates on automatic payments.	
Auto Pay Route Type	Admin Menu, Auto Pay Route Type	
Tender Type	Admin Menu, Tender Type Note: Earlier, you created tender types for things like cash, checks, etc. At this point, you'll need to add a tender type for each type of automatic payments (e.g., direct debt, credit card, etc.).	
Work Calendar	Admin Menu, Work Calendar. You need only set up additional work calendars if the auto pay sources (i.e., the financial institutions) have different working days than does your organization	
Algorithm	Admin Menu, Algorithm. If you need to validate the taxpayer's bank account or credit card number, you will need to set up the appropriate validation algorithms.	
Auto Pay Source Type	Admin Menu, Auto Pay Source Type	
Algorithm	Admin Menu, Algorithm. You may need to set up an algorithm if your taxpayers can define a maximum withdrawal limit on their autopay options.	
Return to Account Type	Admin Menu, Account Type. You should plug-in the Autopay Over Limit Algorithm in each appropriate Account Type.	
Characteristics		
Algorithm	Admin Menu, Algorithm. If you have ad hoc characteristic types, you may need to set up the algorithms that control how they are validated	
Foreign Key Reference	Admin Menu, FK Reference. If you have foreign key characteristic types, you may need to set up foreign key references to control how the user selects the characteristic values (and how the foreign key values are validated).	All base package FK references are automatically populated
Characteristic Type & Values	Admin Menu, Characteristic Type	
Tax Type and Obligation Configuration		
Revenue Calendar	Admin Menu, Revenue Calendar	
Tax Type	Admin Menu, Tax Type	
Algorithm	Admin Menu, Algorithm. You will need to set up the algorithms that determine: • Special processing that should take place when obligations of a given type are created. • Special processing that should take place when a financial transaction is frozen for obligations of a given type.	
Algorithm	Admin Menu, Algorithm. You may want to set up an algorithm that formats the obligation information that is displayed throughout the	

Function	***Menu***	***Auto Setup***
	system. This algorithm is plugged-in on the *installation record*.	
Algorithm	Admin Menu, Algorithm. You may want to set up an algorithm that formats the obligation information that is displayed throughout the system for a specific obligation type. This algorithm is plugged-in on the *Obligation Type*.	
Tax Role Relationship Type	Admin Menu, Tax Role Relationship Type.	
Obligation Type	Admin Menu, Obligation Type	
Overpayment Configuration		
Algorithm	Admin Menu, Algorithm. You will need to define an extract algorithm that formats the data to be sent to a bank for direct deposit requests. These are plugged in on the bank event type.	
Bank Event Type	Admin Menu, Bank Event Type Note, if your implementation supports receiving rejections from the bank that should cause another attempt to refund, you will need a “retry” overpayment process type.	
Algorithm	Admin Menu, Algorithm. You will need to define overpayment validation and refund validation algorithms. These are plugged in on the overpayment process type.	
Overpayment Process Type	Prior to defining your overpayment process types, refer to *Enabling the System to Use Standard Overpayment Process* for information about data that needs to be defined for a typical overpayment process type. Admin Menu, Overpayment Process Type. If you have defined a “retry” overpayment process type, return to bank event to configure this value.	
Refund Control Type	Admin Menu, Refund Control Type	
Wrap Up		
Algorithm	Admin Menu, Algorithm. You will need to set up the algorithms that determine: • Special alerts on the dashboard (assuming you have special alerts)	
Installation Options	Admin Menu, Installation Options - Framework and Admin Menu, Installation Options. At this point, it's a good idea to double-check everything on the installation record.	
Postal Default	Admin Menu, Postal Code Default	

If you have cash drawers you will also need to set up the following information:

- Create a person / account to which you will link your over / under obligation. Refer *How To Get An Unbalanced Tender Control In Balance (Fixing Over/Under)* for more information.
- Create an obligation to which your over/under payments will be linked. This obligation will reference your over / under obligation type. Refer to *Over / Under Cash Drawer Obligations* for more information.

Cross Reference To The Remaining Chapters

The table in the previous section describes the order in which you should enter your control tables. These tables are described at length in the following chapters.

- Refer to *Defining General Options Addendum* and *Defining General Framework Options* for a discussion of the control tables associated with general functionality (e.g., country codes, state codes, etc.).
- Refer to *Defining Financial Transaction Options* for a discussion of the tables affecting your financial transactions (e.g., adjustment types, payment segment types, etc.).
- Refer to *Defining Taxpayer Options* for a discussion of the control tables affecting persons, accounts, tax roles and obligations.
- Refer to *Configuring Obligation Types* for a discussion of the control tables affecting your obligation types.
- Forms processing tables and form rules need to be set up for processing registration forms, tax forms and form upload staging tables. Refer to *Defining Forms Processing Options* and *Defining Form Rules* for more information.
- Penalty and interest related configuration tables need to be set up for penalty and interest calculations. Refer to *Setting Up Penalty and Interest Options* for a description of tables that must be set up to enable this functionality.
- Refer to *Setting Up Asset Options* for a description of tables that must be set up to enable asset and valuation functionality.
- Refer to *Defining Calculation Engine Options* for a description of tables that must be set up to enable calculation engine functionality.
- Refer to *Configuring Bills* for a description of tables that must be set up to enable billing functionality.
- Overdue processing tables need to be set up for collection activity. Refer to *Overdue Processing - Setup Tasks* for a description of tables that must be set up to enable this functionality.
- Refer to *Setting Up Bankruptcy Options* for a list of objects to be defined for bankruptcy handling.
- Refer to *Setting Up Audit Case Options* for a list of objects to be defined for audit cases.
- Refer to *Setting Up Appeal Options* for a list of objects to be defined for appeals.
- Refer to *Setting Up Review Options* for a list of objects to be defined for reviews.
- Refer to *Setting Up Suppression Options* for a list of objects to be defined for suppression handling.
- Refer to *Defining Background Process* for a discussion of the control tables affecting your background processes.
- Refer to *Defining Algorithms* for a discussion of the control tables affecting the algorithms referenced on many control tables.
- Most zones are delivered with the base-package and do not require any configuration. However, some zones are only available if configured by your implementation. Refer to *Configuring Zones* for more information.
- Refer to *Setting Up To Do Options* for more information on how to configure the system to match your organization's To Do management needs.
- If your organization plans to use process flows for business functionality not supported in the base product, refer to *Setting Up Process Flows* for more information on setting up the system to use process flow.
- Open-item accounting tables need only be set up if your organization practices *Open Item Accounting*. Refer to *Setting Up The System To Enable Open Item Accounting* for a description of the tables that must be set up to enable this functionality.
- If your implementation finds that it needs to set up rates for certain calculations, refer to *Rates* for a discussion of the control tables affecting your rates.

Chapter 2

Defining General Options Addendum

This section describes control tables that are used throughout the product.

Defining Installation Options

The topics in this section describe the various installation options that control various aspects of the system that are specific to the Oracle Public Sector Revenue Management product.

FASTPATH:
Refer to *Installation Options - Framework* for options that are common to products on the same framework.

Installation Options - Main

Select **Admin Menu** > **Installation Options** and use the **Main** tab to define system-wide installation options.

Description of Page

Use **Quick Add Tender Type** to define the tender type defaulted on payments added using the *Payment Quick Add* transaction.

Use **Starting Balance Tender Type** to define the tender type of the starting balance recorded on your tender controls (this will almost always be the tender type associated with "cash"). This value is used during tender control balancing as a separate balance is required for each tender type in order to balance a tender control. Refer to *The Lifecycle Of A Tender Control* for more information.

FASTPATH:
For more information, refer to *Setting Up Tender Types*.

Location Geo Type in only visible if the address feature configuration is set to *Legacy Address*. It is used to indicate whether at least one geographic identifier (e.g., GPS coordinate) is **Required** or **Optional** on a location. Refer to *Defining Geographic Types* for more information.

The **Alternate Representation** flag should be set to **None** unless your organization uses multiple character sets for a person's main name and / or a location's address. Alternate representations are typically only used in countries where multiple character sets are used. For example,

- In Hong Kong, a person's name may be written in both Chinese characters and in English.
- In Japan, a person's name may be written in both Kanji and Katakana.

In both of the above situations, users need to be able to use both representations to find a taxpayer or a location.

If your organization uses alternate representations of person name or address, set this flag to one of the following values:

- Use a value of **Name** if you only use alternate representations for a person's primary names.
- Use a value of **Address** if you only use alternate representations for location addresses. This option is only available if the *address support* feature configuration is set to Legacy.
- Use a value of **Name & Address** if you use alternate representations for both location addresses and person names. This option is only available if the *address support* feature configuration is set to Legacy.

The following points describe the ramifications of this flag in the system:

- If you support alternate representations of a person's primary name,
  - The name grid on *Person - Main* allows you to specify an **Alternate** name for the person.
  - If you use the base package *name formatting algorithm*, a person's name will be shown throughout most of the system in the format AAA (BBB), where AAA is the person's primary name and BBB is the person's alternate name. Note, this format does not apply to names that appear in search results (i.e., the alternate name is not concatenated to the main name in search results; however you can search for information using the alternate name).
  - Most of the system's person name-oriented searches will allow users to use both a person's primary and alternate names to search for information.
- If you support alternate representations of a location's address,
  - A tab is available on the *Location* page that allows a user to define an alternate address for a location.
  - If you use the base package *location formatting algorithm*, a location's address will be shown throughout most of the system in the format AAA (BBB), where AAA is the location's primary address and BBB is the location's alternate address.
  - Most of the system's location-oriented searches will allow users to use both a location's primary and alternate addresses to search for information.

CAUTION:
In order to improve response times, installation options are cached the first time they are used after a web server is started. If you change this field's option and you don't want to wait for the cache to rebuild, you must clear the cached information so it will be immediately rebuilt using current information. Refer to *Caching Overview***for information** on how to clear the system login cache (this is the cache in which installation options are stored).

Installation Options - Person

Select **Admin Menu** > **Installation Options** and use the **Person** tab to define person-specific installation options.

Description of Page

Use the **Person ID Usage** to indicate whether or not at least one form of identification is **Required** or **Optional** when a new person is added.

Each form of identification has an identifier type. For persons that are humans (as defined by the *person type's* person/business identifier), the *Person* page defaults the identifier type defined in **Identifier Type (Person)**. For persons that are businesses (as defined by the *person type's* person/business identifier), the Person page defaults the identifier type defined in **Identifier Type (Business)**.

Installation Options - Account

Select **Admin Menu** > **Installation Options** and use the **Account** tab to define account-specific installation options.

Description of Page

When a new account is added on the *Account* page, the system requires it to have an account type. If the main taxpayer linked to the account is a human (as defined by the taxpayer's person type), the system defaults the account type defined in **Account Type (Person)**. For persons that are businesses (as defined by the person type), the system defaults the account type defined in **Account Type (Business)**. For more information, refer to *Setting Up Account Types*.

In addition to requiring an account type when a new taxpayer is added, the system also requires a "main taxpayer" (i.e., a reference to a person who is identified as the main taxpayer for the account). Enter the default **Account Relationship Type** code to be used to define the main taxpayer relationship. For more information, refer to *Setting Up Account Relationship Codes*.

Enter the default **Bill Route Type** to be used to define how classic bills should be routed to a taxpayer. For more information, refer to *Setting Up Bill Route Types*.

NOTE: The bill route type field is only visible if your implementation has not *turned off* the **BI — Billing (Classic)** module. This information is used only for classic billing functionality.

Installation Options - Billing

Select **Admin Menu** > **Installation Options** and use the **Billing** tab to define classic billing-specific installation options.

NOTE: **Optional information.** This tab is only visible if your implementation has not *turned off* the **BI — Billing (Classic)** module. The information is used only for classic billing functionality.

Description of Page

The **Bill Segment Freeze Option** controls when an obligation's balance and the general ledger are affected by classic bill segments and certain types of adjustments. Refer to *Preventing Obligation Balances And The GL From Being Impacted Until Bill Completion* to understand the significance of this option.

The **Accounting Date Freeze Option** controls how the accounting date defined on financial transactions is populated. Refer to *Forcing The Freeze Date To Be Used As The Accounting Date* to understand the significance of this option.

Define the **Minimum Amount for Final Bill**. If a final bill is less than this amount, the bill is still produced; it's just not printed.

Typically, the system sets a bill's Bill Date equal to the date on which it is completed. If you want to be able to specify a bill's Bill Date when you complete a bill, turn on **User Can Override Bill Date**. You would only want to override the bill date if you are setting up sample bills from historical period whose bill date needs to reflect the respective historical period.

Installation Options - Collections

Select **Admin Menu** > **Installation Options** and use the **Collections** tab to define collections-specific installation options.

Description of Page

Enter what you consider to be an excellent compliance rating in **Beginning Compliance Rating**. Overdue events can cause an account's compliance rating to decrease. When an account's compliance rating falls below a certain level, different overdue processes may ensue.

Use **Compliance Rating Threshold** to define when an account's compliance rating becomes risky. When an account's compliance rating falls beneath the Compliance Rating Threshold, the system will show an alert in the alert zone highlighting such.

Installation Options - Financial Transaction

Select **Admin Menu** > **Installation Options** and use the **Financial Transaction** tab to define financial transaction installation options.

Description of Page

Use **G/L Batch Code** to define the batch process that is used to interface your financial transactions to your general ledger. The process is snapped on FT download records by the GLS background process.

Use **A/P Batch Code** to define the batch process that is used to interface your check requests (initiated with adjustments with an adjustment type that reference an A/P request type) to you accounts payable system.

Use **Fund Accounting** to indicate if *fund accounting* is **Practiced** or **Not Practiced** at your organization. By default, the installation indicates that it is **Practiced**.

The **Accounting Date Freeze Option** controls how the accounting date defined on financial transactions is populated. Refer to *Forcing The Freeze Date To Be Used As The Accounting Date* to understand the significance of this option.

NOTE: If your implementation has not *turned off* the **BI — Billing (Classic)** module, the Billing tab will be visible and the accounting date freeze option field will be maintained on that tab.

Installation Options - Algorithms

The following table describes **System Events** that are provided by the product in addition to the ones provided in the framework as described in *Installation Algorithms*.

System Event	***Optional / Required***	***Description***
Account Information	Optional	We use the term "Account information" to describe the basic information that appears throughout the system to describe an account. The data that appears in "account information" may be constructed using an algorithm plugged in here. Refer to *Defining Account Information* for more information on when this algorithm is used. Click *here* to see the algorithm types available for this system event.
Address Information	Optional	This algorithm allows your implementation to define a common format for displaying an

		address. It receives address constituents and returns a formatted address string. This plug-in spot may be invoked by other algorithms using the business service **C1-FormatAddressInfo**. Click *here* to see the algorithm types available for this system event.
Adjustment Information	Optional	We use the term "Adjustment information" to describe the basic information that appears throughout the system to describe an adjustment. The data that appears in "Adjustment information" may be constructed using an algorithm plugged in here. Refer to *Adjustment Information* for more information on when this algorithm is used. Click *here* to see the algorithm types available for this system event.
Alert	Optional	There are two types of alerts that appear in the *Alert Zone* : 1) hard-coded system alerts and 2) alerts constructed by plug-in algorithms. You cannot change the hard-coded alerts (see the *Alert Zone* for the complete list). However, by plugging in this type of algorithm you can introduce additional alerts. An error displays if more than 60 alerts are generated for an account by plug-in algorithms. Click *here* to see the algorithm types available for this system event.
Automatic Payment Creation	Required if you allow taxpayers to *pay automatically*	This algorithm is executed to create automatic payments whenever the system creates automatic payments. Refer to *How And When Are Automatic Payments Created* for the details. Click *here* to see the algorithm types available for this system event.
Bank Details Determination	Required	This plug-in spot is used for processing bank details for an Account, Tax Role or Obligation. It can be called in any of three modes: • Read - Retrieves the effective bank details. • Remove - Resets or end-dates existing bank details that are deemed no longer valid. • Get Master - Finds the related Account ID / Tax Role ID / Obligation ID using supplied bank details. Click *here* to see the algorithm types available for this system event.
Case Information	Optional	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. Plug an algorithm into this spot to override the system default "Case information". Note: This algorithm may be further overridden by a "Case information" plug-in on the Case Type. Refer to *Case Type* for how algorithms of this type are used. Click *here* to see the algorithm types available for this system event.

Collection Agency Referral Information	Optional	We use the term "Collection Agency Referral information" to describe the basic information that appears throughout the system to describe a collection agency referral. Plug an algorithm into this spot to override the system default "collection agency referral information". Click *here* to see the algorithm types available for this system event.
Compliance Rating "Created By" Information	Required	The data that appears in the compliance rating "created by" information is constructed using this algorithm. Refer to *Account - Collections* for more information about the compliance rating. Click *here* to see the algorithm types available for this system event.
Compliance Rating History Information	Optional	We use the term Compliance Rating History information to describe the basic information that appears throughout the system to describe a *compliance rating history* entry. Plug an algorithm into this spot to override the system default "compliance rating history information". Click *here* to see the algorithm types available for this system event.
Determine Distribution Rule	Required if your implementation uses the distribution rule method to interface payments.	This plug in spot is use to populate the account, distribution rule and value on payment event upload staging records as the first stage of processing. Refer to *Payment Event Upload Processing Overview* for more information on when this algorithm is used. Click *here* to see the algorithm types available for this system event.
Financial Transaction Info	Optional	We use the term "financial transaction information" to describe the basic information that appears throughout the system to describe a financial transaction. The data that appears in "financial transaction info" is constructed using this algorithm. Click *here* to see the algorithm types available for this system event.
Find Address	Optional	This plug-in spot is used to find an address in the centralized address repository based on input address constituents. It receives constituents and an algorithm plugged into this spot is responsible for looking at the Address table to find a unique match and return the ID. Refer to *Centralized Address* for more information . Click *here* to see the algorithm types available for this system event.
Obligation Information	Optional	We use the term "obligation information" to describe the basic information that appears throughout the system to describe an obligation. The data that appears in "obligation information" may be constructed using an algorithm plugged in here. Refer to *Defining Obligation Information* for more information on when this algorithm is used.

		Click *here* to see the algorithm types available for this system event.
Online Letter Image	Optional	This algorithm constructs a PDF that contains the image of a letter. This algorithm is executed when the Display Letter button is pressed on *Customer Contact - Main*. Refer to *Technical Implementation of Online Letter Image* for more information. Click *here* to see the algorithm types available for this system event.
Payment Amount Calculation	Required	This algorithm is executed to calculate the amount of an automatic payment for an account with an active auto pay option. Refer to *How And When Are Automatic Payments Created* for more information on automatic payments. This algorithm is also executed to default the amount of a manually added payment. Refer to *How To Add A New Payment Event* for more information on adding a payment manually. Click *here* to see the algorithm types available for this system event.
Payment Information	Required	We use the term "payment information" to describe the basic information that appears throughout the system to describe a payment. The data that appears in "payment information" is constructed using this algorithm. Click *here* to see the algorithm types available for this system event.
Person Information	Required	We use the term "person information" to describe the basic information that appears throughout the system to describe a person. The data that appears in "person information" may be constructed using an algorithm plugged in here. Refer to *Person Information* for more information on when this algorithm is used. Click *here* to see the algorithm types available for this system event.
Person Name Validation	Optional	This plug-in spot is only applicable if an implementation has not implemented person name validation via person business objects. The format of names entered on *Person - Main* is validated using this algorithm. Click *here* to see the algorithm types available for this system event.

The following table highlights algorithms related to *legacy addresses* or *classic billing*.

System Event	***Optional / Required***	***Description***
Bill Information	Required	This plug-in spot is only applicable to implementations supporting the *classic billing* functionality. We use the term "Bill information" to describe the basic information that appears throughout the system to describe a bill. The data that appears in "bill information" may be constructed using an algorithm plugged in here. Click *here* to see the algorithm types available for this system event.
Determine Open Item Bill Amounts	Required if you use overdue functionality to *collect on bills*	This plug-in spot is only applicable to implementations supporting the *classic billing* functionality and open item accounting.

		This algorithm is responsible for determining the unpaid amount of an open-item bill. It can also be used to return the unpaid amount for a specific Obligation on a bill. Click *here* to see the algorithm types available for this system event.
Location Information	Optional	This plug-in spot is only applicable to implementations supporting the *legacy* address functionality. We use the term "location info" to describe the basic information that appears throughout the system to describe a location. The data that appears in "location info" may be constructed using an algorithm plugged in here. Refer to *Defining Location Information* for more information on when this algorithm is used. Click *here* to see the algorithm types available for this system event.
Online Bill Display	Optional	This plug-in spot is only applicable to implementations supporting the *classic billing* functionality. This algorithm constructs a PDF that contains the image of a bill. This algorithm is executed when the Display Bill button is clicked on the *Classic Bill* page. Refer to *Technical Implementation of Online Bill Image* for more information. Click *here* to see the algorithm types available for this system event.
Override Proration Factors	Optional	This plug-in spot is applicable to rates but is not expected to be a common requirement. It is most likely to be applicable to implementations supporting the *classic billing* functionality. Algorithms for this spot are only used if your organization has unusual rate proration requirements that necessitate the overriding of the base package proration logic. For example, you may have certain rate components whose charges should never be prorated. Refer to *Overriding Proration Factors* for more information. Click *here* to see the algorithm types available for this system event.
Override Seasonal Proration	Optional	This plug-in spot is applicable to rates but is not expected to be a common requirement. It is most likely to be applicable to implementations supporting the *classic billing* functionality. Algorithms for this spot are only used if your organization has unusual method of determining the seasons for your rate components. Refer to the description of the seasonal attributes for a *rate component* for more information. Click *here* to see the algorithm types available for this system event.

Defining Taxpayer Languages

As described under *Defining Languages*, you define the language in which each user sees the system. In addition to defining each user's language, the system allows you to define each person's preferred language. For example, one person can receive correspondence in English whereas another person could receive their correspondence in Chinese.

Each person's language is defined by the *language code* on the *person record*. The language code is passed on to all taxpayer-facing interfaces, such as correspondence.

NOTE: To support correspondence, you must also provide translations of the correspondence text. This must be handled by your printing software vendor.

Defining Accounting Calendars

The accounting calendar determines the accounting period to which a financial transaction will be booked. The following points describe how the system determines a financial transaction's account period:

- Every financial transaction references an accounting date and its obligation
- Every obligation references an obligation type
- Every obligation type references a GL division
- Every GL division references an accounting calendar
- The accounting calendar contains the cross reference between the accounting date specified on the financial transaction and related accounting period in your general ledger

CAUTION: This information must be the same as the information in your financial database.

To add or review an accounting calendar, choose **Admin Menu** > **Accounting Calendar**.

Description of Page

Enter a unique **Calendar ID** and **Description** for the calendar.

Enter the **Number Of Periods** for the calendar. Don't count the adjustment period, if you use one, or any special "system" periods.

Specify the **Fiscal Year**, each **Accounting Period** in that year, a **Period Description**, the **Begin Date** and the **End Date**.

When you enter begin and end dates, you can define monthly calendar periods or any fiscal period that matches your accounting calendar (weekly, bimonthly) as long as the begin and end dates of successive periods do not overlap.

For each fiscal period, enter the **Open From Date** and **Open To Date**. These dates define when that particular business dates are open for posting financial transactions to that fiscal period. For example, you might create a financial transaction on Sept 1 for taxes reported on August 31. To post this financial transaction in the August period, you must keep it open through Sept 1.

As time passes, you will need to return to this transaction to manually enter ensuing years. You can enter several years at a time or incorporate the task into end-of-year system maintenance.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_CAL_GL*

Defining General Ledger Divisions

There are two types of divisions referenced in the system: a division and a GL division.

- General Ledger divisions typically comprise individual entities (e.g., companies) in your general ledger. You must set up a GL division for each such entity. The GL division's sole purpose in the system is to define the accounting period associated with financial transactions linked to obligations associated with the GL division (obligations are associated with GL divisions via their obligation type). The system cares about accounting periods in order to prevent a user from booking moneys to closed periods. It also uses accounting periods when it produces the flat file that contains the consolidated journal entry that is interfaced to your general ledger (refer to *The GL Interface* for more information).

  NOTE: When determining how many GL Divisions you need, be sure to consider your general ledger and how your chart-of-accounts is structured. You will typically have one GL division for each "company" in your general ledger.

- A division is typically associated with a jurisdiction. The definition of a jurisdiction is a geographic-oriented entity with unique business rules. You must set up a division for each jurisdiction in which you conduct business.

  FASTPATH: Refer to *Setting Up Divisions* for information about divisions.

To define a general ledger division, select choose **Admin Menu** > **General Ledger Division**.

Description of Page

Enter a unique **GL Division** for the general ledger division.

Enter a **Description** of this general ledger division.

Define the accounting **Calendar ID** that controls how to convert an FT's accounting date into an accounting period. Refer to *Defining Accounting Calendars* for more information.

You may define a **Currency Code** for the GL division. Note that the system does not use this currency code.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_GL_DIVISION*.

Defining Banks & Bank Accounts

The topics in this section describe how to maintain your implementation's bank accounts.

Bank - Main

To add or review Banks choose **Admin Menu** > **Bank**.

Description of Page

Enter a unique **Bank Code** and **Description** for the bank.

The **Bank Accounts** collection displays the bank accounts currently linked to this bank code. Use the drill down button to view more details or to modify the bank account details. Alternatively, you may navigate to the Bank Account tab and scroll to the desired bank account.

Bank - Bank Account

To add or review Bank Accounts for a Bank, choose **Admin Menu, Bank** and then navigate to the **Bank Account** tab.

Description of Page

Use the **Bank Accounts** tab to define the attributes of each bank account. For each account, enter the following information:

- Enter a **Bank Account Key** to identify an Account at a Bank. You may have more than one account at a given bank, and you may have accounts at more than one bank. This code will allow the system to easily identify a specific account.
- Enter a **Description** to appear on prompt lists, inquiries, and reports.
- Enter the **Account Number**, **Check Digit** and if needed, the **Branch ID** of the bank where the account is held.
- Enter the **Currency Code** for the currency in which the account is denominated.
- Use **DFI ID** to define the Depository Financial Institution ID that is interfaced to the automatic payment-processing agent as part of the automatic payment interface.
- Enter the **Distribution Code** to be used for cash GL distributions when a payment is frozen or canceled.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_BANK_ACCOUNT*.

To Do Lists Addendum

This section is an addendum to the general *To Do Lists* chapter. This addendum describes the To Do functionality that is specific to Oracle Public Sector Revenue Management.

Assigning A To Do Role

As described in *To Do Entries Reference A Role*, each To Do entry requires a role. To Do entries created in Oracle Public Sector Revenue Management may attempt to assign a role based on an account management group or division if it is applicable to the type of data related to the To Do entry.

As described in *The Big Picture of To Do Lists*, users are informed that something requires their attention by entries that appear in a To Do List. For example, consider what happens when a form suspends:

- Form processing moves the form to suspended and stores an exception record.
- This in turn, triggers the creation of a To Do entry.
- The To Do entry is assigned a role. A role is one or more users who can look at / work on the entry.
- When users view a To Do List, they only see entries addressed to roles to which they belong.

You can optionally use account management groups (AMG) to define the respective role to be assigned to the To Do entries that are associated with an account and To Do type. For example, you can create an AMG called **Credit Risks** and assign this to accounts with suspect credit. Then, whenever an account-oriented To Do entry is created for such an account, it will be assigned a role based on the **Credit Risks** AMG. Refer to *Setting Up Account Management Groups* for more information.

By assigning an AMG to an account, you are telling the system to address this account's To Do list entries to the roles defined on the AMG (note, each To Do type can have a different role defined for it on an AMG).

You can optionally use division to define the respective role to be assigned to the To Do entries that are associated with an account and To Do type. Refer to *Setting Up Divisions* for more information.

A **To Do Pre-Creation** installation options plug-in is provided to determine the appropriate To Do Role for an account based on AMG and division setup. If plugged in, the logic to determine To Do role for an account is performed whenever a To Do entry is created. Refer to *C1-TDCR-DFRL* for further details on how this plug-in works.

FASTPATH: Refer to *To Do Entries Reference A Role* for the details of how an initial role is assigned to the To Do entries.

System To Do Types

NOTE: **List of available To Do types.** The To Do types available with the product may be viewed in the *application viewer'sTo Do type* viewer. In addition if your implementation adds To Do types, you may *regenerate* the application viewer to see your additions reflected there.

Configuring Zones

Most zones in the product do not require configuration by your implementation team. There are however a limited number of zones that require configuration before they can be used because their behavior is dynamic. The topics in this section provide some information about zones that require configuration.

FASTPATH: 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 taxpayer, and another that shows when forms have been posted for the taxpayer.

FASTPATH: For a complete description of the numerous features available on a timeline zone, refer to *Timeline Zone*.

The following points describe how to set up a timeline zone:

- Determine which portal you wish the zone to appear on. For example, you may configure the zone to appear on *360 Degree View - Account*, *360 Degree View - Tax Role* or for implementations still using Control Central, the *Control Central Account* tab.
- 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. Note that the base product supplies pre-configured algorithms for many of the timeline zone algorithm types that may be used if the configuration is acceptable. Click *here* to see the algorithm types available for this plug-in spot along with any pre-configured algorithms supplied with the product. Please note the following about the parameter values defined on these algorithms:
    - Most timeline algorithms are designed to display information about the account in context. However, three are several that are designed to display information for an account or for a specific tax role in context (for use on the 360 Degree View - Tax Role tab). If the algorithm includes a Restrict by Entity flag, indicate the appropriate value to control the data that should be shown.
    - Each timeline algorithm allows you to define the appropriate foreign key reference that controls the "info string" that appears as well as the navigation option. The object's info string appears in the zone's info area. When a user clicks on an "info string", they are transferred to an appropriate page (typically the one used to maintain the object).

- 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 payment 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., transfer a payment). 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 payment event is kicked off, the system populates a field called PAY_ID with the prime-key of the selected payment.

  The script can use this page data model field to navigate to the pertinent pages. For example, if you were setting up a script to transfer a payment, the first line of the script would reference a navigation option to transfer the user to the payment page where they can initiate the transfer. 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 one or more *zones* that reference these algorithms. The zones should reference the **F1-TIMELINE** zone type.
- Configure the zone's visibility parameter to define an appropriate visibility script so that the zone only displays when the relevant data is in context:
  - For account oriented zones, configure the following: ss='F1-ShldShwZn' input=[id=ACCT_ID] output=shouldShowZone
  - For tax role oriented zones, configure the following: ss='F1-ShldShwZn' input=[id=TAX_ROLE_ID] output=shouldShowZone
- 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 collection 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. Each separate zone includes its own date range controls.

Miscellaneous Topics

The following sections describe miscellaneous system wide topics.

Advanced Search Options

The product offers an option to perform fuzzy searching in the *360 Degree Search* by Person Name and in the Determine Taxpayer Existence form rule's apply rule algorithm (*C1-FR-CHTXEX*). The option is only available if your implementation has appropriately configured your database for fuzzy searching.

Refer to the DBA guide in the installation guide for details on setting up the database to support fuzzy searching. Note that there are some implementations where fuzzy searching will not be possible. For example, it's only available for implementations using the Oracle database. Additionally, not all languages are supported. Refer to the Oracle Database documentation for more information about fuzzy searching.

If your implementation has configured its database to enable fuzzy searching, you must also set up a *Feature Configuration* to indicate that feature is supported. Find the Feature Configuration record for the **General System Configuration** feature type. (It may need to be defined if it does not exist). Choose the option type **Fuzzy Searching Enabled** and define a value of **Y**.

Control Table Status

Some of the product’s control tables or configuration tables include a status.

In most cases, the status is simply Active / Inactive. The purpose of the status is for an implementation to be able to mark a record as no longer in use (Inactive) and prevent end users from using that control record when attempting to create a master or transaction record governed by the record.

- The status is only used on “type” records that a user may have to choose from when creating a new master or transaction record. The add dialogues that prompt for the “type” will only show types that are Active. For example, when adding a new Appeal record, Appeal Types that are inactive are not included in the dropdown list.
- Typically the product does not include server validation to prevent a master / transaction record from being added if the “type” record is inactive. This is only an issue when a record is not created by an online user, but rather when a record is created via an algorithm and the “type” is provided as configuration somewhere. For example, if a form rule creates an overpayment process as part of posting a form, the overpayment validation does not check that the overpayment process type is active. The expectation is that if an implementation has decided to make a record inactive, the onus is on the administrative users to checks other configuration that may reference that “type” to change the value. In this case, the admin users should update the form rule to an appropriate overpayment process type.
- A status will not typically be found on a configuration object that is implemented via configuration on other objects. For example, when defining an Asset Type, the valid Asset External ID types must be defined on the Asset Type. Asset External ID Type does not have a status because if an implementation determines that a given Asset External ID Type should no longer be used, it will simply not configure any Asset Types to refer to that external id type.
- Not all “type” objects have a status. This is a pattern that was established with the introduction of business object driven maintenance objects and user interface. Some older objects that were introduced earlier do not follow this pattern.
- Some “type” objects use a BO lifecycle to implement the Active / Inactive states. This was a pattern followed when the business object driven maintenance objects was first introduced. In subsequent releases the current standard of a simple Active / Inactive lookup based status was introduced.

There are some cases where an administrative object may have a more sophisticated lifecycle, for example, the form type status. This occurs when the administrative object itself has some business rules that are driven by a lifecycle, for example when child records need to be finalized and cross-validated before the “type” object may be used for master / transaction records.

Chapter 3

Defining Financial Transaction Options

This topics in this section explain the financial design of the system and describe how to set up the tables that control the financial impact of these transactions.

The Financial Big Picture

This section provides an overview of the relationship between an account and the various financial transactions that influence how much a taxpayer owes.

NOTE:
In the following sections, the term ‘bill’ is a reference to bills created using calculation rule based functionality. Topics related to rate-based bill functionality can be found under the heading of ‘Classic Bills’.

Assessments Overview

An assessment is a tax liability financial transaction associated with an obligation and represents the legal presentation of a tax liability to a taxpayer. There are different types of assessments, based on how and why the liability was created, and the assessment type can apply differentiated processing rules, such as different penalty and interest calculation rules or different collections rules.

Most revenue or filing periods only have one assessment created that captures any applicable penalty, interest, or fees. However, it is possible for additional assessments to be created within a revenue period.

Return and bill based tax types differ in how assessments get created and processed. The following sections discuss these concepts.

Return Based Taxes

Return based taxes are either expected or event based. Return based taxes represent taxpayers that have an expected obligation to file a Return for a filing period. Return based taxes can also be based on events such as taxpayers filing an excise tax. Income tax, withholding tax, and sales tax are examples of return based taxes.

For expected return-based tax types, the obligation is associated with a filing period as it tracks whether a filing obligation has been met for that period, and groups the assessments and financial transactions related to it. For most tax types, a filing period may be associated with a single type of obligation. For example, an active monthly Sales Tax filer has a single obligation for each month of the year. However, some tax types may require multiple filing obligations (of different types) over the same period of time. For example, a Withholding Tax type may have monthly Withholding Tax Returns as one obligation type, plus an annual Reconciliation Return and an Annual Taxpayer Withholding Detail statement as other obligation types.

Most return-based tax liability is created through the original self-assessment reported by a taxpayer on their tax return. There are two common models for self-assessment:

- Full Self-Assessment. This is common in the US. In full self-assessment, a taxpayer reports their income details on their tax return, calculates their tax per the tax return instructions, and files (and pays or expects a refund) by the due date of the filing period. If the tax return has errors or is changed during processing, the taxpayer receives an adjustment notice, but never receives an assessment notice.
- Partial Self-Assessment. This is common in taxes modeled after the British tax system. Like full self-assessment, the taxpayer reports their income tax details on a tax return. However, the tax return does not include instructions for calculating tax. When the tax return is processed, a Notice of Assessment is generated and sent to the taxpayer. The Notice of Assessment is a legal document informing the taxpayer of their tax obligation, and the due date for paying it.

The assessment established by the taxpayer's initial tax return is called the original assessment. If the taxpayer is later audited, and their income is increased (such that they now owe more tax), the additional tax established by the audit is part of an audit assessment. The incremental tax from the audit will be subject to different business rules, such as different rates and rules for calculating penalty and interest, and different collections notices or processes. It is also possible for a change to a tax return to result in a decrease in tax (and therefore a refund).

Bill Based Taxes

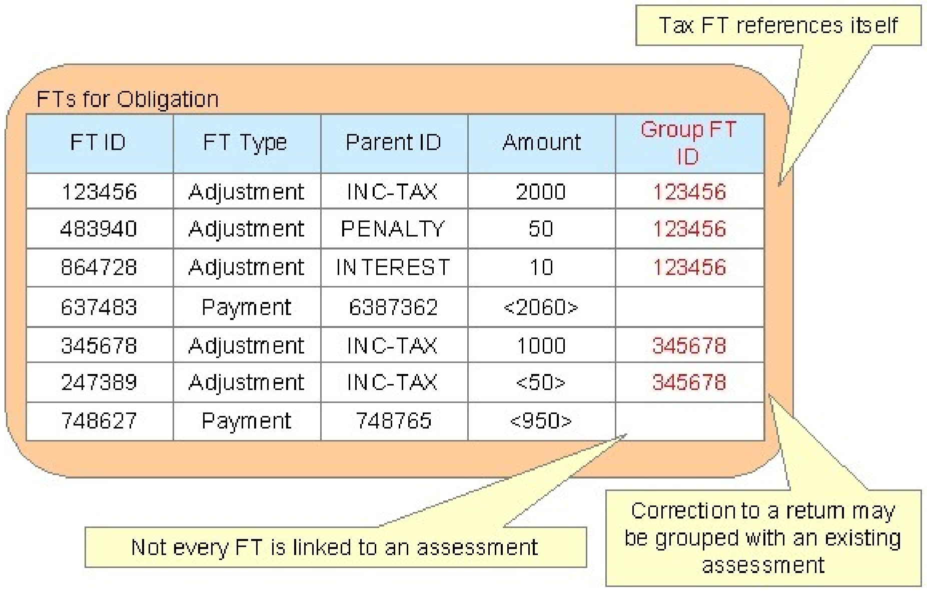
Bill based taxes are billed on a predefined schedule. Property tax, periodic licensing fees, and vehicle taxes are examples of bill-based taxes.

For bill-based taxes, the system has all of the necessary information to initiate assessments. The system creates assessments on a schedule. One or more assessments may exist for each billing period, depending on whether the bill is payable in installments and the applicable business rules for applying penalty and interest. If a taxpayer disputes a bill, the bill is cancelled and rebilled without creating incremental assessments.

Group Financial Transactions

As financial transactions are created for an obligation, often they are related to a specific liability such as an assessment, which is also a financial transaction (associated with an adjustment or a bill segment). For example, a payment may be made for a specific assessment. Penalties, interest and fees are often levied toward a specific assessment. Associating these subsequent financial transactions with the appropriate assessment FT ensures accurate penalty and interest calculations and accurate views of the balance of each assessment associated with an obligation.

The system provides a means for your implementation to group financial transactions (via the Group FT ID). When creating a financial transaction that should be associated with a given assessment, set the group FT id to the FT Id of the assessment. The following example shows the usage of the Group FT Id field.


FTs for Obligation

FT ID	FT Type	Parent ID	Amount	Group FT ID
123456	Adjustment	INC-TAX	2000	123456
483940	Adjustment	PENALTY	50	123456
864728	Adjustment	INTEREST	10	123456
637483	Payment	6387362	<2060>	
345678	Adjustment	INC-TAX	1000	345678
247389	Adjustment	INC-TAX	<50>	345678
748627	Payment	748765	<950>	

Tax FT references itself
Not every FT is linked to an assessment
Correction to a return may be grouped with an existing assessment

The original assessment FT (sometimes referred to as the "header FT") references its own FT id in the Group FT field. This allows system functionality to easily identify the assessment financial transaction.

Bills, Payments & Adjustments

The system maintains debt on each individual obligation for an account. An account's debt is the sum of its obligations' debt. The following points are high level descriptions of the various financial transaction supported

Adjustment Details

Adjustments are used for many different financial effects on a obligation, including tax assessments, penalties, interest, and fees. The following points highlight important concepts related to adjustments

- Over time, an obligation may have many adjustments.
- An adjustment has a related financial transaction. The financial transaction contains the financial effects of the adjustment on the obligation's debt and on the general ledger.
- Canceling an adjustment cancels the financial transaction. If the adjustment is eventually canceled, another financial transaction will be linked to the adjustment to reverse its financial effect.

Payment Details

Over time, many payments may be applied to an account's debt. The following points highlight important concepts related to payments:

- A payment amount is ultimately directed towards an obligation via a payment segment.
- If an account has multiple obligations that are in debt, the various payment segments created for the obligations may be grouped into a single Payment for the account.
- A payment segment has a related financial transaction. A financial transaction contains the financial effects of the segment on the obligation's balances and on the general ledger.
- Canceling a payment cancels the financial transaction. If the payment is eventually cancelled, another financial transaction will be linked to the related payment segment(s) to reverse their financial effect.

Bill Details

For bill based taxes, many bills are produced for an account over time. The following points highlight important concepts related to bills:

- Bills are produced for an account for its bill-based obligations. Over time, many bills may be produced for an account.
- A bill is created to levy charges for a single bill-based tax role and obligation.
- Bills contain value details. Bill value details contain information about the values on which a bill’s charges are based, such as asset valuation details.
- Bills contain calculation details. A bill calculation line contains information showing how the line was calculated and how it should be printed on the taxpayer's bill.
- Bills contain bill segments. A bill segment has a related financial transaction. A financial transaction contains the financial effects of the bill segment on the obligation's balance and on the general ledger.
- Canceling a bill cancels its financial transactions. When a bill is cancelled, other financial transactions will be linked to the bill for each bill segment to reverse the original financial transaction.

Effective Date

Financial transactions are assigned an effective date. This is the date that the debit or credit amount affects the obligation's balance and is important for penalty and interest calculations. The following are some examples of how the effective date is set:

- An assessment's effective date is typically the due date of the assessment. This is the date at which this amount impacts the obligation's balance with respect to penalty and interest calculations. If the taxpayer files a form earlier than the due date of the form, the form's due date is still used for the assessment's effective date because if the taxpayer does not pay, penalty and interest should only start from the due date, not the date the form was posted. The same is true for forms that are filed after the due date. The penalty and interest should calculate from the due date, therefore the assessment's effective date should be set to the due date.
- Penalty and interest transactions are assigned an appropriate effective date based on period of time that the penalty or interest charge covers. For charges that accrue after the period passes (such as interest charges), the effective date should be set to the end date of the covered period. For charges that accrue ahead of the period (such as charges that are "per month or any part thereof"), the effective date should be set to the start date of the covered period.
- A payment's effective date is typically the date the payment was considered received.

  FASTPATH: For more information about effective dates and payment dates, see *Payment Date and Effective Date for Payment Events*.

The system requires an effective date by default. However, it is possible to use a feature configuration setting to allow effective date to be optional for new debit adjustments. In this case the FT's "new charge" switch must be checked. This functionality assumes that a separate process is able to determine the proper effective date based on other information and can update the financial transaction at that time to populate the effective date properly. For example, if there is a miscellaneous charge that should only be considered effective after the taxpayer is informed of the charge, the adjustment can be created as a "new charge" with no effective date. The process that produces the correspondence to the taxpayer to inform them of the charge is responsible for updating the FT to set the effective date (and reset the "new charge" switch).

CAUTION: The base product's penalty and interest calculation logic will ignore any financial transactions that do not reference an effective date. It means that the amount will not factor into any calculation basis amount.

FASTPATH: Refer to *Configuring Effective Date as Optional* for configuration steps.

Penalty and Interest

Calculating penalty and interest (P&I) for delinquent assessments is an important element of a tax authority's business.

FASTPATH:
Refer to *The Big Picture of Penalty and Interest* for details about this functionality.

Current Amount versus Payoff Amount

A financial transaction contains two amount attributes: payoff amount and current amount. These attributes allow you to keep track separately of amounts related to how much the taxpayer owes.

- Current amount contains how much the taxpayer is asked to pay. In other words, this is the amount that affects the taxpayer's balance. It is what the taxpayer owes with respect to collections and is the amount that penalty and interest is calculated on.
- Payoff amount contains how much the taxpayer actually owes. This is the amount posted to the general ledger.

For most financial transactions, these values are the same. There are a small number of cases where this amount may be different. One example is a charitable contribution. If a taxpayer makes a charitable contribution when paying their tax liability, the payment or adjustment associated with the contribution credit should affect the payoff amount and the general ledger (because this amount is actually cash in hand). However, it should not affect the current amount because the contribution is not paying off any debt incurred and should not cause the taxpayer's balance to go into credit.

The topics in this section provide more information about these two fields.

What Controls What Gets Booked To Current And Payoff Amount?

As described in *Payment Details*, every payment segment has a sibling financial transaction. The financial transaction defines the payment segment's affect on current and payoff amounts due. The system populates these two fields as per the Financial Transaction Algorithm defined on the payment segment's payment segment type.

FASTPATH:
For more information, refer to *Payment - Current Balance versus Payoff Balance* and *Setting Up Payment Segment Types*.

As described in *Adjustment Details*, every adjustment has a sibling financial transaction. The financial transaction defines the adjustment's affect on current and payoff amounts due. The system populates these two fields as per the Financial Transaction Algorithm defined on the adjustment's adjustment type.

FASTPATH:
For more information, refer to *Adjustments - Current Balance versus Payoff Balance* and *Setting Up Adjustment Types*.

The following information applies only to Classic Bills. As described in *Classic Bill Details*, every bill segment has a sibling financial transaction. The financial transaction defines the bill segment's affect on current and payoff amounts due. The system populates these two fields as per the Financial Transaction Algorithm defined on the bill segment's bill segment type.

FASTPATH:

For more information, refer to *Billing - Current Balance versus Payoff Balance* and *Defining Bill Segment Types*.

GL Accounting Information

Be aware that if payoff amount is non-zero, the financial transaction has general ledger detail lines.

The effect on your GL is controlled by the financial transaction algorithm defined on your bill segment and payment segment types.

FASTPATH:
Refer to *The GL Interface* for how GL account information is interfaced to the general ledger.

Preventing Obligation Balances And The GL From Being Impacted Until Bill Completion

NOTE:
This topic relates to Classic Billing. The concepts described below apply only if your implementation uses classic billing. The business rules relating to financial transactions for rules based bills are governed by the algorithms configured on the bill type.

It's important to understand that when any type of financial transaction is **frozen**, the related obligation's balance is affected. For example:

- When a payment is **frozen**, the taxpayer's balance is reduced.
- When an adjustment is **frozen**, the taxpayer's balance is impacted.
- When a bill segment is **frozen**, the taxpayer's balance is increased (typically).

For payments, there is no issue. However, for bill based tax types, you may not want the taxpayer's balance to be impacted until the bill is completed. Consider the following:

- If a taxpayer has multiple obligations that should be presented on the same bill, it's possible for one of the obligations to have a bill segment that's in **error** and the other obligation's bill segment to be **frozen**.
- The **frozen** bill segment will impact the taxpayer's balance and the general ledger. This is because a financial transaction is marked for *interface* to the general ledger when it is frozen. This can be problematic if you have a long period between FT freeze and bill completion (you could impact the general ledger but not impact the taxpayer's balance).

If this is unacceptable, you can setup the system to not allow certain types of FT's to be frozen until the bill is completed. This means that neither the taxpayer's balance nor the general ledger will be impacted until bill completion time. To do this:

- Choose the **Freeze At Bill Completion** option on *Installation Options - Billing*.
- Determine if any of your *adjustment types* are ones that would be included on a bill-based tax bill. Select **Freeze At Bill Completion** for those that should not impact the taxpayer's balance or the general ledger until the next bill is completed. Select **Freeze At Will** for those that should impact the taxpayer's balance and the GL when they are frozen.

Please be aware of the following in respect of the **Freeze At Bill Completion** options:

- If you turn on **Freeze At Bill Completion** on *Installation Options - Billing*:
  - Users will not be allowed to freeze bill segments online.
  - Any background process created for batch billing should not freeze bill segments until all segments on a bill are error free.
  - Bill segments will exist in the **freezable** state until the bill is **completed**.

- If you turn on **Freeze At Bill Completion** for an adjustment type:
  - Users will not be allowed to freeze adjustments of this type online.
  - Background processes that create adjustments will not freeze this type of adjustment. Rather, the adjustments will be frozen when the next bill is completed.
  - Adjustments of this type will therefore exist in the **freezable** state until the next bill is **completed**.

NOTE:
Alerts highlight freezable FT's. Please be aware that messages appear in the *Financial Information Zone* to highlight the existence of freezable financial transactions.

Please be aware of the following in respect of the **Freeze At Will** options:

- If you turn on **Freeze At Will** on *Installation Options - Billing*:
  - Users will be allowed to freeze bill segments online.
  - Any background process created for batch billing should freeze bill segments when the individual segment is error-free.
  - Bill segments will exist in the **frozen** state regardless of whether the bill is completed.
  - The **frozen** bill segment's FT will be interfaced to the GL when the interface next runs.
  - All adjustment types must also be set to **Freeze At Will** (otherwise they wouldn't get frozen).
- If you turn on **Freeze At Will** for an adjustment type:
  - Users will be allowed to freeze adjustments of this type online.
  - Background processes that create adjustments will freeze this type of adjustment.
  - Adjustments of this type will exist in the **frozen** state prior to bill completion.
  - The **frozen** adjustment's FT will be interfaced to the GL when the interface next runs.

Forcing The Freeze Date To Be Used As The Accounting Date

Every financial transaction references an accounting date. The accounting date controls the accounting period to which the financial transaction is booked as described below:

- Every financial transaction references an accounting date and an obligation
- Every obligation references an obligation type
- Every obligation type references a GL division
- Every GL division references an *accounting calendar*
- The accounting calendar contains the cross reference between the accounting date specified on the financial transaction and the related accounting period in your general ledger

The accounting date is populated on financial transactions when they are initially generated. The following points describe the source of the accounting date:

- The user who creates or cancels a bill segment online defines the accounting date as part of the generation / cancel dialog (note, the current date defaults).
- The user who creates or cancels an adjustment online defines the accounting date as part of the generation / cancel dialog (note, the current date defaults).

- Payments are unusual in that their financial transaction is only created when they are frozen (rather than when the payment is first distributed amongst the account's obligations). At payment freeze time, the accounting date is set to the current date.

For payments, there is no issue because the accounting date is only populated on the financial transaction when a payment is frozen. However, for bill segments and adjustments, your business practice may dictate that the freeze date should be used as the accounting date rather than the original accounting date. Alternatively, your business practice may dictate that the accounting date that's originally stamped on bill segments / adjustments should be used (unless this associated period is closed at freeze time). It's really a question of the interpretation of the local accounting rules. After you've decided on your approach, populate the **Accounting Date Freeze Option** on *Installation Options - Billing* with one of the following values:

- Choose **Always change** if the accounting date on your financial transactions should be populated with the freeze date (i.e., the current date when the financial transaction is frozen).
- Choose **Change if period is closed** if the accounting date defined when the financial transaction is generated should be used (unless the associated accounting period is closed).

Please be aware of the following in respect of your choice:

- If you choose **Always change**:
  - When a user freezes a bill segment online, they will be prompted to supply an accounting date. The current date will default, but the user can override this value.
  - When a user freezes an adjustment online, they will be prompted to supply an accounting date. The current date will default, but the user can override this value.
  - Any batch billing process should use the current business date as the accounting date on bill segments that it freezes.
  - Also note, if you have chosen the **Freeze At Bill CompletionBill Segment Freeze Option** on the *installation record*, bill segments and certain types of adjustments are frozen when a bill is completed. This means that the accounting date on the related financial transactions will be set to the completion date (because the completion date is the freeze date with this setting). Refer to *Preventing Obligation Balances And The GL From Being Impacted Until Completion* for more information.
- If you choose **Change if period is closed**:
  - When a user freezes a bill segment online, they will only be prompted to supply an accounting date if the related accounting period is closed (because the accounting period closes after the bill segment is generated but before it's frozen). The current date will default, but the user can override this date.
  - When a user freezes an adjustment online, they will only be prompted to supply an accounting date if the related accounting period is closed (because the accounting period closes after the adjustment is generated but before it's frozen). The current date will default, but the user can override this date.

Obligation Type Controls Everything

The previous section illustrated three important concepts:

The true financial impact of the three financial events - bills, payments, adjustments - is at the obligation level, not at the account level. This means that bills and payments are meaningless on their own. It's the obligations' bill segments, payment segments and adjustments that affect how much a taxpayer owes.

- Every bill segment, payment segment, and adjustment has a related financial transaction. These financial transactions contain the double-sided journal entries that will be interfaced to your general ledger. They also contain the information defining how the taxpayer's debt is affected by the financial event (i.e., current amount and payoff amount).
- A single bill can contain many bill segments, each of which may have a different frequency.

You control the financial effects of the various financial events using a single field on the obligation. This field is called the **Obligation Type**. In this section, we describe many of the tables that must be set up before you can create an obligation type.

NOTE:
An obligation type controls numerous aspects of an obligation's behavior in addition to its financial behavior. The non-financial aspects are discussed in later chapters. It's only after you have set up all of the control tables in this manual that you'll be able to finally define your obligation types. Refer to *Setting Up Obligation Types* for more information.

CAUTION:
Take the time to define how you will record the various financial events in your general ledger before you attempt to set up these control tables. If you have simple accounting needs, this setup process will be straightforward. However, if you sell many services and use sophisticated accounting, this setup process will require careful analysis.

The Source Of GL Accounts On Financial Transactions

The following is a summary of the source of GL accounts on financial transactions:

- If a bill segment has a financial effect, the distribution code to debit comes from the distribution code on the obligation type; the distribution codes to credit come from the calculation lines used to calculate the bill segment. Calculation rules define the distribution codes to be used when creating bill calculation lines.
- Payment segments always have a financial effect; the distribution code to debit comes from the bank account on the tender source of the tender control of the tender, the distribution code to credit comes from the obligation type.
- For adjustments that have a financial effect, refer to *Adjustment Type Defines the GL Account* for more information.

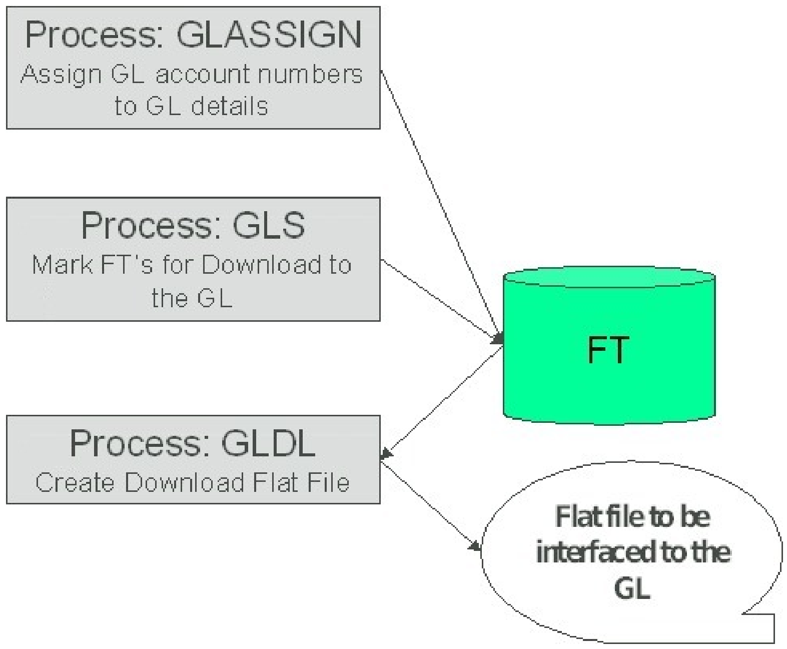
NOTE:
The information in this topic refers to bills created using calculation rule based functionality. For details of the source of FTs for rate-based bills, refer to topics under the heading of 'Classic Bills'.

The following table lists some examples of financial events, their standard accounting, and the source of distribution codes used to derive the GL accounts sent to your general ledger.

Financial event	***GL Accounting***	***Source Of Distribution Code***
Create a bill segment.	Debit: A/R	Obligation Type
	Credit: Revenue	Calculation Line / Rule
Create a payment segment for a normal obligation	Debit: Cash	Bank Account on the Tender Source of the Tender Control for the Payment Segment's Tender.
	Credit: A/R	Obligation Type
Canceling a payment	Debit: A/R	Obligation Type
	Credit: Cash	Bank Account specified by the user on the cancel tender page. Note that this defaults to the original tender's bank account.
Create an adjustment to levy a charge	Debit: A/R	Obligation Type
	Credit: Revenue	Adjustment Type
Create an adjustment with disbursement details	Debit: A/R	Obligation Type
	Credit: Revenue (multiple)	Adjustment calculation line details

The GL Interface

The following diagram illustrates the GL Interface.


Process: GLASSIGN
Assign GL account numbers to GL details
Process: GLS
Mark FT's for Download to the GL
Process: GLDL
Create Download Flat File
FT
Flat file to be interfaced to the GL

GLASSIGN - Assign GL Account Numbers To GL Details

The **GLASSIGN** process assigns GL account numbers to the GL details associated with financial transactions. GL account numbers are assigned as follows:

- Every GL detail references a distribution code.
- Every distribution code references a GL assignment algorithm. The base package algorithm simply uses the default GL account associated with the distribution code. However, you can construct your own algorithm(s) to assemble your GL account numbers in your desired fashion. Refer to *Setting Up Distribution Codes* for more information about the GL format algorithm.
- The **GLASSIGN** process simply calls each GL's details distribution code's GL assignment algorithm and updates the GL detail with the result (i.e., the GL account number). This GL account number is then used by the **GLDL** process when it creates the consolidated journal entry that's interfaced to the GL.

NOTE:
If incorrect GL account numbers get assigned to GL details... If you do not plug in the correct algorithm or your algorithm is wrong, you can correct the GL account numbers that are assigned to the GL details. How? Write a simple program that resets the GL account numbers on the erroneous GL details. Then run **GLASSIGN**. **GLASSIGN** will re-execute the distribution code GL assignment algorithm and refresh the account number accordingly.

GLS - Prepare FTs for Download

The **GLS** process creates FT download staging records for all FTs that are ready to be posted to the GL (the FT download staging records are stored on the FT/Process table). Each FT download staging record is marked with a batch process ID and run number.

- The batch process ID is the process responsible for creating the flat file that contains the consolidated journal entry that is interfaced to your general ledger. The batch process ID is defined on *Installation Options - Financial*. The system comes with a single GL download program (*GLDL - Create General Ledger Download Flat File*). The GL account numbers are provided by the GL account algorithms that are specified on the distribution codes. Refer to *Setting Up Distribution Codes* for more information about the GL account algorithm.

- The run number is the batch process ID's current run number.

This process also changes the status of the FT to **distributed**. You may not change the FT's GL details after this time.

GLDL - Create General Ledger Download Flat File

The **GLDL** process creates the flat file that contains the consolidated journal entry that is interfaced to your general ledger. One header and multiple detail records are created as described below. At the conclusion of the batch process, a validation is performed to compare debits against credits. If they are not equal, the process terminates with an error.

Header Record Layout

Field Name	*Format*	*Source/Value/Description*
REC_TYPE	A1	1 (1 is used for header records)
BATCH_CD	A8	The code for the batch process that created the file. (This value will always be 'GLDL'.)
BATCH_NBR	N10	The batch process will be run many times. This field indicates which of those runs produced a particular output file. This may be thought of as an instance identifier for the batch process.
BATCH_RERUN_NBR	N10	Any given instance of the GLDL batch process may be re-run at any time. If this instance has been re-run, this field will be populated with a number indicating how often.
EXTRACT_DTTM	A26	The date and time of the run that created a particular output file.
DETAIL_REC_CNT	N12	The number of details records
DETAIL_REC_TOTAL_DR	N13.2	This is the sum of FINANCIAL_AMOUNTs from all detail records that were greater than zero (debits).
DETAIL_REC_TOTAL_CR	N13.2	This is the sum of FINANCIAL_AMOUNTs from all detail records that were less than zero (credits).

Detail Record Layout for GLDL

NOTE:
A record is created for each unique occurrence of REC_TYPE, GL_DIVISION, CURRENCY_CD, GL_ACCOUNT and ACCT_PERIOD. If a given GL account has debit and credit FINANCIAL_AMOUNTs, two records will be created - one shows the debit amount, the other shows the credit amount.

Field Name	*Format*	*Source/Value/Description*
REC_TYPE	A1	2 (2 is used for detail records)
GL_DIVISION	A5	This is the GL division from the CI_FT record. It determines a) which Accounting Calendar will be used to calculate the Accounting Period (below) and b) the currency of the FT.
CURRENCY_CD	A3	The currency in which the Amount is denominated.
GL_ACCOUNT	A48	Contains the GL account number as supplied by the *distribution code's GL account algorithm*.

Field Name	***Format***	***Source/Value/Description***
ACCT_PERIOD	A6	An accounting period in the format YYYYPP where YYYY is the fiscal year and PP is the accounting period. This is derived from the GL detail's FT's accounting date and GL division.
FINANCIAL_AMOUNT	N13.2	This is the debit or credit amount. Debits are represented by positive numbers; credits are represented by negative numbers. If a given GL account has debit and credit entries, two records will be created - one shows the debit amount, the other shows the credit amount.
STAT_CODE	A8	This is the GL distribution code's statistic code (if any)
STAT_AMOUNT	N13.2	This is the statistical amount from the GL detail lines

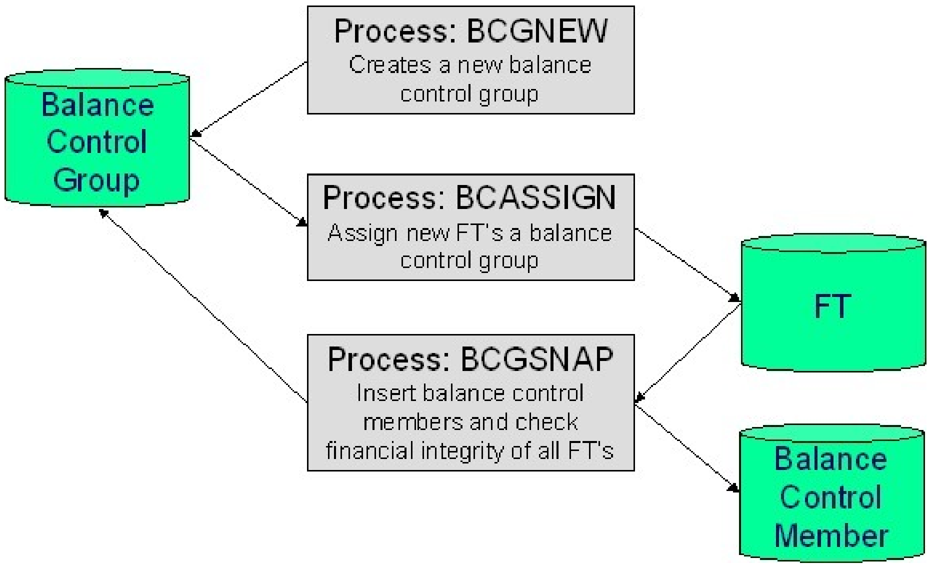
NOTE:
The GLASSIGN process updates an FT's GL details with the appropriate GL account number.

The Big Picture of Balance Control

The balance control processes are used to check the financial integrity of your system. The contents of this section describe how these processes work.

The Balance Control Background Processes

The following diagram illustrates the balance control background processes:


Process: BCGNEW
Creates a new balance control group
Balance Control Group
Process: BCASSIGN
Assign new FT's a balance control group
FT
Process: BCGSNAP
Insert balance control members and check financial integrity of all FT's
Balance Control Member

BCGNEW - Create A New Balance Control Group

This process creates a **Pending** balance control group if one doesn't already exist. You may wonder why an entire process is dedicated to such a trivial task. The reason is because the next process, BCASSIGN, is a multi-threaded (i.e., parallel) process and we only need one **Pending** balance control group regardless of the number of threads used to assign balance control ID's to financial transactions.

BCASSIGN - Assign New Financial Transactions A Balance Control Group

This multi-threaded (i.e., parallel) process assigns the **Pending** balance control group to new FTs whose freeze date/time is before the create date/time of the balance control record.

BCGSNAP - Insert BC Members And Check Financial Integrity Of All FTs

This process performs the following two functions:

- It summarizes new financial transactions under the current Pending balance control group as follows:
    - It creates a balance control member for every combination of **division**, **obligation type** and **FT Type** referenced on the financial transactions belonging to the balance control group.
    - It updates each balance control member with the following information:
        - The number of financial transactions (FTs)
        - The sum of the total amounts on the FTs in this balance control group.
        - The sum of the current amounts on the FTs in this balance control group.
        - The sum of the total amounts from all FTs (in this and all other balance control groups).
        - The sum of the current amounts from all FTs (in this and all other balance control groups).
    - It sets the status of the balance control group to **Complete**.
- It checks the integrity of the financial transactions in each historical Balance Control Group. It does this by summarizing EVERY financial transaction throughout time and determining if the sums are in sync with the values maintained on the balance control members. If integrity problems are detected, a detailed error message is displayed on the run control associated with the process.

If you opt to run the balance control processes on a nightly basis, you will find that the verification processing will take longer every night (because there are more financial transactions over time). In order to deal with this issue, the **BCGSNAP** process has a parameter that allows you to control which of the above functions is implemented (it's call VERIFY-ONLY-SW). In order to speed nightly processing, run this process with the switch set to "G" (this causes new financial transactions to be summarized under a new balance control). Then, once a week (or month), run this process with the switch set to "Y" (this checks the financial integrity of all financial transactions in the system).

If you run the balance control processes less frequently, you can set the VERIFY-ONLY-SW to "N" (this causes both of the above functions to execute).

Consider A Backup At This Point

We recommend you backup your database AFTER you run the above processes. Why? So that if a financial integrity problem is spotted in the future, you can compare the current database against the backup to see what changed.

Balance Control Information Is Available Online

Your internal auditors may be interested in the total number and amount of financial transactions that have been posted since a given point in time. You can use the *Balance Control* page to see this information.

General Financial Setup

The following sections describe maintenance transactions related to common financial objects.

Setting Up Divisions

There are two types of divisions referenced on an obligation type: a division and a GL division.

- General Ledger divisions typically comprise individual entities (e.g., companies) in your general ledger. You must set up a GL division for each such entity. The GL division's sole purpose in the system is to define the accounting period associated with financial transactions linked to obligations associated with the GL division (obligations are associated with GL divisions via their obligation type). The system cares about accounting periods in order to prevent a user from booking moneys to closed periods. It also uses accounting periods when it produces the flat file that contains the consolidated journal entry that is interfaced to your general ledger (refer to *The GL Interface* for more information).
- A division is used to separate business rules and can often be associated with a jurisdiction. The definition of a jurisdiction is a geographic-oriented entity with unique business rules. Another example of where you may use separate divisions is when your tax authority is responsible for other types of debt in the system, such as state university debts or child support debts. You must set up a division for each segmentation in your authority where business rules may differ.

Division is also referenced on obligation, account and location.

- The division on obligation is actually part of the obligation's obligation type. Because obligation type controls many business rules, all business rules that are on the obligation type can be thought of as being defined for a given jurisdiction and obligation type combination. Refer to *Configuring Obligation Types* for more information.
- The division on account when combined with the account's account type defines the jurisdiction that governs certain financial business rules (e.g., compliance review date). Refer to *Setting Up Account Types* for more information about these rules. The division on account can also play a part in the addressee of To Do entries associated with the account. To assign To Do entries to a role based on the division, simply link the To Do type to the division. Refer to *To Do Entries Reference A Role* for more information.
- The product was originally built with address functionality that is referred to now as "legacy" address functionality. If your implementation uses this functionality, the division on location defines the jurisdiction in which the location is located. This jurisdiction controls the types of obligations that can be associated with the location. Refer to *Defining Address Options* for more information about addresses.

NOTE:
Both division and GL division are stored on the financial transactions associated with an obligation. However, only GL division plays a part in *The GL Interface*. Refer to *Setting Up GL Divisions* for information about GL Divisions.

The topics in this section describe the pages used to maintain a division.

Division - Main

To define a division, select choose **Admin Menu** > **Division**.

Description of Page

Enter an easily recognizable **Division** and **Description** for the division.

Enter the **Work Calendar** that defines the days on which this division operates. This calendar is used to ensure system-calculated dates (e.g., bill due date, credit and collection event dates, etc.) fall on a workday.

Use the **To Do Roles** scroll area if an account's division influences the role assigned to the To Do entries associated with the account. In the collection, define the **To Do Role** to be assigned to entries of a given **To Do Type** that are associated with accounts that reference the **Division**. Refer to *Assigning A To Do Role* for more information.

NOTE:
Only To Do entries that are account-oriented take advantage of the roles defined for a division.

Where Used

Follow this link to view the tables that reference *CI_CIS_DIVISION* in the data dictionary schema viewer.

Division - Characteristics

You can define characteristics for a division. You may need these for reporting purposes or in your algorithms. Refer to *Characteristic Types* for more information.

Open **Admin Menu** > **Division** and navigate to the **Characteristics** tab to maintain a division's characteristics.

Description of Page

Select a **Characteristic Type** and **Characteristic Value** to be associated with this division. Indicate the Effective Date of the characteristic type and value.

NOTE:
You can only choose characteristic types defined as permissible on a division record. Refer to *Setting Up Characteristic Types & Their Values* for more information.

Setting Up Revenue Classes

Every obligation references an obligation type. Amongst other things, the obligation type defines an obligation's revenue class. The revenue class is used when the obligation's rate books revenue to different GL distribution codes based on the obligation's revenue class.

FASTPATH:
See *Rate Component - GL Distribution* for more information about how revenue class is used to determine the GL revenue accounts referenced on a bill.

To set up revenue classes, choose **Admin Menu** > **Revenue Class**.

Description of Page

Enter an easily recognizable **Revenue Class ID** and **Description** for every revenue class.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_REV_CL*.

Setting Up Debt Categories

Debt Categories are used to categorize financial transactions that are debits. Debt categories may also be assigned to credit financial transactions if the credit is associated with a given type of debit. Refer to *Debt Categories and their Priorities* for more information.

To set up a debt category, open **Admin Menu** > **Debt Category**.

The topics in this section describe the base-package zones that appear on the Debt Category portal.

Debt Category List

The Debt Category *List zone* lists every debt category. The following functions are available:

- Click the *broadcast* icon to open other zones that contain more information about the adjacent debt category.
- The standard actions of **Edit, Delete** and **Duplicate** are available for each debt category.

Click the **Add** link in the zone's title bar to add a new debt category.

Debt Category

The Debt Category zone contains display-only information about a debt category. This zone appears when a debt category has been broadcast from the Debt Category List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_DEBT_CAT*.

Setting Up Debt Category Priorities

Debt category priorities allow you to define a priority order of allocating credit financial transactions to debit financial transactions during credit allocation. Refer to *Debt Categories and their Priorities* for more information.

To set up a debt category, open **Admin Menu** > **Debt Category Priority**.

The topics in this section describe the base-package zones that appear on the Debt Category Priority portal.

Debt Category Priority List

The Debt Category Priority *List zone* lists every debt category priority. The following functions are available:

- Click the *broadcast* icon to open other zones that contain more information about the adjacent debt category priority.
- The standard actions of **Edit, Delete** and **Duplicate** are available for each debt category priority.

Click the **Add** link in the zone's title bar to add a new debt category priority.

Debt Category Priority

The Debt Category Priority zone contains display-only information about a debt category priority. This zone appears when a debt category priority has been broadcast from the Debt Category Priority List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_DEBT_CAT_PRIO*.

Setting Up Distribution Codes

Distribution codes simplify the process of generating accounting entries by defining valid combinations of chart of account field values.

FASTPATH:
Refer to *The Source Of GL Accounts On Financial Transactions* for more information about the accounting entries associated will bills, payments and adjustments.

To set up distribution codes, open **Admin Menu** > **Distribution Code**.

Description of Page

Enter a unique **Distribution Code** and **Description** for the distribution code.

If this distribution code is a holding account used for payables cash accounting, check the **Use For Cash Accounting** switch, and enter the actual payable **Cash Accounting Code**. The system will transfer the holding amount to this distribution code when the cash event occurs. For more information, refer to *Payables Cash Accounting*.

Define the **GL Account Algorithm** used by the system when it interfaces financial transactions that reference this distribution code to your general ledger (refer to *GLDL - Create General Ledger Download* for more information about the download process). The logic embedded in this algorithm constructs the actual GL account number. 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 constructs your general ledger account number. Click *here* to see the algorithm types available for this plug-in spot.

The **Write Off Controls** provides the ability to define configuration to write off debt associated with the distribution code by transferring the written off debt to a separate obligation. You would only do this if you wanted to the written off debt to remain visible in the account's balance and you don't want it to remain in the original obligation's balance.

- Define the **Division** and **Obligation Type** of the obligation to which bad debt associated with this distribution code should be transferred at write-off time. Note: only obligation types with a special role of **Write Off** may be selected.
- When the system transfers debt to the write-off obligation defined above, the distribution code defined on this **Division / Obligation Type** will be debited unless you turn on the **Override Switch**. When this switch is turned on, the system overrides the distribution code of the transfer to side of the adjustment with the distribution code associated with the debt being written off. You'd typically turn this switch on for liability distribution codes because you want to debit the original liability account when the debt is written off. Note: if this switch is on the system also overrides the characteristic type / value with the respective value associated with the debt that is being written off.

NOTE:
The write off algorithms provided in the base product (as part of overdue processing) do not transfer the debt to a separate **write off** obligation.

Use the **GL Account Details** scroll to define how the system constructs the GL account associated with the distribution code when it interfaces the financial transaction to your general ledger. For each distribution code, enter the following information:

- Enter the **Effective Date** of the following information.
- Define whether, on the **Effective Date**, the following information is **Active** or **Inactive**. The system will only use effective-dated information that is **Active**.
- Enter the **GL Account** that the general ledger uses to process financial transactions tagged with this distribution code.
- By default, the installation is configured to practice *fund accounting*. With this option activated, you can define the **Fund** associated with this distribution code. If your installation options indicate that fund accounting is **not practiced**, the field is not visible.
- Use the grid to define characteristic values for the **Distribution Code**. To modify a characteristic, simply move to a field and change its value. The following fields display:

- **Characteristic Type**. Indicate the type of characteristic.
- **Characteristic Value**. Indicate the value of the characteristic.

NOTE:
You can only choose characteristic types defined as permissible on the distribution code record. 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_GL_DIVISION*.

Configuring Effective Date as Optional

As described in *Effective Dates*, if your implementation would like the FT effective date to be optional on debit adjustments, you must set up a *Feature Configuration* option. Find the Feature Configuration record for the **Financial Transaction Options** feature type. (It may need to be defined if it does not exist).

Choose the **Option Types** with the description **Effective Date Optional for Debit Adjustments** and enter a value of **Y**.

Managing Adjustment Setup

An obligation's debt may be changed with an adjustment. Every adjustment must reference an adjustment type. The adjustment type contains a great deal of information that is defaulted onto the adjustment, including whether the adjustment amount is calculated. It also controls many business processes associated with the adjustment. The topics in this section describe how to design and set up adjustment types.

Adjustment Types Define Business Rules

An adjustment type contains the business rules that govern how its adjustments are managed by the system. The topics in this section describe how adjustment type controls the behavior of an adjustment.

Defines the Business Object

The adjustment user interface relies on a business object to define appropriate UI maps for display and maintenance. In addition, the business object may be used to define additional business rules. The adjustment's business object is defined on the adjustment type.

NOTE:
The adjustment maintenance object does not have a flexible lifecycle so the adjustment's business object may not be used to define a lifecycle or lifecycle rules.

The base product supplies several business objects based on broad categories of adjustments. For each broad category, the system provides two versions of the business object.

- One version defines the standard elements expected for business objects of that type and does not define any characteristics in the schema. These business objects may be used as is or may be extended for implementations that choose to "flatten" characteristic values.

- The second version defines a generic characteristic collection. The UI maps supplied with these business objects provide a generic user interface for defining characteristics where the user must choose the characteristic type and then the appropriate value.

FASTPATH: Refer to *Controls Adjustment Characteristics* for additional information about the configuration required for adjustment characteristics.

To view the details of the business objects provided by the base product, navigate using **Admin Menu** > **Business Object** and select the business objects for the **ADJUSTMENT** maintenance object.

If the base business objects do not provide all the desired functionality for capturing and displaying your adjustment information, you may extend the base business objects as desired or create new ones that are appropriate for your business.

Defaults the Adjustment Amount

The adjustment type may default the adjustment amount in one of the following ways:

- The adjustment type may specify a default amount. This would be used for those adjustment types that have a standard charge for all taxpayers that receive this adjustment.
- The adjustment type may specify a default adjustment amount algorithm. This would be used for those adjustment types that have a charge that varies based on other factors. For example, a non-sufficient funds charge may be based on a taxpayer's credit rating.

When an amount is defaulted onto a new adjustment it may be overridden by a user.

Generated Adjustments

You can use an algorithm to calculate an adjustment amount or to build details related to the adjustment amount. The following are some examples of where this may be used:

- Taking a base adjustment amount and applying a rate to add additional charges.
- Taking the adjustment amount and capturing or producing additional supporting details in the calculation details collection. For example, if a P&I adjustment should be "disbursed" into detailed buckets as per the disbursement of the related assessment adjustment, a generate algorithm may be used to produce the appropriate P&I calculation details. Refer to the base product algorithm type **C1-PIDISTRII** for an example.
- Receiving calculation details from a calling program and storing them with the adjustment.
- Receiving calculation details from a calling program and using them to generate general ledger details for the adjustment's financial transaction.

All generated adjustment types must be set up as follows:

- Set the adjustment type's Adjustment Amount Type to **Calculated Amount**
- Plug in an appropriate Generate algorithm on the adjustment type as per the business rules
- Configure an appropriate FT creation algorithm on the adjustment type. A typical reason for using generated adjustments is that additional details are required for the general ledger. The base product FT creation algorithms all include an option to use the calculation details as a source for the GL.

Adjustment Calculation Details

Generated adjustments are used to produce details related to the adjustment amount. These details may be provided by a calling program or may be determined using the appropriate business logic.

In general, the adjustment calculation lines are used to capture the details. But there are unusual points related to this logic:

- Algorithms that populate the calculation line details must also populate calculation "header" details, including the number of lines and the total amount. However, calculation "header" details are never instantiated in the database. The information populated in the calculation header is used by the adjustment logic to process the calculation lines. Refer to any base product Generate algorithm for an example of populating the calculation "header".
- Calculation lines include a switch called "create bill line." (Adjustment calculation lines are reusing bill calculation line functionality). In the adjustment logic, adjustment calculation lines are only instantiated if this switch is set to true. The reason that an algorithm may produce a calculation line that should not be stored is that this information is available in memory for subsequent processes, namely the FT creation algorithm, which produces the GL details.

There are times when the calling program has the detailed information to be stored in the adjustment calculation lines and in the FT GL details. The services provided in the product to add and freeze an adjustment do not include the full adjustment calculation details collection or the FT GL details collection as input. Rather, there is a special field called "custom common area" which may be used to pass information into the adjustment routines. The base product provides a Generate algorithm that accepts calculation details passed in the custom common area and returns calculation lines. Refer to the base product algorithm type **C1-ADJ-GN-CL** for more information.

Applying a Rate

One use of the adjustment generation plug-in is to call rate application. The base product supplies an algorithm type **ADJG-RT** that enables you to call a rate application, passing in the base adjustment amount and receiving calculated details from the rate.

The adjustment type's generate adjustment algorithm controls which rate is applied to the base amount. A user supplied calculation date controls which version of the rate is used. The user may supply the base amount or it may be defaulted from the adjustment type and possibly overridden by the user prior to calculating the adjustment amount.

Controls Which Balance(s) Are Affected

The adjustment type's financial transaction (FT) algorithm controls how payoff balance and current balance are affected by the adjustment amount.

Controls Adjustment Characteristics

The types of characteristics that are valid for an adjustment are configured on the adjustment type. In addition it is possible to indicate which types of characteristics are required and whether any values should be defaulted.

Note that the underlying adjustment validation rules are using the adjustment type's adjustment characteristic configuration to validate adjustment characteristics. As a result, careful consideration must be paid to this characteristic configuration with respect to defining business objects for the adjustment:

- If the adjustment business object referenced by the adjustment type does not define any schema elements that map to the characteristic collection, then required adjustment characteristic types should not be defined on the adjustment type. Otherwise the users will encounter validation errors.
- If the adjustment business object referenced by the adjustment type defines "flattened" schema elements that map to the characteristic collection, then the same characteristic type must also be defined on the adjustment type as valid.
- If the business object defines a generic characteristic collection in its schema (as per the base business objects that include the characteristics), the valid characteristic types are taken from this configuration.

Defines the GL Account Affected by the Adjustment

Most adjustments affect the general ledger (GL) in some way. The following points describe the source of these GL accounts.

- For many adjustments there is a single accounting entry generated:
- One side of the accounting entry is taken from the distribution code on the obligation type of the obligation affected by the adjustment. For example, if you are adjusting the payoff balance on a normal obligation, the A/R account is constructed from the distribution code on the obligation's obligation type.
- The other side of the accounting entry is taken from the distribution code on the adjustment's adjustment type.
- For transfer adjustments (i.e., adjustments used to transfer moneys between two obligations), there are two accounting entries generated - one for the "from" side and one for the "to" side. Each adjustment carries its own set of balanced GL accounting details.
- For each adjustment, one side of the entry is taken from the distribution code on the obligation type of the obligation affected by the adjustment
- The other sides of both accounting entries have the same GL account. This account should be the intermediate clearing GL account that is to be used for the transfer. The source of this clearing GL account is the distribution code on the adjustment type used to transfer the funds.
- For generated adjustments, the accounting entry may include several GL details:
- One side of the entry is taken from the distribution code on the obligation type of the obligation affected by the adjustment.
- The other side of the entry depends on configuration on the adjustment financial transaction algorithm. If it is configured to use the Calculation Lines as the source, the distribution codes are taken from the calculation lines. If it is configured to use the adjustment type as the distribution code source, the other side of the accounting entry is taken from the distribution code on the adjustment type.

NOTE:
Not all adjustments affect the GL. As a general rule of thumb, only those adjustments that affect an obligation's payoff balance affect the GL.

Controls the Interface to A/P and Income Statement Reporting

If the adjustment type is associated with a payment of money to a taxpayer (e.g. a refund) that is managed by the accounts payable system, the adjustment type indicates such with a reference to an A/P request type.

When an adjustment that references an A/P request type is **frozen**, an A/P download request record is created. This record is the interface request to ask the A/P system to issue a check. This interface record is marked with a batch process ID and run number.

- The batch process ID is the process responsible for creating the flat file that contains check request that is interfaced to your account's payable system. The batch process ID is defined on *Installation Options - Financial*. The base package is supplied with a skeletal background process (referred to by the process ID of **APDL**) that must be populated with logic to format the records in the format compatible with your accounts payable system.
- The run number is the batch process ID's current run number.

FASTPATH: Refer to *Accounts Payable Check Request* for more information.

If the resultant check needs to be reported for income tax purposes under a specific income statement category, the category is also specified on the adjustment type. The income statement category is in turn interfaced to the A/P system. (The system does not manage income statement reporting).

Controls Information Printed On the Bill

If the adjustment is one that appears on a taxpayer's bill, the verbiage is specified on the adjustment type (and may NOT be overridden on the adjustment).

Controls if the Adjustment Requires Approval

FASTPATH:
Refer to *The Big Picture of Adjustment Approvals* for more information.

May Control the Adjustment Information

The adjustment information displayed throughout the system is controlled by a plug-in.

The system first looks to see if the adjustment type references an adjustment business object and if the business object defines an information plug-in. If a BO is not provided or if that BO does not define an information algorithm, the system looks for an information algorithm plugged into the adjustment maintenance object.

If no plug-ins are found on the BO or the MO, the system looks for a plug-in algorithm on the *Adjustment Type*.

If such an algorithm is not plugged-in on the Adjustment Type, the system looks for a corresponding algorithm on the *installation record*.

If you prefer different formatting logic, your implementation should provide a plug-in at one of the above plug-in spots, as appropriate for your business rules.

Setting Up Adjustment Types

The topics in this section describe how to set up adjustment types.

NOTE:
When a new adjustment type is added. When you introduce a new adjustment type, you must update one or more adjustment profiles with the new adjustment type. This is because adjustment profiles define the adjustment types that may be levied on obligations (adjustment profiles are defined on obligation types). If you don't put the adjustment type on an adjustment profile, the adjustment type can't be used on any adjustment.

Adjustment Type - Main

To set up adjustment types, open **Admin Menu** > **Adjustment Type**.

Description of Page

Enter a unique **Adjustment Type** and **Description** for the adjustment type.

If an *adjustment type extension* exists for the adjustment type, a link to the extension record is displayed next to the Adjustment Type ID.

The **Adjustment Amount Type** indicates whether or not the adjustment amount or details to support the adjustment amount is generated or not. Select **Calculated Amount** when you want to use a generate algorithm to generate the adjustment amount or generate calculation line details to support the adjustment amount, otherwise select **Non-Calculated Amount**. Refer to *Generated Adjustments* for more information about generated adjustments.

Enter the **Distribution Code** that references the GL account associated with the adjustment. For example, if this adjustment type is used to levy a charge for a bad check, the distribution code would reference the revenue account to which you associate such revenue. Note, the offsetting distribution code is kept on the obligation type.

NOTE:
Distribution Code for Generated Adjustments. Depending on the algorithm used for the *generated adjustment*, the distribution code may come from the adjustment type or the calculation lines of the adjustment. If the adjustment's FT creation algorithm gets the distribution code from the calculation lines, you do not need to specify a distribution code on the adjustment type.

FASTPATH:
For more information about the source of the distribution codes on financial transactions, see *The Source Of GL Accounts On Financial Transactions*.

Use **Adjustment Type Category** to assign a broad category to the adjustment type. Valid values are **Assessment**, **Credit**, **Manual Adjustment**, **Manual P&I**, **Penalty & Interest**, **Waiver**, **Write Off**.

FASTPATH: **Used by P&I.** The values of **Assessment**, **Penalty & Interest** and **Waiver** play an important role in the base product P&I Calculation algorithm. Refer to *Apply P&I Rules for Each Time Period* for more details.

NOTE: **Multi-Adjustment Transactions** The values of **Manual Adjustment** and **Manual P&I** control whether the adjustments of this type may be created on the Manual Adjustment and Manual P&I transactions respectively.

NOTE: The values for this field are customizable using the Lookup table. This field name is ADJ_TYPE_CAT_FLG.

Define the **Debt Category** to assign to the financial transaction for debit adjustments (adjustments with amounts greater than or equal to 0) created for this adjustment type.

NOTE: **Required for base algorithms**. The base P&I calculation algorithm and the base *Determine Detailed Balance* algorithm require that every debit adjustment reference a debt category.

If this adjustment type is for a certain type of credit adjustment and it has special rules for how credits are applied in the base Determine Detailed Balance algorithm, define the appropriate **Debt Category Priority**. Refer to *Debt Categories and their Priorities* for more information.

Enter the **Currency Code** for adjustments of this type.

Turn on **Sync. Current Amount** if adjustments of this type exist to force an obligation's current balance to equal its payoff balance. These types of adjustments are issued before an obligation's funds are transferred to a write-off obligation. If this switch is on, choose an **Adjustment Fin Algorithm** that does not impact payoff balance or the GL, but does affect the obligation's current balance (refer to *ADJT-CA* for an example of such an algorithm).

Enter a **Default Amount** if an amount should be *defaulted* onto adjustments of this type.

FASTPATH: For more information about current and payoff amounts, refer to *Current Amount versus Payoff Amount*.

If the A/P Adjustment should be recorded in respect of the taxpayer's income statement amounts, indicate the **Income Statement**. The values of this field are **Interest** and **Miscellaneous**. This type of adjustment would also have an **A/P Request Type Code** selected, as income statement reporting is handled in A/P.

Turn on **Print By Default** if information about adjustments of this type should print on the account's next bill.

Choose an **A/P Request Type** if this adjustment is interfaced to accounts payable (i.e., it's used to send a refund check to a taxpayer). Refer to *A/P Check Request* for more information.

The **Adjustment Freeze Option** defines when adjustments can be frozen and therefore when an obligation's balance and the general ledger are affected by an adjustment. Refer to *Preventing Obligation Balances And The GL From Being Impacted Until Bill Completion* to understand the significance of this option. Also note, if the *installation option's* Bill Segment Freeze Option is **Freeze At Will**, this field is defaulted to **Freeze At Will** and cannot be changed.

CAUTION: Adjustment types for adjustments created during bill completion (e.g., by a bill completion algorithm) must have their adjustment freeze option set to *Freeze At Will*. Otherwise (i.e., if the option is *Freeze At Bill Completion*) they will not be frozen until a subsequent bill is completed.

NOTE: The adjustment freeze option field is only visible if your implementation has not *turned off* the **BI — Billing (Classic)** module. This information is used only for classic billing functionality.

Set **Allow Manual Creation** to **Allowed** if you want to allow users to manually create adjustments of this type from the adjustment page, otherwise set it to **Not Allowed**.

Set **Allow Manual Cancellation** to **Allowed** if you want to allow users to manually cancel adjustments of this type from the adjustment page, otherwise set it to **Not Allowed**.

If adjustments of this type require approval, define an **Approval Profile**. For more information, refer to *The Big Picture of Adjustment Approvals*.

Enter the verbiage to appear on the printed bill in **Description on Bill**.

Use an **Adjustment Business Object** to define a *BO* that may govern the display and maintenance UI maps for adjustments of this type as well as additional rules related to adjustments of this type.

Use the characteristics collection to define a **Characteristic Type** and **Characteristic Value** common to all adjustments of this type. These can be used for reporting purposes or in your algorithms.

Adjustment Type - Adjustment Characteristics

To define characteristics that can be defined for adjustments of a given type, open **Admin Menu** > **Adjustment Type** and navigate to the **Adjustment Characteristics** tab.

Description of Page

Use the **Adjustment Characteristics** collection to define characteristics that can be defined for adjustments of a given type. Turn on the **Required** switch if the **Characteristic Type** must be defined on adjustments of a given type. Enter a **Characteristic Value** to use as the default for a given **Characteristic Type** when the **Default** switch is turned on. Use **Sequence** to control the order in which characteristics are defaulted.

Adjustment Type - Algorithms

Description of Page

The grid contains **Algorithms** that control important adjustment functions. If you haven't already done so, you must *set up the appropriate algorithms* in your system. 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
Adjustment Cancellation	Optional	When an adjustment is canceled an algorithm of this type may be called to do additional work. Refer to *The Lifecycle Of An Adjustment* for more information about canceling an adjustment. Click *here* to see the algorithm types available for this system event.
Adjustment Freeze	Optional	When an adjustment is frozen an algorithm of this type may be called to do additional work. Refer to *The Lifecycle Of An Adjustment* for more information about freezing an adjustment. Click *here* to see the algorithm types available for this system event.
Adjustment Information	Optional	We use the term "Adjustment information" to describe the basic information that appears throughout the system to describe an adjustment. The data that appears in "Adjustment information" may be constructed using an algorithm plugged in here. Refer to *Adjustment Information* for more information on when this algorithm is used. Click *here* to see the algorithm types available for this system event.
Adj. Financial Transaction	Required	Algorithms of this type are used to construct the actual financial transaction associated with the adjustment. The financial transaction controls the adjustment's affect on the obligation's payoff and current balances. It also constructs the information that is eventually interfaced to your general ledger. Click *here* to see the algorithm types available for this system event.
Default Adjustment Amount	Optional	Algorithms of this type are used to default the adjustment amount. Refer to *Default the Adjustment Amount* for more information. Click *here* to see the algorithm types available for this system event.
Determine Obligation	Optional	Algorithms of this type are used to find an obligation for which the adjustment can be posted. This plug-in is used particularly during adjustment upload when a staging record does not identify the obligation ID. Refer to *Interfacing Adjustments From External Sources* for more information. Click *here* to see the algorithm types available for this system event.
Resolve Suspense	Optional	Algorithms of this type are used to automatically resolve adjustments that are in suspense. Refer to *Suspense Adjustments* for more information Click *here* to see the algorithm types available for this system event.
Generate Adjustment	Optional	Algorithms of this type are used to generate the adjustment amount or generate details to support the adjustment amount if an adjustment type indicates that the adjustment amount is calculated. Refer to *Generated Adjustment* for more information. Click *here* to see the algorithm types available for this system event.

Validate Adjustment	Optional	Algorithms of this type are used to validate information for the adjustment after it is generated. 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_ADJ_TYPE*.

Setting Up Adjustment Type Profiles

Adjustment type profiles categorize your adjustment types into logical groups. When you link a profile to an obligation type, you limit the type of adjustments to be linked to the obligation type's obligations. The creation of adjustment profiles and their linkage to obligation types prevents inappropriate adjustments from being linked to your obligations. More than one adjustment type profile may be linked to an obligation type.

For example, you can create an adjustment type profile called **Miscellaneous Fees** and link to it the miscellaneous fee adjustment types. Then, you would link this profile to those obligation types that are allowed to levy such fees.

NOTE:
Bottom line. An adjustment can only be linked to an obligation if its adjustment type is part of an adjustment type profile that is valid for the obligation's obligation type. If an adjustment type is not linked to a profile, it could never be levied.

To set up adjustment type profiles, open **Admin Menu** > **Adjustment Type Profile**.

Description of Page

Enter a unique **Adjustment Type Profile** and **Description** for the adjustment type profile.

Indicate the **Adjustment Types** that are part of the profile.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_ADJ_TYP_PROF*.

Setting Up Adjustment Type Extensions

Adjustment Type Extensions let you maintain additional information related to an adjustment type. The information you maintain is controlled by the business objects that your implementation defines based on your business needs.

The base product supplies an adjustment type extension business object that supports mapping between assessment distribution codes and P&I distribution codes. Refer to the business object **C1-PIDisbursement** for more information.

To set up an Adjustment Type Extension, open **Admin Menu** > **Adjustment Type Extension**.

The topics in this section describe the base-package zones that appear on the Adjustment Type Extension portal.

Adjustment Type Extension List

The Adjustment Type Extension *List zone* lists every Adjustment Type Extension. The following functions are available:

Click the *broadcast* icon to open other zones that contain more information about the adjacent Adjustment Type Extension.

The standard actions of **Edit, Delete** and **Duplicate** are available for each Adjustment Type Extension.

Click the **Add** link in the zone's title bar to add a new Adjustment Type Extension.

Adjustment Type Extension

The Adjustment Type Extension zone contains display-only information about an Adjustment Type Extension. This zone appears when an Adjustment Type Extension has been broadcast from the Adjustment Type Extension List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_ADJ_TYPE_EXT*.

Setting Up Adjustment Creation Reasons

Adjustment creation reasons are defined using an extendable lookup. To view or create adjustment creation reasons:

- Open **Admin Menu** > **Extended Lookup**.
- Search for and select the **Adjustment Creation Reason** extended lookup business object.
- The list of existing adjustment creation reasons are displayed in a standard *List zone*.
- Choose an existing adjustment creation reason to view, edit, delete or duplicate.
- Use the **Add** link in the zone header to create a new adjustment creation reason.

Setting Up Adjustment Cancellation Reasons

Open **Admin Menu** > **Adjustment Cancel Reason** to define your adjustment cancellation reason codes.

Description of Page

Enter an easily recognizable **Cancel Reason** and **Description** for each adjustment cancellation reason.

A/P Check Request

Adjustments whose adjustment type is marked with an A/P check request code are interfaced to your A/P system. Your A/P system then issues the checks.

FASTPATH: Refer to *Controls The Interface To A/P and Income Statement Reporting* for more information about the accounts payable interface.

You must set up at least one A/P check request code if you want A/P to issue checks.

To set up A/P check request types, open **Admin Menu** > **A/P Request Type**.

Description of Page

Enter an easily recognizable **A/P Request Type** for the accounts payable request type.

Use **Due Days** to define when the check is cut. The cut date is equal to the adjustment date plus due days.

Select a **Payment Method**. The current system option is **System Check**

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_APREQ_TYPE*.

The Big Picture Of Adjustment Approval

Some implementations require adjustments to be approved by one or more managers before they impact an obligation's balance and the general ledger. For example, an adjustment used to refund a credit balance may require managerial approval before the refund is sent to the taxpayer. The topics in this section describe how to set up the system to support the approval of adjustments.

Approval Is Controlled By Approval Profiles

An approval profile contains the rules that define if and how an adjustment is approved. If an adjustment type does not reference an approval profile, the related adjustments do not require third-party approval before they impact an obligation's debt. If an adjustment type references an approval profile, the approval profile's approval hierarchy defines if the adjustment requires approval and who the authorized approvers are. For example, an approval profile can be configured with the following approval hierarchy:

- Adjustments < $0 require approval by the "credit approvers role"
- Adjustments >= $0 and <= $10 do not require approval
- Adjustments > $10 and <= $100 require the approval of a user that belongs to the "level 1 approvers role"
- Adjustments > $100 requires the approval of a user that belongs to the "level 2 approvers role"

The approval profile includes a setting to indicate if the adjustment requires **Single** approval or **Multiple** levels of approval. If there is an adjustment for $200 and the approval profile is configured for **Single** approval, only one approval is needed by the "level 2 approvers role". If instead the approval profile for this adjustment type indicate **Multiple** levels of approval: first a user that belongs to the "level 1 approvers role" must approve the adjustment; afterwards, the adjustment must be approved by a user that belongs to the "level 2 approvers role".

NOTE:
Transfer adjustments. The term "transfer adjustment" refers to two adjustments that are used to transfer moneys between two obligations. The adjustment with the positive amount is considered to be the debit adjustment; the other adjustment is considered the credit adjustment. When a transfer adjustment requires approval, only one of the adjustments needs to be approved. You control whether the debit side or the credit side of a transfer adjustment is used to control the approval process when you set up the approval profile.

NOTE:
Separation of Duties. The base product logic includes validation that prevents a user from approving an adjustment if that user was the one that created the record or if that user already approved a previous approval "level" (for approvals that define **Multiple** levels).

Approval Profiles Can Be Linked To Multiple Adjustment Types

Approval hierarchies are frequently the same for many adjustment types. The system allows an approval profile to be linked to multiple adjustment types to simplify the definition and maintenance of the rules over time.

Adjustments Created In Batch Are Not Approved

The system assumes that no approval is necessary for adjustments created by batch processes even those whose adjustment type references an approval profile.

Approval Inserts A Step Into An Adjustment's Lifecycle

The Lifecycle Of An Adjustment explains how an adjustment is transitioned from the **Freezable** state to the **Frozen** state when it should impact the general ledger and the obligation's balance. If an adjustment's adjustment type references an approval profile, the user cannot freeze the adjustment directly. Rather, the user must submit the adjustment for approval when it's ready and only when the last applicable approver approves the adjustment will it become **Frozen**.

NOTE:
Freeze during bill completion. You can configure the system to only freeze certain types of adjustments when the next bill is completed for the adjustment's account. When the last approver approves such adjustments, they remain in the **Freezable** state. When the next bill is completed for the account, these adjustment become **Frozen**. Such adjustments that have not been approved at the time of bill completion will remain in the **Freezable** state. Refer to *Preventing Obligation Balances And The GL From Being Impacted Until Bill Completion* for more information.

Approval Requests Manage And Audit The Approval Process

Users submit an adjustment for approval using a dedicated button on the *Adjustment* page. When an adjustment is submitted for approval, the system creates an "approval request". The approval request determines if the adjustment requires approval and, if so, the list of approvers. If the adjustment does not require approval, the approval request is updated to indicate such and the adjustment is **Frozen** immediately (if freezing is allowed prior to bill completion). If the adjustment requires approval, the approval request's state becomes **Approval In Progress** and the approver(s) are notified.

NOTE:
Approval submission logic is customizable. The previous paragraph describes how the base-package works when an adjustment is submitted for approval. This logic resides in an algorithm that's plugged in on the **C1-AdjustmentApprovalProfile** business object in the **Determine Approval Requirements** system event. Your implementation can change this logic by developing a new algorithm and plugging it into this business object. If your logic is meant to supersede the base-package algorithm, remember to inactivate the base-package algorithm by adding an appropriate inactivation option to this business object.

To Do Entries Are Created To Notify Approvers

When an approval request detects an adjustment requires approval, it notifies the first approver by creating a To Do entry. The To Do entry is created using the To Do type and To Do role defined on the approval profile. All users who belong to the approving To Do role can see the entry. When a user drills down on an adjustment approval To Do entry, the *Adjustment - Main* page is opened. This page includes information about the adjustment and the approval history of the adjustment. This portal is also where the user approves or rejects the adjustment.

When a user in the To Do role approves an adjustment, the To Do entry is **Completed** and the approval request's audit log is updated. If there are no higher levels of approval required, the adjustment is **Frozen** (if freezing is allowed prior to bill completion) and the approval request is moved to the **Approved** state. If there are higher levels of approval required, a new To Do entry is created to the next To Do role in the approval hierarchy.

NOTE:
To Do entries can create email. A To Do entry can be configured to create an email message for every user in the To Do role to inform the user(s) of new adjustments requiring their attention. Refer to *To Do Entries May Be Routed Out Of The System* for the details.

Monitoring and Escalating Approval Requests

The base-package is supplied with an algorithm that your implementation can use to monitor approval requests that have been waiting too long for approval. This algorithm can complete the current To Do entry and create a new one for a different role when the time out threshold defined on the algorithm's parameters is exceeded. If you've configured the system to send email for approval, this algorithm can also send x reminder emails (where x is defined on the algorithm's parameters) before the approval request is escalated to the new To Do role. Refer to *C1-APR-TMOUT* for more information about this algorithm. If you plan to enable this functionality, plug-in your configured algorithm on the **Approval In Progress** state on the **C1-AdjustmentApprovalRequest** business object.

Rejecting Deletes The Adjustment

When an adjustment is being approved, anyone with access to the adjustment can reject it by using the *Adjustments - Approval* portal. Users other than the current approver are allowed to reject an adjustment to allow an "in process" an adjustment to be withdrawn.

When an adjustment is rejected, the following takes place:

- The user is prompted for a reject reason.
- The approval request's audit log is updated with the reject reason and the approval request is moved to the **Rejected** state.
- The adjustment is deleted.

Designing Your Approval Profiles

The following points describe a recommended design process:

- Create logical groups of adjustment types where each group has the same monetary hierarchy and approvers. An approval profile will be required for each of these groups.
- The number of To Do types (if any) that need to be created is dependent on how the adjustment approval To Do entries should be organized on To Do lists. For example, if all approval request To Do entries can appear in the same To Do list, you can use the base-package adjustment approval To Do type. However, if your organization prefers each approval profile's To Do entries to appear in a distinct To Do list, a separate To Do type will be needed for each list. Note that the base-package is supplied with a To Do type called *C1-ADAPP* that should be used as the basis for any new approval request To Do type.
- The number of To Do roles is dependent on who approves your adjustments. At a minimum, you will require a separate To Do role for each level in your approval profiles. Remember that every user in a To Do role will see its entries (and receive email if you've configured the system to do such).
- Refer to *Monitoring and Escalating Approval Requests* for how to configure the system to escalate approval requests that have been waiting too long.
- If your implementation requires email notification when an adjustment requires approval, the following setup is required:
  - Set up an outbound message type, external system, and XAI sender. Refer to *To Do Entries May Be Routed Out Of The System* for the details.
  - Every To Do type referenced on your approval profiles should be configured as follows:
    - Define the *F1-TDEER* batch process as the To Do type's routing process
    - Set up an algorithm that references the *C1-ADJAREQEM* algorithm type and plug it in the External Routing system event.

Exploring Adjustment Approval Data Relationships

Use the following links to open the application viewer where you can explore the physical tables and data relationships behind the approval functionality:

- Click *C1-APPR PROF* to view the approval profile maintenance object's tables.
- Click *C1-APPR REQ* to view the approval request maintenance object's tables.

Implementing Other Approval Paradigms

The above sections describe how the base-package adjustment approval process works. Because adjustment approval has been implemented using the **C1-AdjustmentApprovalProfile** and the **C1-AdjustmentApprovalRequest** business objects, your implementation can add additional business rules and change the approval user interface as required. Alternatively, if your implementation has a radically different approval process, you can create different business objects with their own business rules.

Setting Up Approval Profiles

Approval profiles contain the rules that control how adjustments are approved. To set up an approval profile, open **Admin Menu** > **Approval Profile**.

FASTPATH:
Refer to *The Big Picture Of Adjustment Approval* for a detailed description of how approval profiles govern the adjustment approval process.

The topics in this section describe the base-package zones that appear on the Approval Profile portal.

Approval Profile List

The Approval Profile *List zone* lists every approval profile. The following functions are available:

- Click the *broadcast* icon to open other zones that contain more information about the adjacent approval profile.
- The standard actions of **Edit, Delete** and **Duplicate** are available for each approval profile.

Click the **Add** link in the zone's title bar to add a new approval profile.

Actions

This is a standard actions zone. The **Edit, Delete** and **Duplicate** actions are available.

Approval Profile

The Approval Profile zone contains display-only information about an approval profile. This zone appears when an approval profile has been broadcast from the Approval Profile List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Approval Profile's Adjustment Types

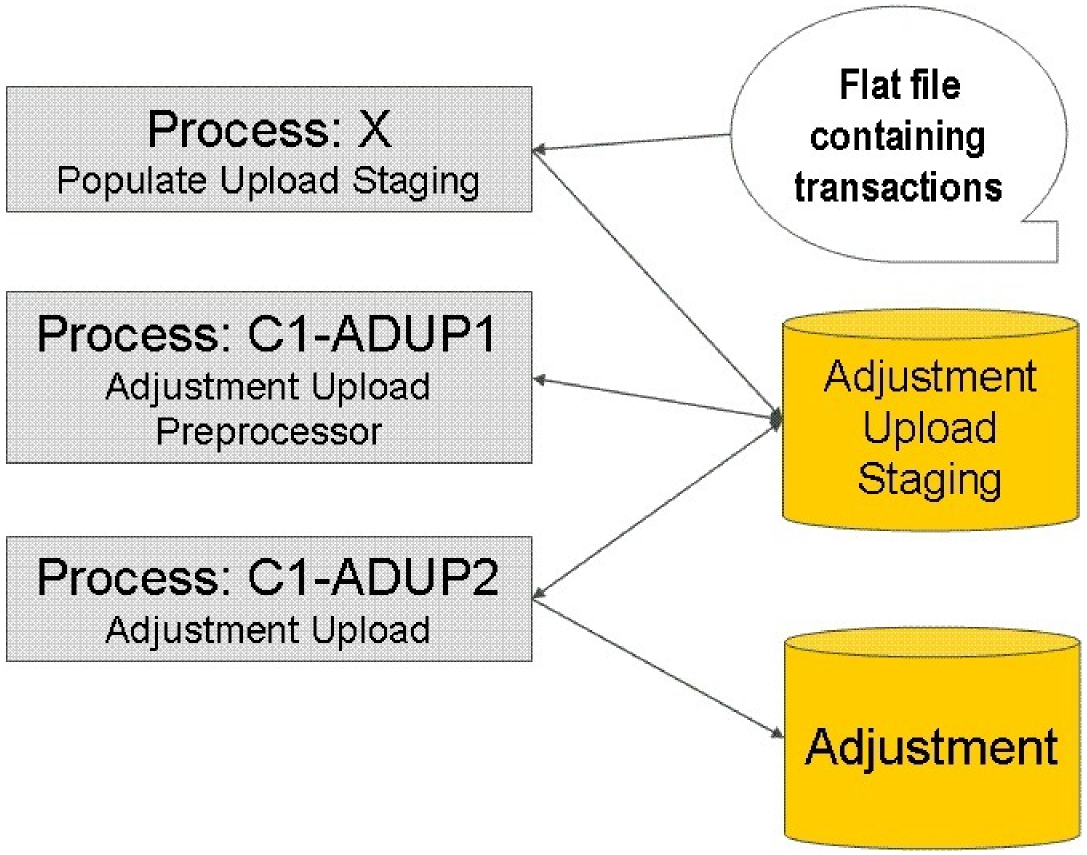
The Approval Profile's Adjustment Types zone lists every *adjustment type* that is governed by this approval profile. This zone appears when there is at least one adjustment type governed by the approval profile displayed in the Approval Profile zone.

Interfacing Adjustments From External Sources

The topics in this section describe how adjustments are uploaded from an external source.

Interfacing Adjustments

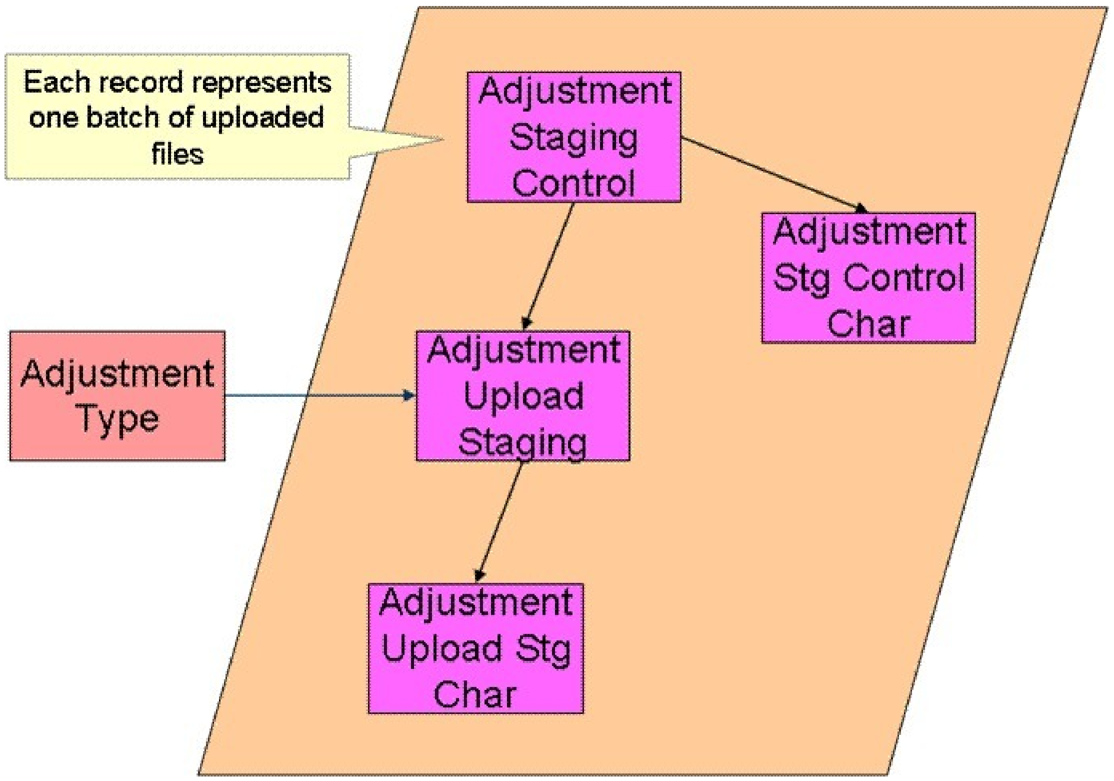
The following diagram illustrates the processes involved in the uploading of adjustments into the system.



The topics in this section describe how these processes work.

Process X - Populate Adjustment Upload Records

Process X refers to the mechanism used by your organization to populate the various staging tables (shown in the shaded section of the following ERD).


Each record represents one batch of uploaded files
Adjustment Staging Control
Adjustment Stg Control Char
Adjustment Type
Adjustment Upload Staging
Adjustment Upload Stg Char

The topics in this section describe each of these tables.

Adjustment Staging Control

You must create an adjustment staging control record for each batch of adjustments to be uploaded into the system. The name of this table is CI_ADJ_STG_CTL. The following describes each column on this table.

Column Name	***Length***	***Req'd***	***Data Type***	***Comments***
ADJ_STG_CTL_ID	12	Y	N	This is the unique identifier of the adjustment staging control record. This key is a sequential number so you can use a database function to assign the value when populating the table.
CRE_DTTM	15	Y	DateTime	This is the date / time on which the adjustment staging control record was created. This must be populated with the current date / time.
ADJ_STG_CTL_ STATUS_FLG	4	Y	A/N	This must be set to **P** for **Pending**.
ADJ_STG_UP_REC_ CNT	10	Y	N	This is the total number of adjustment upload staging records that are linked to this adjustment staging control.
TOT_ADJ_AMT	13.2	Y	N	This column must equal the sum of adjustment amounts on

				the adjustment upload staging records that are linked to this adjustment staging control.
CURRENCY_CD	3	Y	A/N	This must be a valid currency code in the system. Refer to *Defining Currency Codes* for more information.
MESSAGE_CATEGORY	5	N	N	Leave this blank. The adjustment upload preprocessor populates this when an error occurs during upload.
MESSAGE_NBR	5	N	N	Leave this blank. The adjustment upload preprocessor populates this when an error occurs during upload.

Adjustment Staging Control Characteristic

You must create an adjustment staging control characteristic record for each characteristic that you would like to link to the adjustment staging control. The name of this table is CI_ADJ_STG_CTL_CHAR. The following describes each column on this table.

Column Name	***Length***	***Req'd***	***Data Type***	***Comments***
ADJ_STG_CTL_ID	12	Y	N	This must correspond with the prime key of the related CI_ADJ_STG_CTL record.
CHAR_TYPE_CD	8	Y	A/N	This must correspond with a characteristic type that is defined as valid for **adjustment staging control**. Refer to *Setting Up Characteristic Types & Their Values* for more information.
SEQ_NUM	3	Y	N	This should be set to **10** unless you have multiple values for a given adjustment staging control and characteristic type.
CHAR_VAL	16	N	A/N	Populate this field if your *characteristic type* is **predefined**.
ADHOC_CHAR_VAL	254	N	A/N	Populate this field if your *characteristic type* is **ad-hoc** or **file location**.
CHAR_VAL_FK1 - CHAR_VAL_FK5	50 each	N	A/N	Populate these fields if your *characteristic type* is **foreign key reference**. Up to five columns of 50 bytes each are provided to accommodate compound keys.

Adjustment Upload Staging

You must create an adjustment upload staging record for each adjustment you want to upload. The name of this table is CI_ADJ_STG_UP. The following describes each column on this table.

Column Name	***Length***	***Req'd***	***Data Type***	***Comments***
ADJ_STG_UP_ID	12	Y	N	This is the unique identifier of the adjustment upload staging record. This key is a sequential number so you can use a database function to assign the value when populating the table.
ADJ_STG_CTL_ID	12	Y	N	The ID of the adjustment staging control that is linked to this adjustment upload staging record.
ADJ_TYPE_CD	8	Y	A/N	This must correspond to the prime key of one of your adjustment types. Refer to *Setting up Adjustment Types* for more information.
ADJ_STG_UP_STATUS_FLG	4	Y	A/N	This must be set to **P** for **Pending**.
CREATE_DT	10	Y	Date	The date when the adjustment occurred.
ADJ_AMT	13.2	Y	N	The amount of the adjustment.
ADJ_SUSPENSE_FLG	4	Y	N	This must be set to **NSUS** for **Not In Suspense**.
SA_ID	10	N	A/N	This must correspond to a valid obligation in the system. If you leave this blank and opt to let the system find the obligation during adjustment upload preprocessing, a **Determine Obligation** algorithm must be plugged in on the associated adjustment type. This plug-in is meant to derive the obligation ID based on supplied miscellaneous information - e.g. information supplied via adjustment characteristic upload staging.
ADJ_ID	10	N	A/N	Leave this blank. The adjustment upload process populates this.
SUSPENSE_ADJ_ID	10	N	A/N	Leave this blank. The adjustment upload process populates this.
MESSAGE_CATEGORY	5	N	N	Leave this blank. The adjustment upload background processes populate this when an

				error occurs during upload.
MESSAGE_NBR	5	N	N	Leave this blank. The adjustment upload background processes populate this when an error occurs during upload.

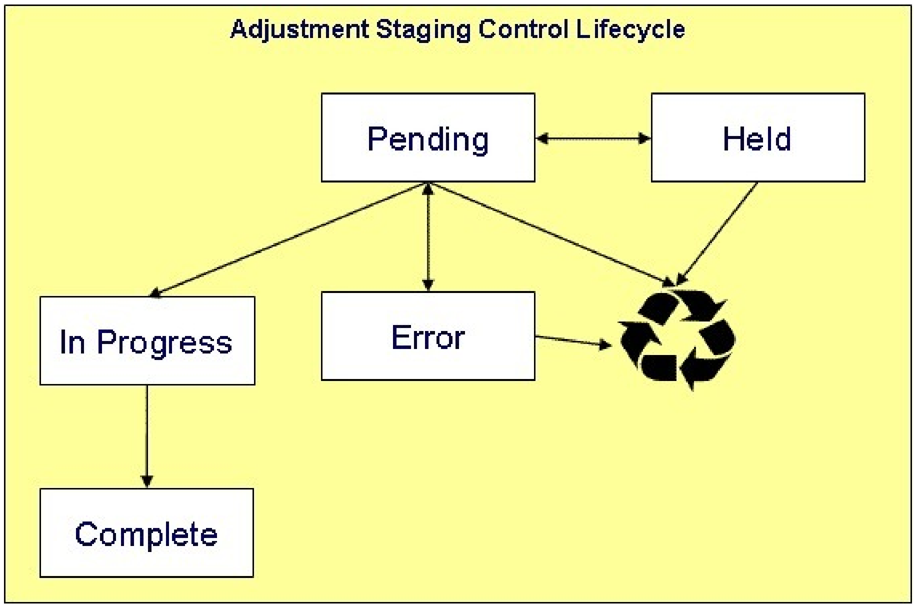
Adjustment Characteristic Upload Staging

You must create an adjustment characteristic upload staging record for each characteristic that you would like to link to the adjustment upload staging and copied to the resulting adjustment. The name of this table is CI_ADJ_STG_UP_CHAR. The following describes each column on this table.

Column Name	*Length*	*Req'd*	*Data Type*	*Comments*
ADJ_STG_UP_ID	12	Y	N	This must correspond with the prime key of the related CI_ADJ_STG_UP record.
CHAR_TYPE_CD	8	Y	A/N	This must correspond with a characteristic type that is defined as valid for **Adjustment**. And must be defined as valid for the adjustment type. Refer to *Adjustment Type Controls Adjustment Characteristics* for more information.
SEQ_NUM	3	Y	N	This should be set to **10** unless you have multiple values for a given adjustment upload staging and characteristic type.
CHAR_VAL	16	N	A/N	Populate this field if your *characteristic type* is **predefined**.
ADHOC_CHAR_VAL	254	N	A/N	Populate this field if your *characteristic type* is **ad-hoc** or **file location**.
CHAR_VAL_FK1 - CHAR_VAL_FK5	50 each	N	A/N	Populate these fields if your *characteristic type* is **foreign key reference**. Up to five columns of 50 bytes each are provided to accommodate compound keys.

The Lifecycle of an Adjustment Staging Control Record

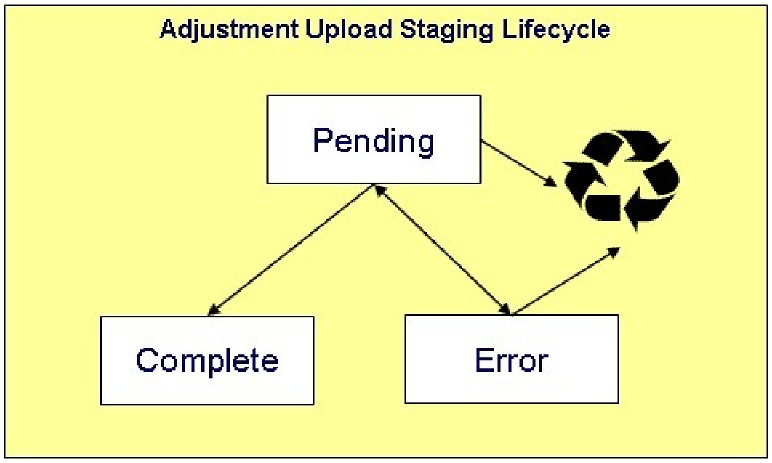
The following diagram shows the possible lifecycle of an adjustment staging control record.



- **Pending.** An adjustment staging control record is created in this state. The **C1-ADUP1** process selects **pending** adjustment staging control records for validation.
- **In Progress.** The **C1-ADUP1** process sets a **pending** record to **in progress** when the totals on the adjustment staging control are successfully validated against the totals from the associated adjustment upload staging records.
- **Complete**. The **C1-ADUP2** process sets the adjustment staging control's status to **complete** when all adjustment upload staging records linked to the adjustment staging control are **complete**.
- **Error.** The **C1-ADUP1** process sets a **pending** record to **error** if the adjustment staging control fails validation. The status may be set back to **pending** after the error is fixed.
- **Held.** The **held** status is available for situations where you want to prevent or delay the upload of a batch of adjustment staging records. The status may be set back to **pending** when the batch of records is ready for upload.
- **Pending, error** and **held** records may be deleted.

The Lifecycle of an Adjustment Upload Staging Record

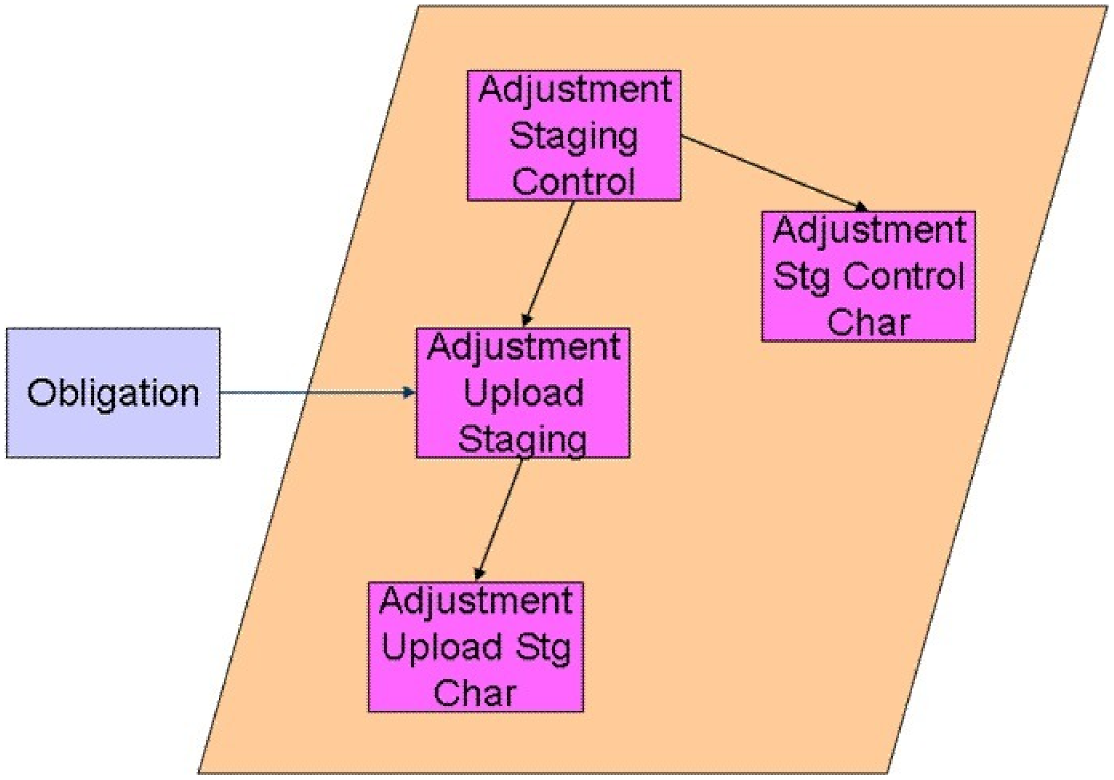
The following diagram shows the possible lifecycle of an adjustment upload staging record.



- **Pending.** Adjustment upload records are created in this state. The **C1-ADUP2** process selects **pending** adjustment upload records and creates adjustments for each of them.
- **Complete.** The **C1-ADUP2** process sets a **pending** record to **complete** if an adjustment is successfully created.
- **Error.** Either **C1-ADUP1** or **C1-ADUP2** process sets **pending** record to **error** when it encounters an error during the upload process. The status may be set back to **pending** after the error is fixed.
- **Pending** and **error** records may be deleted.

Process C1-ADUP1 - Preprocess Adjustment Uploads

The batch process identified by batch process ID **C1-ADUP1** refers to the background process that validates adjustment staging control records and populates obligation IDs on adjustment upload staging records that do not specify an obligation ID.


Adjustment Staging Control
Adjustment Stg Control Char
Obligation
Adjustment Upload Staging
Adjustment Upload Stg Char

Phase 1 - Validate Adjustment Staging Controls

The following points describe, at a high level, the first phase of the adjustment upload pre-process:

For each **Pending** adjustment staging control,

- Check that the record count on the adjustment staging control record equals the number of adjustment upload staging records that are linked to the adjustment staging control.
- Check that the total adjustment amount on the adjustment staging control record equals the sum of the adjustment amounts from the adjustment upload staging records that are linked to the adjustment staging control.
- If the adjustment staging control passes validation, set its status to **In Progress**. Otherwise, set the status to **Error**. Create a To Do entry using the inputs To Do type and To Do role (if supplied) for adjustment staging control errors. (Complete any outstanding To Do entries for the adjustment staging control before creating a new To Do entry.)

- If no errors occur, perform To Do cleanup by completing any outstanding To Do entries that were previously created for the adjustment staging control.

NOTE: You can fix errors by going to the To Do entry and drilling into the adjustment staging control page. Don't forget to change the adjustment staging control's status back to **Pending** after fixing the error.

Phase 2 - Populate Obligation ID

The following points describe, at a high level, the second phase of the adjustment upload pre-process:

For each **Pending** adjustment upload staging that is linked to an **In Progress** adjustment staging control and does not have the obligation ID,

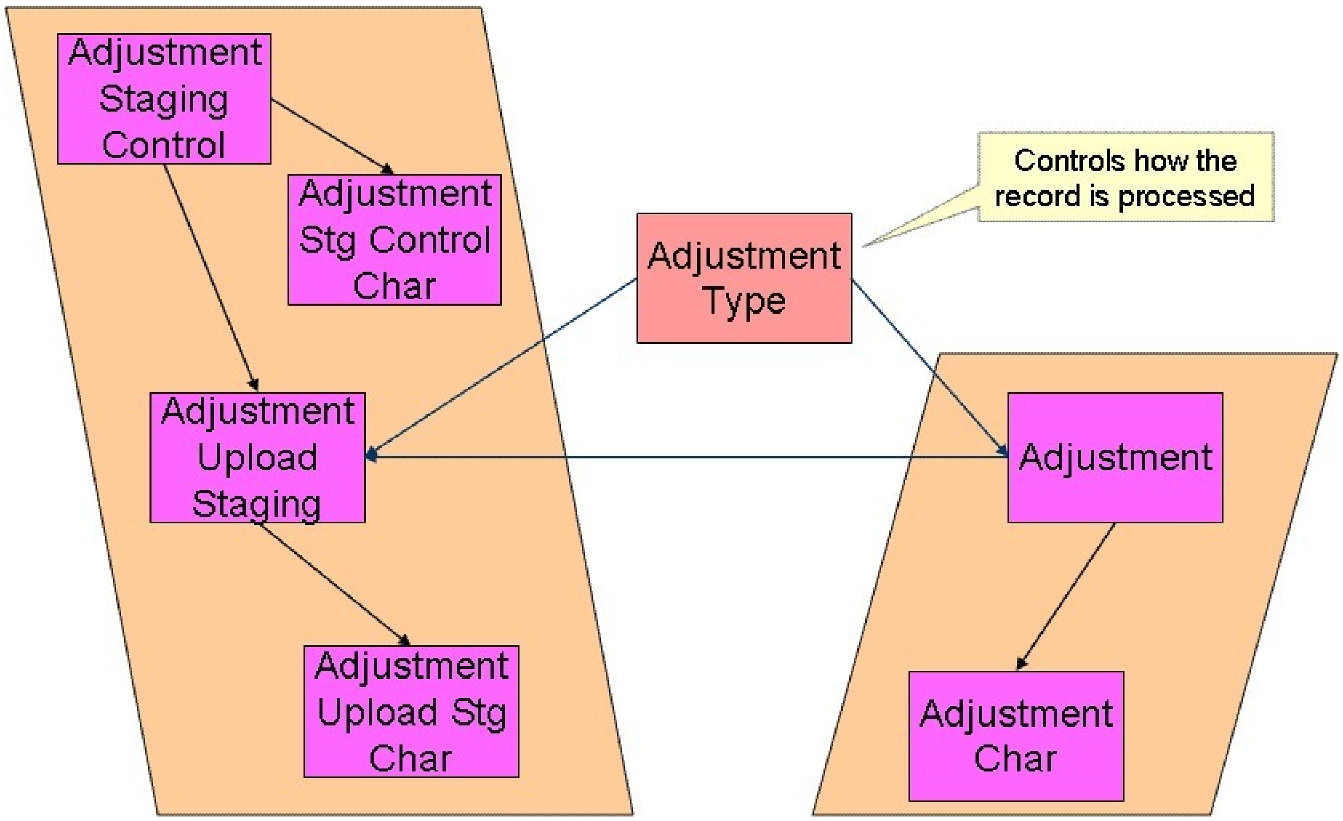
- Execute the **Determine Obligation** algorithm that is plugged in on the *adjustment type*.
- If the algorithm returns a valid obligation ID, stamp that obligation ID onto the adjustment upload staging. If the algorithm also returns an indication that the adjustment is to be put into suspense, set the suspense flag on the adjustment upload staging to **In Suspense**. Refer to *Suspense Adjustments* for more information on how suspense adjustments are handled.
- If the algorithm returns an error, set the adjustment upload staging record's status to **Error**. Create a To Do entry using the inputs To Do type and To Do role (if supplied) for adjustment upload staging errors. (Complete any outstanding To Do entries for the adjustment upload staging before creating a new To Do entry.)
- If no errors occur, perform To Do cleanup by completing any outstanding To Do entries that were previously created for the adjustment upload staging.

NOTE: You can fix errors by going to the To Do entry and drilling into the adjustment upload staging page. Don't forget to change the adjustment upload staging's status back to **Pending** after fixing the error.

Process C1-ADUP2 - Upload Adjustments

The batch process identified by batch process ID **C1-ADUP2** refers to the background process that creates adjustments for all adjustment upload staging records that are stamped with an obligation ID.

The following diagram and section describe, at a high level, the processing done in the **C1-ADUP2** background process.


Adjustment Staging Control
Adjustment Stg Control Char
Adjustment Upload Staging
Adjustment Upload Stg Char
Adjustment Type
Controls how the record is processed
Adjustment
Adjustment Char

- For each **Pending** adjustment upload staging that has an obligation ID,
- Add a frozen adjustment using the amount, creation date and adjustment type on the staging record. Populate the adjustment characteristics with the values in the adjustment characteristic upload staging records.

NOTE:
If the type of adjustment being uploaded is one that is calculated, the algorithm that's plugged in on the adjustment type will calculate the adjustment amount and generate associated calculation lines (if applicable). See *Calculated Adjustments* for more information.

- If adjustment creation is successful, update the adjustment ID on the staging record and set the staging record's status to **Complete**.
- If adjustment creation results in an error, set the staging record's status to **Error**. Create a To Do entry using the To Do type on which this batch process (**C1-ADUP2**) is defined as creation process. (Complete any outstanding To Do entries for the adjustment upload staging before creating a new To Do entry.)
- If no errors occur, perform To Do cleanup by completing any outstanding To Do entries that were previously created for the adjustment upload staging.

Suspense Adjustments

What Are Suspense Adjustments?

When the adjustment upload preprocessor is unable to identify a valid obligation ID, it can post the adjustment to a suspense obligation. This is similar to how a payment is posted to a suspense account when a valid account ID could not be identified during payment upload.

Putting adjustments in suspense is optional. Therefore, this logic sits in a plug-in spot. If suspense adjustments are applicable to you, you must plug in your suspense logic in a **Determine Obligation** algorithm on *Adjustment Type*.

How Are Suspense Adjustments Resolved?

A background process can be run periodically to automatically resolve suspense adjustments. Since rules for resolving suspense may vary, this logic sits in a plug-in spot. You must plug in your resolve logic in a **Resolve Suspense** algorithm on *Adjustment Type*.

Process C1-ADURS - Resolve Suspense Adjustments

The batch process identified by batch process ID **C1-ADURS** refers to the background process that resolves suspense adjustments.

For each adjustment upload staging that is **In Suspense**, execute the **Resolve Suspense** algorithm that is plugged in on the *adjustment type*.

Managing Bill Setup

Billing is one method used to establish assessments for bill-based tax types. Examples of these tax types include real property and vehicle taxes. The billing process is used when the revenue authority has the information required to calculate and assess charges for a taxpayer, in contrast to tax types where the taxpayer must file a tax return.

The topics in this section provide background information about a variety of billing topics.

The Big Picture of Bills

A bill is an object provided to support the calculation of charges to establish assessments. In particular, bills are designed for levying assessments based on the value of an asset. A bill references a single tax role, obligation and revenue period. An obligation may have more than one bill for the same period but the product does not expect two bills of the same type within the period.

A bill captures three key sets of information:

- Value details are the values that form the basis of the bill calculation. Value details will most commonly be derived from asset valuations.
- Calculation lines capture the results of the computation steps used to calculate the bill. In addition, they capture the information required to create bill segments.
- Bill segments record bill details from which the financial transactions can be created, including due dates and amounts.

The topics in this section highlight billing functionality.

How Are Bills Created?

Many asset based bills, such as property taxes, cover a single revenue year. They are usually created all at once and in advance to meet statutory requirements for notifying taxpayers However, other bill types may have differing cycles. Vehicle taxes, for example, may be billed on the anniversary of the registration date.

The product provides a batch process that will create pending bills for a given revenue period and tax type. It creates bills for all active tax roles for the tax type and it expects the appropriate obligation to already exist.

FASTPATH:
Your tax types may be configured to create obligations at the appropriate intervals. Refer to the section on maintaining obligations in *Configuring Tax Types* for more information.

Bill Lifecycle

Bills require information unique to the tax type, tax role and bill type in order to calculate charges. In addition, the business rules governing the applicable charges and the computations themselves can be complex. Unlike other system objects, the product does not expect bill details to be manually maintained. Instead the bill details and associated assessments are created by background processes invoked at specific times in the bill's lifecycle.

The product supports the following bill lifecycle, covering the most typical scenarios for billing:

- Bills are initially created in a **Pending** status with minimal information other than the tax role, obligation and revenue period.
- When bills transition to the **Generated** state, algorithms are invoked to capture the value details and create the calculation lines and bill segments that form the basis of the assessed charges.
- If generation is successful, the bills transitions directly to the **Ready To Complete** state.
- If there are any bill generation exceptions, the bills transitions to the **Error** state for manual correction.
- When bills transition to the **Completed** state, algorithms are invoked to finalize the bill. The product expects that the algorithms that create financial transactions for the bill's obligation will be invoked at this time.
- If a bill needs to be reversed, it must be manually transitioned to the **Canceled** state. The product expects that the algorithms that reverse the financial transactions for the bill will be invoked at this time.

Refer to the base business object for bills **C1-TaxBill** for more details.

Bill Types and Algorithms

The processing required at the various stages of the bill's lifecycle is performed by algorithms linked to the bill type. This allows for various business use cases for different bill and tax types to be implemented without requiring different business objects. A specific business object for a bill type is only required if there are explicit elements that need to be provided for a given use case.

Bill types are associated with specific tax types, to provide control over the bill types that may be generated for a tax role.

Using The Calculation Engine With Bills

The bill object is designed to support the use of the calculation engine to handle the creation of the calculation lines that make up the bill. Bills can reference a calculation control version that defines the calculations to be performed.

The base batch process that creates bills refers to configuration on the tax role to determine the appropriate calculation control version.

FASTPATH:
Refer to *Defining Calculation Engine Options* for more information on using calculation rules to compute bill charges.

Using The Batch Control Group

Some revenue authorities choose to process bills in groups based on criteria specific to their business practice.

The base batch process that monitors and processes bills provides the ability to select which bills are processed based on the value of the batch control group field on the bill. Multiple control groups can be selected for inclusion or exclusion based on the batch parameter setting.

The product expects that the batch control group is maintained on the bill either manually or via a custom batch process for that identifies bills for grouping based on your implementation’s business rules.

Configuring Bills

The topics in this section describe configuration requirements.

In addition, your implementation should read the detailed description for the base product business objects provided for Bill Type (**C1-TaxBillType**).

Configuring Bill Type Algorithms

The system provides four plug-in spots on the bill type to perform logic that is specific to maintaining a bill of that type.

- The generation plug-in is executed when transitioning a bill of this type from pending to generated. It is responsible for ensuring that the bill is ready for completion. In particular, algorithms for this plug in are responsible for populating the bill value details, calculation lines and bill segments.

  NOTE: Base plug-ins. Click *here* to see the algorithms types available for this system event.

- The determine due dates plug-in is executed from within the algorithm responsible for creating bill segments and is responsible for determining the number of bill segments / installments for the bill and their due dates.

  NOTE: Base plug-ins. Click *here* to see the algorithms types available for this system event.

- The completion plug-in is executed when transitioning a bill of this type from generated to completed. It is responsible for ensuring that the bill is finalized. In particular, algorithms for this plug in spot are responsible for creating financial transactions from bill segments.

  NOTE: Base plug-ins. Click *here* to see the algorithms types available for this system event.

- The cancellation plug-in is executed when transitioning a bill of this type from completed to canceled. It is responsible for ensuring that the bill is reversed. In particular, algorithms for this plug in spot are responsible for creating cancellation financial transactions from bill segments.

  NOTE: Base plug-ins. Click *here* to see the algorithms types available for this system event.

Setting Up Bill Types

A Bill Type defines the configuration information that is common to bills of a given type. The type of information captured on the bill type is governed by the bill type's business object.

To set up a Bill Type, select **Admin Menu** > **Bill Type**.

The topics in this section describe the base-package zones that appear on the Bill Type portal.

Bill Type List

The following functions are available:

- Click a *broadcast* button to open other zones that contain more information about the adjacent bill type.
- The standard actions of **Edit, Duplicate** and **Delete** are available for each bill type.

Click the **Add** link in the zone's title bar to add a new bill type.

Bill Type

The Bill Type zone contains display-only information about a bill type. This zone appears when a bill type has been broadcast from the Bill Type List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference C1_TAX_BILL_TYPE.

Billing Batch Jobs

The following batch job must be run to progress bill.

- **C1-TBMDF** (Bill Deferred Monitor with Date). This batch job performs standard transitioning logic to progress bills to the next state. It provides additional selection criteria based on monitor date, tax type and batch control group

Managing Payment Setup

The topics in this section describe maintenance transactions related to payment topics.

FASTPATH:
Refer to *Tender Management and Workstation Cashiering* before setting up the control tables described in this section.

Setting Up Payment Segment Types

Every obligation references an obligation type. Among other things, the obligation type references a payment segment type. The payment segment type controls how payment segments and their related financial transactions are created. To set up payment segment types, open **Admin Menu** > **Payment Segment Type**.

Description of Page

Enter an easily recognizable **Payment Segment Type** and **Description** for every type of payment segment.

FASTPATH:
For more information about the source of the distribution codes on financial transactions, see *The Source Of GL Accounts On Financial Transactions*.

For each payment segment type, define the **Payment Segment Fin Algorithm**. The logic embedded in this algorithm constructs the actual financial transaction associated with the payment segment. Refer to *Examples of Common Payment Segment Types* for examples of how algorithms are used on common payment segment types.

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 constructs the payment segment financial transaction in the appropriate manner. Click *here* to see the algorithm types available for this plug-in spot.

FASTPATH:
For more information about current and payoff amount, see *Current Amount versus Payoff Amount*.

Examples of Common Payment Segment Types

The following table shows several classic payment segment types used by many organizations:

Payment Segment Type	***Payment Segment Financial Transaction Algorithm***
Normal payment (if you practice accrual accounting). Refer to *Accrual versus Cash Accounting* for more information.	*Payoff = Pay Amount / Current = Pay Amount (no cash accounting)*
Normal payment (if you practice cash accounting). Refer to *Accrual versus Cash Accounting* for more information.	*Payoff = Pay Amount / Current = Pay Amount (plus Cash Accounting)*
Charity payment	*Payoff =0 / Current = Pay Amount (the GL is affected)*

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_PAY_SEG_TYPE*.

Setting Up Tender Types

Tender types are used to indicate the method in which the tender was made. A unique **Tender Type** must exist for every type of tender that can be remitted. For example, if you allow cash, checks, direct debits from a checking account, and direct debits from a credit card to be tendered, you'd need the following tender types:

Tender Type	***Description***	***Like Cash***	***Generate Auto Pay***	***Require External Source ID***	***Require Expiration Date***	***External Type***
CASH	Cash	Yes	No	N/A	N/A	N/A
CHEC	Check	No	No	N/A	N/A	N/A
OVUN	Cash drawer - over/under	No	No	N/A	N/A	N/A
DDCH	Direct debit - checking	No	Yes	Yes	No	***Checking withdrawal***
CRED	Direct debit - credit card	No	Yes	No	Yes	***Credit card withdrawal***

Go to **Admin Menu** > **Tender Type** to define your tender types.

Description of Page

Enter a unique **Tender Type** and **Description** for the tender type.

Turn on the **Like Cash** switch if this tender type is cash or the equivalent of cash. This indicator controls if the system generates a warning if a cash-only account remits a tender other than cash. It is also used to generate a warning for online cashiers to turn in their tenders when the cash-like amount exceeds the maximum amount balance defined for the *tender source*.

Turn on **Generate Auto Pay** if this type of tender causes an automatic payment request to be routed to a financial institution. For example, this switch will be on if this tender type is used for direct debits from a taxpayer's checking account (because every tender of this type will have an automatic payment request created when the tender is created).

The following fields are only used for tender types associated with automatic payments:

- **External Type** is used by the background process that creates the information that is interfaced to the automatic payment source. Specifically, it controls the record type associated with the different types of automatic payments that are routed to the automated clearinghouse (ACH).

  NOTE: The values for this field are customizable using the Lookup table. This field name is EXT_TYPE_FLG.

- **External ID Required** indicates if an Auto Pay Source that references this type of tender must contain an External Source ID. The External Source ID is the unique identifier of the financial institution to which the automatic payment will be routed.

  This switch is typically turned on for tender types associated with checking / saving direct debits. It is turned off for tender types associated with credit card debits (you don't need an external source for a credit card debit; you just need the credit card number).

- **Expiration Date Required** indicates if an Auto Pay option that references an Auto Pay source that references this type of tender must also contain an expiration date (e.g., automatic debit / credit cards).

  Turn this switch off for tender types associated with checking / saving direct debits.

Turn on **Allow Cash Back** if the system should automatically calculate a cash back amount when a tender is remitted for this tender type and the amount tender exceeds the amount being paid.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_TENDER_TYPE*.

Setting Up Tender Sources

A unique **Tender Source** must exist for every potential source of funds. For example,

- Every cashiering station will have a unique tender source.
- Every lock box will have a unique tender source.
- Your remittance processor will have a unique tender source.
- If you allow taxpayers to pay bills automatically (e.g., via EFT), you'll need a tender source for each institution to which you route automatic payment requests. For example, if you route automatic payment requests to the automated clearinghouse (ACH), you'll need a tender source for the ACH.

For example, if you have 3 lock boxes, 2 cash drawers at an area office A, 2 cash drawers at area office B, and a single remittance processor, you'd need the following tender sources:

Tender Source	*Type*	*External Source ID (Lockbox ID)*	*Default Starting Balance*	*Currency Code*	*Suspense Obligation*

CASH-A01	Cashiering	N/A	150.00	USD	N/A
CASH-A02	Cashiering	N/A	150.00	USD	N/A
CASH-B01	Cashiering	N/A	150.00	USD	N/A
CASH-B02	Cashiering	N/A	150.00	USD	N/A
LB-INDUS	Lockbox	112910-A	N/A	USD	9291019281
LB-COMM	Lockbox	938219-C	N/A	USD	4739837372
LB-RESID	Lockbox	372829-B	N/A	USD	1912910192
REMIT	Lockbox	N/A	N/A	USD	1920038437
ACH	Auto Pay	N/A	N/A	USD	N/A

To set up a tender source, select **Admin Menu** > **Tender Source**.

Description of Page

Enter an easily recognizable **Tender Source** and **Description** for the tender source.

Define the **Tender Source Type**. Valid values are: **Ad Hoc, Auto Pay**, **Online Cashiering** and **Lockbox**. The system uses this information to prevent tender controls from different sources from being included under the same deposit control. In other words, you can't mix ad hoc, automatic payment, cashiering and lockbox tenders under the same deposit control.

FASTPATH:
For more information, refer to *Maintaining Deposit Controls*.

If the source is an external system (e.g., a lockbox or an automatic payment destination), use **External Source ID** to define the unique identifier of the source. The background process that interfaces tenders from this source uses this information to create the appropriate tender control when it interfaces payments from external sources.

If this source is a cash drawer, define the **Default Starting Balance**. This balance is defaulted onto new tender controls and may be overridden.

NOTE:
The tender type of the **Start Balance** is defined on the installation record.

If this source is a cash drawer, define the **Max Amount Balance**. When the amount of *cash-like* tenders in a cash drawer exceeds this balance, a warning is issued to remind the cashier to turn in some of the funds to a tender control.

Define the **Currency Code** of tenders linked to this source. All tenders in a source must be of the same currency.

If this tender source is associated with payments that are *interfaced from an external source* (e.g., a lockbox or a remittance processor), use Suspense **Obligation** to define the obligation whose account will hold uploaded payments with an invalid account. Refer to *Payment Upload Error Obligations* for more information about suspense obligations. Also note, because the payment upload process simply books payments that reference invalid accounts to the account associated with this obligation, this account should belong to an account type with the appropriate payment distribution algorithms. This may entail creating a new account type that will only be used on these "suspense accounts". This account type would need the following algorithms:

- We'd recommend using a simple payment distribution algorithm like *PYDIST-PPRTY* (distribute payment based on obligation type's payment priority).
- We'd recommend using an overpayment distribution algorithm like *OVRPY-PPRTY* (distribute overpayment to highest priority obligation type).

Define the **Bank Code** and **Bank Account** into which the tender source's moneys will be deposited. The bank account defines the distribution code used to build the GL details for the payment. Refer to *The Source of GL Accounts on Financial Transactions* for more information. Note that the bank code and bank account can later be overwritten when entering Tender Deposits on *Deposit Control*.

If this tender source is associated with payments that are *interfaced from an external source*, for example tender sources associated with **Auto Pay** and **LockboxTender Source Types**, the information is also used as follows:

- The *payment upload process* uses this information to populate the bank and bank account when it creates deposit control records for the tender controls it creates during the interface. Refer to *Managing Payments Interfaced From External Sources* for more information.
- The *automatic payment interface* uses this information to populate the bank and bank account when it creates deposit control records for the tender controls it creates during the interface.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_TNDR_SRCE*.

Setting Up Payment Cancellation Reasons

Open **Admin Menu** > **Pay Cancel Reason** to define your payment cancellation reason codes.

Description of Page

Enter an easily recognizable **Cancel Reason** and **Description** for the payment cancellation reason.

Cancel Activated Auto Pay indicates whether or not a payment tender with an activated (i.e. stamped with the extract batch code and next run number) auto pay staging record can be canceled. During tender cancellation, if the selected Cancel Reason has this indicator set to **Allowed**, the tender is canceled. (The auto pay staging record is kept, but its scheduled extract date and distribution/freeze date are removed.) If the selected Cancel Reason has this indicator set to **Not Allowed**, tender cancellation will issue an error.

Turn on the **NSF Charge** switch if the system should invoke the non-sufficient funds (NSF) algorithm when a tender is cancelled using this reason code. Refer to *NSF Cancellations* for more information.

The following fields are used to change an account's compliance rating if a tender is canceled using the respective reason code.

- Use **Affect Compliance Rating By** to define how tenders canceled using this reason will affect the account's compliance rating. This should be a negative number. A taxpayer's compliance rating is equal to the start compliance rating amount defined on the installation record plus the sum of compliance rating demerits that are currently in effect.
- Use **Months Affecting Compliance Rating** to define the length of time the demerit remains in effect. This information is used to define the effective period of the compliance rating demerit record.

FASTPATH:
For more information, refer to *Account - Collections*.

NOTE:
The payor gets the compliance rating hit. When you cancel a tender you must specify a cancellation reason. If the cancellation reason indicates a compliance rating demerit should be generated, the system levies the compliance rating transaction on the payor's account.

The **System Default Flag** is specified on those cancellation reasons that are placed on payment segments that are automatically cancelled by the system. Valid values are: **Re-opened Bill**. The **Re-opened Bill** value is used as follows:

- Payments are automatically created for accounts who pay their bills automatically when their bills are completed.
- If such a bill is reopened before the automatic payment is interfaced to the paying authority, the system automatically cancels the payment. The **Re-opened Bill** cancellation reason is placed on such payments.

Automatic Payment Options

If your taxpayers can pay their bills automatically (via direct debit or credit card debits), you'll need to set up the various control tables described in this section.

FASTPATH:
Refer to *Automatic Payments* for more information about how automatic payments are handled in the system.

Setting Up Auto-Pay Route Types

Auto Pay Route Types are used to control when and how automatic payment requests are routed to a financial institution, and when the general ledger is impacted. Select **Admin Menu** > **Auto Pay Route Type** to define your route types.

Description of Page

To modify an auto pay route type, simply move to a field and change its value.

To add a new route type, press + to insert a row, then fill in the information for each field. The following fields display:

- **Route Type** is the unique identifier of the route type.
- **Description** is the description of the route type.
- **Tender Source** is the tender source to use when creating a tender control for the payments. The background process that routes automatic payment requests to a financial institution (e.g., the automated clearing house interface) will mark each automatic payment's associated tender with a tender control for audit and control purposes. The following points describe how this happens:
    - When the system sees that it's time to send an automatic payment to a financial institution, it looks at the automatic payment's auto pay source.
    - Every auto pay source references an auto pay route type.
    - Every auto pay route type references a tender source.
    - A **Tender Source** has a tender control for each group of tenders deposited / interfaced together one batch.
    - The system marks each automatic payment's associated tender with the latest tender control for the **Tender Source**. The system will create a new tender control each time it routes automatic payments to the tender source. Refer to *Managing Payments Interfaced From External Sources* for more information about tender source and tender control.
- **Extract Batch Code** defines the background process that interfaces the automatic payment requests to the financial institution.
- **Auto Pay Date Calculation Alg** populates 3 dates associated with the automatic payment: 1) the date the automatic payment will be sent to the financial institution, 2) the date the general ledger will be impacted by the automatic payment, 3) the date of the payment.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_APAY_RT_TYPE*.

Setting Up Auto-Pay Source Codes

A unique **Auto Pay Source** must exist for every bank / credit card company / bill payment service that your taxpayer's use as the source of the funds when they sign up for automatic payment. For example,

- Every bank will have a unique auto pay source.
- Every credit card company will have a unique auto pay source.

To set up an auto pay source, select **Admin Menu** > **Auto Pay Source Type**.

Description of Page

Enter an easily recognizable **Auto Pay Source Code** and **Description** for the auto pay source.

The **Source Name** is the name of the financial institution.

When the system creates an automatic payment request, it also creates an associated payment tender. This tender (like all tenders) must have a tender type. This field defines the **Tender Type** associated with this auto pay source's tenders. Refer to *Setting Up Tender Types* for more information.

The **External Source ID** is the unique identifier of the financial institution to which the automatic payment will be routed (e.g., the bank routing ID of the bank). This field is typically blank on automatic payments routed to credit card companies because the credit card company doesn't have an external source ID (whereas direct debits from banks must have a bank routing number). Whether this field is required is controlled by the **Tender Type**.

The **Auto Pay Route Type** controls when and how automatic payment requests get routed to a financial institution. It also controls when the general ledger is impacted by the automatic payments financial transaction. Refer to *Setting Up Auto-Pay Route Types* for more information.

The **Work Calendar** defines the financial institution's workdays. This information is used to determine the date on which automatic payment requests will be sent to the financial institution. Refer to *Setting Up External Workday Calendars* for more information.

The **Validation Algorithm** defines how the system validates the taxpayer's account ID at the financial institution. 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 validates the taxpayer's account ID at the financial institution. Click *here* to see the algorithm types available for this plug-in spot.

Refer to *Account - Auto Pay* for more information.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_APAY_SRC*.

SEPA Payments

The topics in this section provide background information about Single Euro Payments Area (SEPA) payment functionality.

NOTE:
This section is only relevant for some organizations. The system configuration requirements described in this section are only relevant if your organization is a participant in SEPA payment transactions such as direct debit collections.

What Is SEPA?

SEPA (Single Euro Payments Area) is a European Union (EU) integration initiative that is aimed at streamlining processes that are related to cross-border payments.

SEPA consists of the EU member states plus a few additional countries. In SEPA, customers can make electronic Euro payments within and across these countries under the same rights and obligations.

SEPA payment services are based on global ISO (International Organization for Standardization) standards.

SEPA Direct Debit

The SEPA Direct Debit (SDD) scheme allows a creditor to collect money from the debtor, with prior approval from the debtor.

The direct participants are the following:

- Creditor
- Creditor's bank
- Debtor
- Debtor's bank

The debtor and creditor must each hold an account with a payment service provider located within SEPA. The accounts may be in Euro or in any other SEPA currency. However, the transfer of funds between the debtor's bank and the creditor's bank always takes place in the Euro currency.

The indirect participants are the following:

- Clearing and Settlement Mechanisms (CSMs) such as an automated clearinghouse
- Intermediary Banks that offer intermediary services to debtor banks and/or creditor banks

SEPA Direct Debit Mandate

The mandate is the consent and authorization that the debtor gives to the creditor, to allow the creditor to initiate direct debit collections. The creditor is responsible for storing the original mandates, together with any amendments relating to the mandate or information regarding its cancellation or lapse.

The topics in this section describe how mandates are issued and canceled.

Issuing A Mandate

The creditor initiates the issuance of a mandate by sending the mandate form (either paper or electronic) to the debtor with the creditor information filled in. The creditor information includes the creditor's unique identifier as a SEPA creditor. An ISO standard specifies the structure of the creditor identification, which includes country code, a check digit, the creditor's business code and a country-specific identifier for the creditor.

The creditor ensures that the mandate form contains the mandatory legal wording and the mandatory set of information, as specified in the standards.

The debtor must ensure that the required information is supplied and that the mandate is signed, either in writing or electronically. The mandatory debtor information includes the debtor's international bank account number (IBAN) and bank identifier code (BIC).

The creditor assigns a unique reference for the mandate and provides that reference to the debtor before the first initiation of a collection. The debtor can then use both the unique mandate reference and the creditor identification to verify the bank transactions.

Canceling A Mandate

A mandate can be canceled by either the debtor or the creditor, without the involvement of either bank.

SEPA Master Configuration

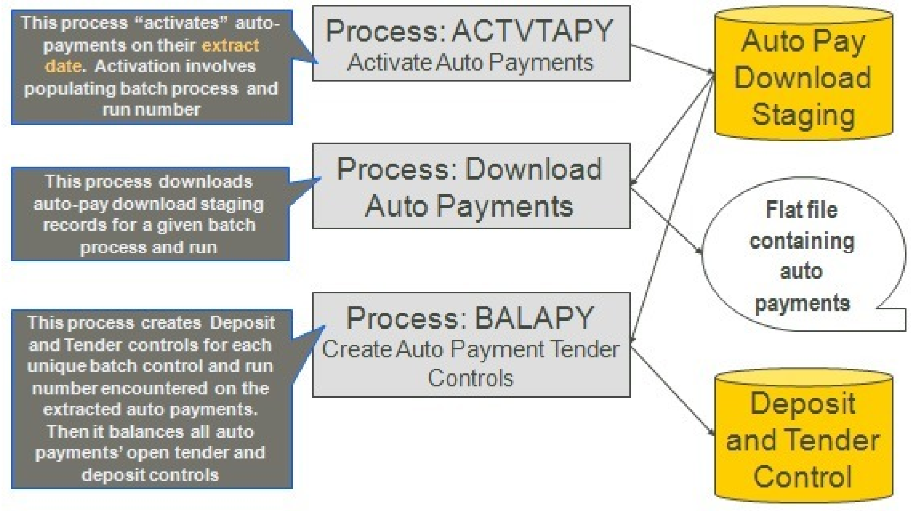
The SEPA Master Configuration contains general information that SEPA extract processes need to format into the output file. This information includes constants and other specific information that identifies the tax authority as a SEPA creditor/ debtor.

The base product provides a business object for SEPA Master Configuration, **C1-SEPAMasterConfiguration.**

For details on how to set up master configurations, refer to *Defining Master Configuration* in the Framework help.

Downloading Automatic Payments and Interfacing Them To The GL

The following diagram illustrates the background processes that interface automatic payment out of the system:


This process "activates" auto-payments on their extract date. Activation involves populating batch process and run number
Process: ACTVTAPY
Activate Auto Payments
Auto Pay Download Staging
This process downloads auto-pay download staging records for a given batch process and run
Process: Download Auto Payments
Flat file containing auto payments
This process creates Deposit and Tender controls for each unique batch control and run number encountered on the extracted auto payments. Then it balances all auto payments' open tender and deposit controls
Process: BALAPY
Create Auto Payment Tender Controls
Deposit and Tender Control

The base product provides two options for extracting auto pay staging records - APAYACH (for ACH payments) or C1-SDDCE (for SEPA payments).

The auto pay background processes are described in the following topics.

ACTVTAPY - Activating Automatic Payments

When an automatic payment is first created, it gets marked with an extract date. The extract date is the date the automatic payment will be downloaded to the respective financial institution. The extract date is determined as follows:

- Every automatic payment references an auto pay source.
- Every auto pay source references an auto pay route type.
- Every auto pay route type contains an algorithm that calculates this date.

On the extract date, the automatic payment is "activated". The automatic payment is activated is marked for download the next time its download process runs. An automatic payment's download process is defined on its auto pay source's route type.

APAYACH - Download Automatic Payments To The ACH (automated clearing house)

This process reads all auto pay download staging records marked with a given batch control ID & run number and creates the flat file that's passed to the ACH. Refer to *ACH Record Layouts* for the details of the record layouts.

NOTE: You can rerun this process. You can reproduce the flat file at any time. Simply request this job and specify the run number associated with the historic run.

If you require a different flat file format, you must create a customized extract program. This program may be used as a starting point.

ACH Record Layouts

The topics in this section describe the layout of the records created by *APAYACH - Download Automatic Payments To The ACH (automated clearing house)*.

File Header Record

The ACH extract flat file must have one record of this type and it must be the first logical record on file.

Field Name	***Format***	***Source/Value/Description***
RECORD-TYPE	A1	"1"
PRIORITY-CD	A2	"01"
RESERVED-01	A1	Spaces
IMMEDIATE-DESTINATION	A9	CI_BANK_ACCOUNT.DFI_ID_NUM
COMPANY-ID	A10	CI_BANK_ACCOUNT.ACCOUNT_NBR
FILE-CRE-DT	A6	YYMMDD. Current date
FILE-CRE-TM	A4	HHMM. Current time
FILE-ID-MODIFIER	A1	"A"
RECORD-SIZE-CONST	N3	"094" The "FILLER" at the end of all but the Entry Detail Record serves to bring the total length of each record up to this constant.
BLOCKING-FACTOR	N2	"10"
FORMAT-CD	A1	"1"
ORIG-FIN-INST-NAME	A23	CI_BANK_L.DESCR
COMPANY-NAME	A23	CI_BANK_ACCOUNT_L.DESCR
REFERENCE-CD	A8	Spaces

Company Batch Header Record

The ACH extract flat file must have one record of this type and it must be the second logical record on file.

Field Name	***Format***	***Source/Value/Description***
RECORD-TYPE	A1	"5"
SERVICE-CLASS-CD	A3	"200"
COMPANY-NAME	A16	CI_BANK_ACCOUNT_L.DESCR
COMPANY-DISCRETIONARY	A20	Spaces

Field Name	Format	Source/Value/Description
COMPANY-ID	A10	CI_BANK_ACCOUNT.ACCOUNT_NBR
STD-ENTRY-CLASS	A3	"PPD"
CO-ENTRY-DESCR	A10	"PAYMENT"
COMPANY-DESCR-DT	A6	Business process date
EFF-ENTRY-DT	A6	Business process date
RESERVED-01	A3	Spaces
ORIGINATOR-STAT-CD	A1	"1"
ORIGIN-DFI-ID	A8	CI_BANK_ACCOUNT.DFI_ID_NUM
BATCH-NBR	N7	The batch number of this batch within the file.

Entry Detail Record

The ACH extract flat file must have one record of this type for every direct debit record.

Field Name	Format	Source/Value/Description
RECORD-TYPE	A1	"6"
TRANSACTION-CD	A2	If the amount is a debit, this is set to CI_ TENDER_TYPE.EXT_TYPE_FLG. If the amount is a credit, this is set based on the external type flag as follows: • If the value is Checking Withdrawal (**27**), this is set to **22** (Checking Deposit) • If the value is Savings Withdrawal (**37**), this is set to **32** (Savings Deposit) • If the value is Credit Card Withdrawal (**47**), this is set to **42** (Credit Card Deposit)
TRANSIT-RTG-NBR	A9	CI_APAY_SRC.EXT_SOURCE_ID
DFI-ACCT-NBR	A17	CI_APAY_CLR_STG.EXT_ACCT_ID
AMOUNT	N8.2	CI_PAY_TNDR.TENDER_AMT
INDIVIDUAL-ID	A15	CI_PAY_TNDR.PAYOR_ACCT_ID
INDIVIDUAL-NAME	A22	CI_APAY_CLR_STG.ENTITY_NAME
DISCRETIONARY-DATA	A2	Spaces
ADDENDA-REC-IND	A1	"0"
TRACE-NBR	N15	"000000000000000"

NOTE:
External Account ID. The EXT_ACCT_ID field supports up to 50 characters. If the value entered in the field is longer than that supported by the record layout (17 characters), the value will be truncated.

Company Batch Control Record

The ACH extract flat file must have one record of this type and it must follow the Entry Detail records.

Field Name	Format	Source/Value/Description
RECORD-TYPE	A1	"8"
SERVICE-CLASS-CD	A3	"200"
ENTRY-ADDENDA-CNT	N6	The total number of entry detail records in this batch.
ENTRY-HASH	N10	The product of the first 8 digits of the external source id of the auto pay source, multiplied

Field Name	Format	Source/Value/Description
		by the number of entry detail records in the batch.
TOTAL-DR-DOLLAR-AMT	N10.2	Total tender amounts of the entry detail records.
TOTAL-CR-DOLLAR-AMT	N10.2	Zero.
COMPANY-ID	A10	CI_BANK_ACCOUNT.ACCOUNT_NBR
RESERVED-01	A19	Spaces
RESERVED-02	A6	Spaces
ORIGIN-DFI-ID	A8	CI_BANK_ACCOUNT.DFI_ID_NUM
BATCH-NBR	N7	The batch number of this batch within the file.

File Control Record

The ACH extract flat file must have one record of this type and it must be the last logical record on file.

Field Name	Format	Source/Value/Description
RECORD-TYPE	A1	"9"
BATCH-CNT	N6	The number of batches in this file.
BLOCK-CNT	N6	Calculation of the total number of records in the file / 10.0 + 0.9
ENTRY-ADDENDA-CNT	N8	The total number of entry detail records in this file.
ENTRY-HASH	N10	The sum of the entry hash values on all the batch control records in this file.
TOTAL-DR-DOLLAR-AMT	N10.2	Sum of the total debit entry dollar amounts of all the batches in this file.
TOTAL-CR-DOLLAR-AMT	N10.2	Sum of the total credit entry dollar amounts of all the batches in this file.
RESERVED-01	A39	Spaces

C1-SDDCE - Download SEPA Direct Debit Collections

This process reads all auto pay download staging records marked with the **C1-SDDCE** batch control code and a given run number and creates the flat file that's passed to the SEPA clearing and settlements mechanism. Refer to *SEPA Direct Debit Record Layouts* for details of the record layouts.

NOTE: You can rerun this process. You can reproduce the flat file at any time. Simply request this job and specify the run number associated with the historic run.

If you require a different flat file format, you must create a customized extract program. This program may be used as a starting point.

Refer to *Batch Process Dependencies* for more information on when this process is run.

SEPA Direct Debit Record Layouts

The topics in this section describe the layout of the records created by *C1-SDDCE - Download SEPA Direct Debit Collections.*

NOTE: The following record layouts are based on the CustomerDirectDebitInitiation message (pain.008.001.02) specifications, as described in the Payments Maintenance Message Definition Report in the ISO 20022 standards.

Group Header Record

The SEPA direct debit flat file must have one record of this type.

Since this record includes the number of transactions processed, this record is written at the end of the file after all transaction records have been processed.

NOTE:
The base product provides a data area **C1-SEPAGroupHeader** that contains the group header record layout. All fields are alphanumeric.

The following table lists the fields that the **C1-SDDCE** extract is mapping. To see the rest of the fields, refer to the data area's schema definition.

Field Name	*Description*	*Source / Value*
MsgId	Message Identification	SEPA Master Configuration: generalMasterConfiguration/ messageIdentification
CreDtTm	Creation Date/Time	System Date/Time
NbOfTxs	Number of Transactions	Computed at the end of extract processing
InitgPty/Nm	Initiating Party -Name	SEPA Master Configuration: Derived from the Initiating Party (Person).

Payment Information Record

The SEPA direct debit flat file must have one record of this type for every obligation. It includes one or more direct debit transactions.

NOTE:
The base product provides a data area **C1-SEPADirectDebitPaymentInfo** that contains the payment information record layout. All fields are alphanumeric.

The following table lists the fields that the **C1-SDDCE** extract is mapping. To see the rest of the fields, refer to the data area's schema definition.

Field Name	*Description*	*Source / Value*
PmtInfId	Payment Information Identification	From 'Payment Information Identification' characteristic value on Obligation, where the characteristic type is defined in the SEPA Master Configuration
PmtMtd	Payment Method	'DD'
PmtTpInf/SvcLvl/Cd	Service Level (Code)	‘SEPA’
PmtTpInf/LclInstrm/Cd	Local Instrument (Code)	SEPA Master Configuration: generalMasterConfiguration/localInstrument (i.e. ‘CORE’)
PmtTpInf/SeqTp	Sequence Type	Direct Debit Payment Type of the related Direct Debit Mandate.
ReqdColltnDt	Requested Collection Date	Payment Event.Effective Date
Cdtr/Nm	Creditor Name	SEPA Master Configuration: Derived from the Initiating Party (Person).

Cdtr/PstlAdr	Postal Address	SEPA Master Configuration: Derived from the Initiating Party (Person). If an address exists: StrtNm is mapped to Address1, PstCd is mapped to Postal (prefixed with State code, if applicable), TwnNm is mapped to City and Ctry is mapped to Country.
CdtrAcct/Id/IBAN	Creditor Account Identification	Bank Account.Account Number, where Bank Account is determined from the Tender Source of the Auto Pay Route Type that is associated with the **C1-SDDCE** extract process.
CdtrAgt/FinInstnId/BIC	Creditor Agent BIC	Bank Account.DFI ID. where Bank Account is determine from the Tender Source of the Auto Pay Route Type that is associated with the **C1-SDDCE** extract process.
CdtrSchmeId/Id/PrvtId/Othr/Id	Creditor Scheme Identification - Private Identification	SEPA Master Configuration: Derived from the Initiating Party (Person) using the Creditor ID Type specified in the master configuration.
DrctDbtTxInf/PmtId/EndToEndId	Direct Debit Transaction: Payment Identification - End To End Identification	Payment Tender.Payment Tender ID
DrctDbtTxInf/InstdAmt	Direct Debit Transaction: Instructed Amount	Payment Tender.Amount
DrctDbtTxInf/DrctDbtTx/MndtRltdInf/MndtId	Direct Debit Transaction: Mandate Identification	External Mandate Reference from the active Direct Debit Mandate of the account on the tender.
DrctDbtTxInf/DrctDbtTx/MndtRltdInf/DtOfSgntr	Direct Debit Transaction: Date of Signature	Effective Date from the active Direct Debit Mandate of the account on the tender.
DrctDbtTxInf/DrctDbtTx/MndtRltdInf/ElctrncSgntr	Direct Debit Transaction: Electronic Signature	From the active Direct Debit Mandate of the account on the tender.
DrctDbtTxInf/DrctDbtTx/CdtrSchmeId/Id/PrvtId/Othr/Id	Direct Debit Transaction: Creditor Scheme Identification - Private Identification	SEPA Master Configuration: Derived from the Initiating Party (Person) using the Creditor ID Type specified in the master configuration.
DrctDbtTxInf/DbtrAgt/FinInstnId/BIC	Direct Debit Transaction: Debtor Agent's BIC	From the active Direct Debit Mandate of the account on the tender.
DrctDbtTxInf/Dbtr/Nm	Direct Debit Transaction: Debtor Name	From the active Direct Debit Mandate of the account on the tender.
DrctDbtTxInf/DbtrAcct/Id/IBAN	Direct Debit Transaction: Debtor Account	From the active Direct Debit Mandate of the account on the tender.

BALAPY - Creating Automatic Payment Tender Controls

This process creates a new tender control (with an associated deposit control) for each unique batch control and run number encountered in the extracted automatic payments (where its payment tender is not yet linked to a tender control). The payment tender of each of these automatic payments is then linked to the corresponding tender control. This process also balances the open tender control records afterwards.

Classic Bills Topics

Various topics about classic, rate-based bills are discussed in this section.

Classic Bills, Payments & Adjustments

Classic Bill Details

The following points highlight important concepts related to classic bills:

- A bill is produced for an account for one or more of its bill-based obligations. Over time, many bills may be produced for an account.
- Bills contain bill segments. A bill segment is created to levy charges for a bill-based obligation. You may configure your system to generate a single bill for a single obligation. For example you may generate one bill for your property tax obligation's charges. You may also include bill segments for multiple obligations in a single bill. For example you may generate a bill that includes bill segments for all the personal property obligations.
- Bill segments contain calculation details. A bill segment contains information showing how the segment was calculated and how it should be printed on the taxpayer's bill.
- A bill segment has a financial transaction. A bill segment has a related financial transaction. A financial transaction contains the financial effects of the bill segment on the obligation's current and payoff balances and on the general ledger.
- Canceling a bill cancels the financial transaction. If the bill segment is eventually cancelled, another financial transaction will be linked to the bill segment to reverse its original financial transaction. The cancellation financial transaction appears on the next bill produced for the account as a bill correction.
- A canceled adjustment’s financial transaction appears on the next bill produced for the account as an adjustment.
- A canceled pay segment’s financial transaction appears on the next bill produced for the account as a negative payment.

The Source Of GL Accounts On Classic Bill FTs

The following describes the source of GL accounts on classic bill financial transactions, some examples of bill related financial events, their standard accounting, and the source of distribution codes used to derive the GL accounts sent to your general ledger.

If a bill segment has a financial effect, the distribution code to debit comes from the distribution code on the obligation type; the distribution code to credit comes from the rate component(s) used to calculate the bill segment.

Financial event	***GL Accounting***	***Source Of Distribution Code***
Create a normal bill segment. *Bill Segment FT Algorithm is Payoff Amt = Bill Amt / Current Amt = Amt Due*	Debit: A/R	Obligation Type
	Credit: Revenue	Rate Component
Create a bill for company usage. *Bill Segment FT Algorithm is Payoff Amt = 0 / Current Amt = 0*	Debit: Company Usage Expense	Obligation Type
	Credit: Revenue	Rate Component
Create a bill for charity. *Bill Segment FT Algorithm is Payoff Amt=0 / Current Amt = Bill Amt*	N/A - charity bills have no effect in the GL	N/A
	N/A	N/A
	Credit: A/R	Obligation Type

Managing Classic Bill Setup

The topics in this section describe maintenance transactions related to classic billing topics. The term ‘bill’ in the following sections refers to refers only to rate based bills.

Defining Bill Segment Types

Every obligation references an obligation type. Amongst other things, the obligation type references a bill segment type. The bill segment type controls how bill segments and their related financial transactions are created.

CAUTION:
We strongly recommend understanding the concepts described in *The Big Picture of Billing* before setting up your bill segment types.

Setting Up Bill Segment Types

To set up bill segment types, open **Admin Menu** > **Bill Segment Type**.

Description of Page

Enter an easily recognizable **Bill Segment Type** and **Description** for every type of bill segment.

For each bill segment type, define the **Create Algorithm**. The logic embedded in this algorithm creates the bill segment.

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 creates a bill segment in the appropriate manner. Click *here* to see the algorithm types available for this plug-in spot.

CAUTION:
The BS Create Algorithm is a very important field as it controls how the system creates bill segments. There are some restrictions in respect of the values of certain fields on the obligation type and the bill segment algorithm used on an obligation type.

For each bill segment type, define the **Financial Algorithm**. The logic embedded in this algorithm constructs the financial transaction associated with the bill segment.

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 constructs the bill segment financial transaction in the appropriate manner. Click *here* to see the algorithm types available for this plug-in spot.

FASTPATH:
For more information about current and payoff amounts, refer to *Current Amount versus Payoff Amount*.

Define the **Pre-creation Algorithm**. The logic embedded in this algorithm retrieves detailed information needed to bill the bill segment.

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 retrieves details needed for billing in the appropriate manner. Click *here* to see the algorithm types available for this plug-in spot.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_BILL_SEG_TYP*.

Setting Up Billable Charge Templates

A user creates a billable charge whenever a taxpayer should be levied an ad hoc charge. For example, if a taxpayer requires a review of their property assessment, you may charge them an administration fee.

NOTE:
Interfacing billable charges from an external system. In addition to being entered manually, billable charges can also be interfaced from an external system. You would interface billable charges if your organization provides "pass through" billing services. Refer to *Uploading Billable Charges* for more information.

A billable charge must reference an obligation. This obligation behaves just like any other obligation:

- **Bill segments are created for the obligation.** Whenever billing is performed for an account with billable charge obligations, the system creates a bill segment for each unbilled billable charge.
- **Payments are distributed to the obligation.** Payments made by an account are distributed to its billable charge obligations just like any other obligation.
- **Overdue debt is monitored.** The credit and collections process monitors billable charge obligations for overdue debt and responds accordingly when overdue debt is detected.

NOTE:
Rates can be applied to billable charges. Billable charges can be connected to an obligation that also specifies a rate. The rate will be applied and lines added to the bill segment after the billable charge lines are added. For example, a rate can insert flat charges or be applied to service quantities associated with the billable charge.

FASTPATH:
Refer to *How To Create An Ad-hoc Bill* for instructions describing how to create a bill for a billable charge outside of the normal bill creation process.

Billable charge templates exist to minimize the effort required to create a billable charge for a taxpayer. A billable charge template contains the default bill lines, amounts and distribution codes used to levy a one-off charge.

The information on the template may be overridden by a user when the billable charge is created.

NOTE:
Templates aren't required. A billable charge can be created without a template for a truly unexpected charge.

After setting up the billable charge templates, you must indicate the obligation types that can use each template. Obviously, only **billable charge** obligation types (as defined on the obligation type's special role) will reference billable charge templates.

Billable Charge Template - Main

Open **Admin Menu** > **Billable Charge Template** to define your billable charge templates.

Description of Page

Enter a unique **Billable Charge Template** ID and **Description** for the billable charge template.

Use **Description on Bill** to define the verbiage that should print on the taxpayer's bill above the billable charge's line item details.

Use **Currency Code** to define the currency in which the billable charge's amounts are expressed.

Use the grid to define the line item details associated with the billable charge (note, the **Total Line Amount** field is automatically calculated. It is the sum of the **Charge Amount** on each of the Line Sequence items). The following fields are required for each entry in the grid.

Sequence	Line sequence controls the order in which the line items appear on the bill segment.
Description on Bill	Specify the verbiage to print on the bill for the line item.
Charge Amount	Specify the default amount to charge for the line item.
Show on Bill	Turn this switch on if the line item should appear on the taxpayer's printed bill. It would be very unusual for this switch to be off.
Appears in Summary	Turn this switch on when the amount associated with this line also appears in a summary line.
Memo Only, No GL	Turn this switch on when the amount associated with this line does not affect the GL (or the total amount owed by the taxpayer).
Distribution Code	Specify the default distribution code associated with this line item.

If you use the drill down button on the left most column in the grid, you will be taken to the Line Characteristics tab with the selected line displayed.

FASTPATH:
For more information about creating a billable charge, refer to *Maintaining Billable Charges*. For more information about billing billable charges, refer to *How To Create An Ad-hoc Bill*.

Billable Charge Template - Line Characteristics

Open **Admin Menu** > **Billable Charge Template** and navigate to the **Line Characteristics** tab to define your billable charge templates line characteristics.

Description of Page

The **Line Sequence** scroll defines the billable charge template line to which you wish to assign characteristic values.

The following fields display:

Characteristic Type	The type of characteristic.
Characteristic Value	The value of the characteristic.

Billable Charge Template - RQ Details

Open **Admin Menu** > **Billable Charge Template** and navigate to the **RQ Details** tab to define your billable charge templates service quantities.

Description of Page

The following fields display:

Sequence	Specify the sequence number of the RQ.
UOM	Select the unit of measure of this RQ. One or more of UOM, TOU, or RQ identifier must be selected.
TOU	Select the time of use period.
RQ Identifier	Select the RQ identifier.
Rate Quantity	Specify the number of units of this rate quantity.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_B_CHG_TMPLT*.

Billable Charge Line Type

NOTE:
Background information. Before using this page, you should be comfortable with the topics described under *Setting Up Billable Charge Templates* and *Uploading Billable Charges*.

Billable charge line types will simplify the effort required to interface billable charges from an external system. Each line type contains values that will be defaulted onto the line details associated with the uploaded billable charges. Obviously, this defaulting is possible only if you specify a billable charge line type on the billable charge upload staging lines.

To set up billable charge line types, select **Admin Menu** > **Billable Charge Line Type**.

Description of Page

Enter an easily recognizable **Billable Charge Line Type** and **Description**.

Use **Currency Code** to define the currency to be defaulted onto billable charge upload lines that reference this line type.

Use **Show on Bill** to define the value to be defaulted into the Show on Bill indicator on billable charge upload lines that reference this line type.

Use **App in Summary** to define the value to be defaulted into the App in Summary indicator on billable charge upload lines that reference this line type.

Use **Memo Only, No GL** to define the value to be defaulted into the Memo Only, No GL indicator on billable charge upload lines that reference this line type.

Use **Distribution Code** to define the values to be defaulted into the Distribution Code field on billable charge upload lines that reference this line type.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_BCHG_UP_XTYP*.

Setting Up Bill (Segment) Cancellation Reasons

Open **Admin Menu** > **Bill Cancel Reason** to define your bill segment cancellation reason codes.

Description of Page

Enter an easily recognizable **Cancel Reason** and **Description** for the bill cancellation reason.

Only use **System Default** on those reason codes that are placed on bill segments that are automatically canceled by the system. Valid value is **Mass Cancel**. The reason code identified as **Mass Cancel** is placed on bill segments that are canceled as a result of the execution of the Mass Cancellation background process. Refer to the **MASSCNCL** batch control (Mass Bill Cancellation (Classic) for more information.

NOTE: **Required values.** You must have one reason code defined for each of the System Default values.

Defining Bill Cycles and Bill Periods

This section discusses issues related to defining bill cycles and bill periods in the system.

An account may reference a bill cycle. An account's bill cycle may be used to control cyclical billing.

In this section, we describe how to design and set up this cycle. In addition, we discuss how to set up bill period schedules. These are used to define the bill segment end date for special types of obligations.

NOTE:
Recommendation. We recommend reading *Bill Frequency - Bill Cycle vs Bill Segment Duration* before setting up this information.

The Big Picture Of Bill Cycles and Bill Periods

The topics in this section provide background information about a variety of bill cycle and bill period issues.

Bill Cycles

The topics in this section provide background information about a variety of bill cycle features.

The Cyclical Billing Process & Window Billing

The cyclical bill creation process creates most bills. This process works as follows:

- An account can reference a bill cycle. The bill cycle's schedule controls when a cyclical billing process attempts to create bills for the account.
- Every bill cycle has a bill cycle schedule that defines the dates when a cycle's accounts are to be billed. Rather than attempt to create bills on one evening, your implementation may use a concept of "window billing" where the system attempts to produce bills for accounts over a few nights. This concept is useful if your billing is based on information being sent from an external source. It allows you to start billing accounts on the earliest possible day and then bill the stragglers over successive evenings. This results in much better cash flow.

- When the bill cycle creation process runs, it looks for bill cycles with open bill windows. It then attempts to create bills for the accounts in each such cycle. If a bill is successfully created, it is completed immediately and ready to send to the taxpayer. If a bill cannot be created, the system will create a bill in the "error" state and initiate a To Do entry with a message that can be analyzed by your billing staff. When the bill cycle creation process next runs, it deletes all "error" bills and attempts to recreate them. It continues this throughout the bill window. If bills are in error at the end of the window, they will remain in this state until a user fixes them. If the bills are still in error when the cycle's next window opens, a billing error will be generated.

Designing Your Bill Cycles

The number of bill cycles is determined by how much you want to stagger the billing of your taxpayers. You may do this to smooth out calls you may receive for your call center with questions about tax bills.

How Does An Account Get Its Bill Cycle?

Most accounts are created behind-the-scenes through taxpayer registration processing. An account created like this doesn't have a bill cycle. Rather, it sits in limbo awaiting the activation of its first obligation that should be billed using cyclical billing. When an obligation is activated, an activation algorithm should assign the account to an appropriate cycle based on business rules.

Note, an account may be configured to protect its bill cycle from being changed by an algorithm. Refer to *Protecting An Account's Bill Cycle* for more information.

NOTE:
A To Do entry highlights accounts without a bill cycle. A To Do entry highlights those accounts that require a user to specify a bill cycle. This entry is generated for accounts without a bill cycle that have active obligations.

Protecting An Account's Bill Cycle

An account has a single bill cycle, but may have multiple obligations that are billed via cyclical billing process. In this case, when a cyclical obligation is activated, you may not want an algorithm to change the account's bill cycle. To prevent this you need to turn on the account's **protect bill cycle** flag.

When the last obligation for an account is stopped, the protect bill cycle flag is reset. This is to ensure that if the taxpayer starts a new obligation in the future the bill cycle can be set based on the new obligation's activation algorithm rules.

Setting Up Bill Cycles

An account may reference a bill cycle. The bill cycle defines the schedule for cyclical billing of its accounts. To define a bill cycle and its bill cycle schedule, open **Admin Menu** > **Bill Cycle**.

Description of Page

Enter a unique **Bill Cycle** and **Description** for every bill cycle.

Use the **Bill Cycle Schedule** collection to define when bills are produced for the accounts in a given bill cycle. The following fields are required for each instance:

Window Start Date	Specify the date on which the system should start trying to create bills for accounts in the cycle.
Window End Date	Specify the last day on which the system will create bills for accounts in the cycle.

Accounting Date	Specify the financial date associated with the bills' financial transaction. The accounting date defines the financial period(s) to which the bills will be booked in your general ledger.
Freeze and Complete	Turn on this switch if the system should freeze and complete any bill that is created without errors. If this switch is turned off, all bills created by a cyclical billing process should be left in the unfinished state. You would only turn this switch off if you want to verify an entire bill run prior to freezing it (e.g., if you are introducing a new version of a rate). If you turn this off, you will need to return to this page after verifying a bill run and turn it back on for the taxpayers to receive bills. When the cyclical billing process next runs, it should delete all unfrozen bills and recreate them as per the instructions on the bill cycle schedule.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_BILL_CYC*.

Setting Up Bill Periods

Some obligation types reference a bill period. The bill period defines when the obligation's bill segments are produced and the respective end date of each bill segment.

To define a bill period and the bill period schedule, open **Admin Menu** > **Bill Period**.

Description of Page

Enter a unique **Bill Period** and a **Description** for every bill period.

Use the **Bill Period Schedules** collection when the system should create bill segments for obligations that use a given bill period. It also defines the end date of each respective bill segment. The following fields are required:

Bill Date	Specify the earliest date on which the system is allowed to create a bill segment for obligations using this bill period.
Bill Seg End Date	Specify the end date of the bill segment. For future bills, this will be after the bill date. For retro bills, this will be before the bill date.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_BILL_PERIOD*.

This information is used by the bill segment creation process to determine the end date of obligations that use a bill period.

Setting Up Bill Route Types

Bill route types define the method used to route bills to accounts. To define a bill route type, open **Admin Menu** > **Bill Route Type**.

Description of Page

Enter a unique **Bill Route Type** and **Description** for every bill route type.

Bill Routing Method controls the type of information that may be defined when the respective **Bill 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 **Bill Routing Method** are customizable using the *Lookup* table. This field name is BILL_RTG_METH_FLG.

The next two fields control how bills that are routed using this method are *printed* (both in batch and online).

- Use **Batch Control** to define the background process that performs the actual download of the billing information. Refer to *Technical Implementation of Printing Bills In Batch* for more information about these processes.
- Use **Extract Algorithm** to define the algorithm that constructs the records that contain the information that appears on a printed bill. Refer to *Printing Bills* for more information.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_BILL_RT_TYPE*.

Setting Up Bill Messages

There are various informational and warning messages that may appear on an account's bills. Each message is identified with a bill message code. To define a bill message code, open **Admin Menu** > **Bill Message**.

Description of Page

Enter a unique **Message Code** and **Description** for every bill message.

The following attributes control how and where the bill message appears on the taxpayer's bill:

Priority controls the order in which the message appears when multiple messages appear on a bill.

NOTE:
The values for this field are customizable using the Lookup table. This field name is MSG_PRIORITY_FLG.

Insert Code controls whether a document should be inserted into the bill envelope when the bill message appears on a bill.

Message on Bill is the actual verbiage that appears on the taxpayer's bill. If the message text is not static (e.g., field values need to be substituted into the body of the message), you can use the %*n* notation within the **Message on Bill** to cause field values to be substituted into a message. Refer to *Substituting Field Values Into A Bill Message* for more information.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_BILL_MSG*.

Other Financial Transaction Topics

Various topics about financial transactions are discussed in this section.

Payment Advices

The topics in this section provide background information about payment advice functionality.

NOTE:

This section is only relevant for some organizations. The system configuration requirements described in this section are only relevant if your organization issues payment advices to the taxpayer instead of initiating electronic funds transfer directly to the taxpayer's bank.

What Is A Payment Advice?

Payment advice is a money order that is established at the initiative of the tax authority. When a debt is incurred, the tax authority sends the taxpayer a document that indicates a payment amount and the taxpayer's bank details. If the taxpayer agrees to the information on the payment advice, he or she signs it and returns it to the clearinghouse address that is indicated on the payment advice. The clearinghouse, in turn, sends the dated and signed payment advice to the taxpayer's bank, which completes the payment.

Payment Advice vs. Direct Debit

The existing functionality that creates automatic payments is referred to as direct debit processing. Payment advice processing differs from direct debit processing in the way that automatic payments get initiated. With payment advice processing, the usual automatic payment records - i.e. payment event, payment, tender and auto pay clearinghouse staging - are not created. Instead, a payment advice is printed and sent to the taxpayer. The taxpayer sends the approved payment advice directly to the clearinghouse.

The system does not provide sample processes for extracting and printing payment advice information. Your implementation team would have to create these.

Setting Up The System To Enable Payment Advice

You must set up a *Feature Configuration* to define parameters that control payment advice functionality. Find the Feature Configuration record for the **Financial Transaction Options** feature type. (It may need to be defined if it does not exist).

The following points describe the various **Option Types** that must be defined:

- **Payment Advice Functionality Supported.** This option controls whether the system allows for payment advice processing.
- Enter **Y** if the system should allow for both direct debit and payment advice processing.
- Enter **N** if the system should only allow for direct debit processing.
- **Default Auto Pay Method.** This option is used for defaulting the auto pay method on new account auto pay records.

Refer to *Account - Auto Pay* for more information on auto pay method.

NOTE:
The system assumes direct debit processing if the above feature options are not defined.

Payment Event Distribution

The base package, by default, creates a single payment for a payment event. If your business requires potentially many payments to be created when payment events are added, you'll need to set up the various control tables described in this section.

FASTPATH:

Refer to *Distributing A Payment Event* for more information about how payment event distribution is handled in the system.

Making Payments Using Distribution Rules

As part of this method, one or more distribution details are provided at payment time along with the usual payment and tender information. Each distribution detail record references a distribution rule and a corresponding value. The distribution rule encapsulates the business rules that govern the distribution of the payment amount into payments using the specified value.

The type of value being captured on the distribution detail and the logic that uses it to create payments are defined on the *distribution rule*.

Rule Value

The primary use of the rule value is to identify the business entity whose balance is to be relieved by creating payment(s). In most cases where the payor account is the same as the payee account it may also used to identify the tender account associated with the payment(s).

Determine Tender Account

The very first step in processing a distribution detail is to identify the tender account (i.e. the payor) associated with the payment. To do that the system calls the **Determine Tender Account***algorithm* defined on the distribution rule providing it with the rule value and other tender information.

Creating Payment(s)

The business logic that distributes a payment amount into one or more payments(s) targeted towards the entity identified by a rule value is held in designated **Create Payment***algorithms* defined on the distribution rule.

Rule Value Can Capture Additional Information

A rule value can also be used to capture additional information provided at payment time, like address information, comments, etc. Obviously payment distribution details with this type of rule value should have a zero payment amount, as they are not real payments. These distribution details end up being linked to a payment event, but unlike other distribution details they do not contribute any payments. You can think of these details as payment event characteristics.

You don't have to set up a **Create Payment** algorithm for distribution rules intended solely to capture additional payment information.

Payments Linked To Assessments

When dynamic credit allocation is practiced, a payment targeted to a specific assessment must have the payment FT referencing the assessment's FT ID. If there is excess credit on the payment segment, it is dynamically allocated to any other assessments for the obligation.

The base package provides a payment freeze algorithm *C1-PYFZ-PYAS* that links a payment FT to a specific assessment. This algorithm looks for a match value on the payment that references the assessment. The assessment is linked to the payment by populating the assessment's FT ID on the payment FT's Group FT ID field.

For more information, refer to *Credit Allocation*.

Setting Up The System To Use Distribution Rules

Setting Up Distribution Rules

Define a Distribution Rule for each payment event distribution method practiced by your business.

Distribution Rule - Main

To set up a distribution rule, **Admin Menu** > **Distribution Rule**.

Description of Page

Enter a unique **Distribution Rule** and **Description** for the distribution rule.

Provide a short and unique **Distribution Rule Label** to be used as the rule's name throughout the system.

Characteristic Type defines the type of entity whose balance is relieved by the payments this rule creates. For example, if this rule targets payments towards a specific obligation, you'd reference a characteristic whose value identifies an obligation. We use the term "rule value" for the characteristic value.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_DST_RULE*

Distribution Rule - Algorithms

Navigate to **Admin Menu** > **Distribution Rule** and navigate to the **Algorithms** tab to set up the algorithms appropriate for your distribution rule.

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 events are optional).

System Event	***Optional / Required***	***Description***
Create Payment	Optional	This algorithm is executed to distribute a payment distribution detail payment amount into one or more payments. Click *here* to see the algorithm types available for this system event.
Determine Tender Account	Optional	This algorithm is executed to determine the tender account associated with the payment distribution detail. Only one such algorithm may be specified. Click *here* to see the algorithm types available for this system event.

Feature Configuration

You must set up a *Feature Configuration* to define parameters that control various payment event distribution options. Find the Feature Configuration record for the **Financial Transaction Options** feature type. (It may need to be defined if it does not exist).

The following points describe the various **Option Types** that must be defined:

- **Always Enable Distribution Rule.** This option controls whether the system should only use the distribution rule method to add payment events or rather allow both the default method and the distribution rule method to coexist.
    - Enter **Y** if the system should always use distribution rules. With this setting, navigation to the Payment Event page in add mode opens up the *Payment Event Quick Add* page (defaulting it to the **single** payment event dialog). This dialog is designed to create a payment event using distribution rules.
    - Enter **N** if the system should allow both methods. With this setting, navigation to the Payment Event page in add mode opens up the standard *Payment Event - Add Dialog* that uses the default method to create a payment event. If you want to use the distribution rule method, navigate to the Payment Event Quick Add page from the menu.
- **Default Distribution Rule.** This option states distribution rule to be used by default.

Set Up Account Type

If you do dynamic credit allocation, set up the payment freeze algorithm *C1-PYFZ-PYAS*.

We recommend specifying **payment distribution** and **overpayment** algorithms because they are used as default logic in case distribution rules are not supplied.

Set Up Match Type

If you use set the payment freeze algorithm *C1-PYFZ-PYAS*, set up a match type for assessment. This match type should NOT reference an override payment distribution algorithm (if this algorithm is blank, the account type's payment distribution algorithm is used). Refer to *Payments Linked to Assessments* for more details.

Payables Cash Accounting

In some cases certain amounts such as sales tax distributions and 3rd party liabilities are not truly payable until the taxpayer remits payment. We refer to this as "payables cash accounting". This practice should be contrasted with "payables accrual accounting" in which the liability is realized when the debt is recorded (as opposed to when it is paid).

If your organization does not practice payable cash accounting, you may skip this section as accrual accounting is the system default. If you practice payables cash accounting, the contents of this section describe how to configure the system appropriately.

Accrual versus Cash Accounting Example

Accrual accounting means that all payables are booked when the debt financial transaction is created.

If the payable is subject to payables cash accounting, the liability is not booked when the financial transaction is created, rather, the amount of the liability is placed in a "holding" GL account. When the taxpayer pays, the moneys are transferred from the "holding" GL account to the true tax payable account.

The following is an example of the financial events that transpire when a simple sales tax assessment is posted and a payment is received using accrual accounting.

Event	*GL Accounting*	*County A Payable Balance*
Tax assessment created	A/R 110 Revenue - State <100> Payable - County A <10>	(10)
Payment received	Cash 110 A/R <110>	(10)

In the above example, you'll notice that the payable is booked when the adjustment is created. Let's contrast this with what takes place if the payable is subject to payables cash accounting.

Event	*GL Accounting*	*County A Payable Balance*	*County A Holding Balance*
Tax assessment created	A/R 110 Revenue - State <100> Holding - County A <10>	0	(10)
Payment received	Cash 110 A/R <110> Holding - County A 10 Payable - County A <10>	(10)	0

Notice that when the adjustment is created, the payable for the county is not booked, rather, the amount is placed in a "holding" GL account. When the taxpayer pays, the moneys are transferred from the "holding" GL account to the true **County A** payable account.

If the above seems simple, consider the following complications that must be considered:

- What happens if a partial payment is received?
- What happens if there are multiple amounts subject to cash accounting rules?
- What happens if the payment is cancelled?
- What if the payment isn't received and we have to write-off debt?
- What happens if the taxpayer overpays?
- What happens if the obligation is subject to penalty and interest and dynamic credit allocation?

The above points, and more, are discussed below.

Distribution Code Controls Cash Accounting For A GL Account

NOTE:
If you do not understand the significance of distribution codes, please refer to *Setting Up Distribution Codes*.

Whether or not cash accounting is used for a specific GL account is defined on HOLDING GL account's distribution code (i.e., the holding GL account references the true payable account).

It is very important that unique payable and holding distribution codes be used for each type of amount subject to cash accounting rules. For example, for sales tax distribution, a pair of payable and holding accounts is required for each separate distribution amount.

Without unique distribution codes for each payable and holding account, the system cannot keep track of how much of a given amount is being held, awaiting payment.

Cash Accounting and Incurring Debt

When calculating debt that includes payables, the distribution codes associated with the payables (for example on the adjustment type) must be the holding payable distribution codes. No other logic is needed for cash accounting at this time.

Cash Accounting and Relieving Debt

The following section describes functionality required for obligations that practice cash accounting and do NOT practice penalty and interest or dynamic credit allocation.

CAUTION:
Dynamic Credit Allocation. Refer to *P&I and Cash Accounting* for functionality required for obligations that do practice penalty and interest or dynamic credit allocation.

Payment Segment Financial Transaction Algorithms Transfer Holding Amounts to Payable GL Accounts

Logic exists in the pay segment's FT algorithm that transfers amounts from payable holding distribution codes to their respective payable real distribution codes.

FASTPATH:
Refer to *Setting Up Payment Segment Types* for how to define the appropriate FT algorithm.

The following table shows what happens to the financial transaction associated with the payment segment for a cash accounting taxpayer.

Event	*GL Accounting*
Adjustment is created	A/R 110 Revenue - State <100> County Holding <10>
Payment segment relieves receivables	Cash 110 A/R <110>
Additional GL details created when the payment segment FT algorithm transfers the holding amount to a payable account	County Holding 10 County Payable <10>
Net affect of the above	Cash 110 A/R <110> County Holding 10 County Payable <10>

How Does The System Know What Amounts To Transfer From Holding To Payables?

When a payment segment is created for an account that is subject to cash accounting processing, the system determines if there is a credit balance for any holding distribution code in respect of the obligation. If so, it generates additional GL details to transfer moneys from the holding distribution code to the payable distribution code in proportion to the amount of receivables relieved by the payment. Therefore, if 100% of receivables are relieved by the payment segment, 100% of the holding amounts will be transferred to payable distribution codes. Refer to *Partial Payments Result In Partial Payables* for an example of what happens when a partial payment is created.

Partial Payments Result In Partial Payables

The previous example showed the entire County holding amount being transferred to the County payable account. The entire holding amount was transferred because the obligation was paid in full. If a partial payment is received, only part of the holding amount will be transferred to the payable amount (proportional to the amount of receivables reduced by the payment). An example will help make the point.

Event	*GL Accounting*	*County Payable Balance*	*County Holding Balance*
Adjustment created	A/R 110 Revenue - State <100> County Holding <10>	0	(10)
Partial payment received	Cash 27.50 A/R <27.50> County Holding 2.50 County Payable <2.50>	(2.50)	(7.50)

Adjustments That Behave Like Payments

There are several types of adjustments that behave just like payments (in respect of payables cash accounting), for example an overpayment transferred one obligation to another

The above events should cause the system to transfer holding amounts to true payable amounts (notice that the above examples are all transfer adjustments).

However, there are many other adjustments that should not behave like payments. You control how the adjustment works by selecting the appropriate FT algorithm when you *set up adjustment types* (refer to *ADJT-AC* and *ADJT-TC* for a description of the base package algorithms that causes the holding amounts to be manipulated in proportion to the amount of receivable being adjusted). In other words, there are adjustment FT algorithms that cause the transference of holding payable amounts to real payable amounts when the A/R balance is decreased by the adjustment.

Open Item Accounting

The topics in this section provide background information about open-item accounting.

NOTE:
This section is only relevant for some organizations. The system configuration requirements described in this section are only relevant if your organization practices open-item accounting. If your organization practices balance-forward accounting, you need only indicate such on your *account types*; no other setup is required. Refer to *Open Item Versus Balance Forward Accounting* for more information about these two accounting practices.

Open-Item Versus Balance-Forward Accounting

If you practice open-item accounting, you match payments against bills. The term "open-item accounting" is used to describe this accounting practice because:

- Payments are matched against "open items" (i.e., unpaid bills and adjustments)
- Only unmatched bills and adjustments (i.e., open items) affect aged debt.

Contrast open-item accounting with "balance-forward" accounting - in a balance-forward world, payments are not matched to bills. Rather, payments implicitly relieve a taxpayer's oldest debt.

Accounting Method Defined On Your Account Types

You define the type of accounting method that is practiced (*balance-forward versus open-item*) on your *account types*. The account type should be configured based on the accounting method practiced for that tax type.

Match Events

Match events are used to match open-items (i.e., debit and credit financial transactions) together. The topics in this section provide an overview of match events.

Match Events Match Debit FTs To Credit FTs

For open-item taxpayers, the system matches credit financial transactions (FT's) to debit FT's under a *match event*. The following are important points about match events.

- A match event may contain an unlimited number of FT's. For example, the match event can match 2 credit FTs against a single debit FT.
- A match event contains FTs associated with a single account. While the FTs under a match event may belong to multiple obligations, all FTs under a match event must belong to the same account.
- The status of the match event is **balanced** when the sum of the debits equals the sum of the credits. If debits do not equal credits, the status of the match event is **open** and the various FTs would still affect the taxpayer's aged debt. Refer to *Match Event Lifecycle* for more information.

When Are Match Events Created?

The following points describe when match events are created for open-item accounts:

NOTE:
Match events are only created for open-item accounts (i.e., those accounts with an account type that indicates open-item accounting is practiced). Match events may not be created for balance-forward accounts.

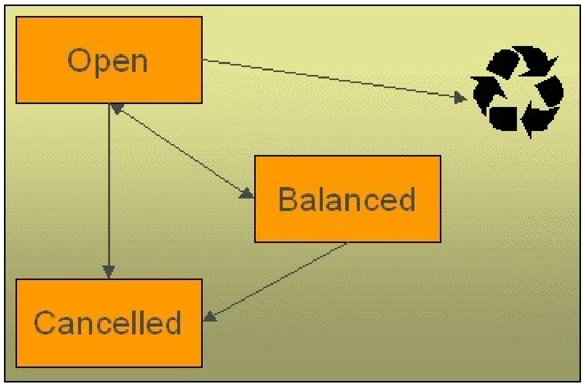
- The system can create one or many match events when a payment is added. This match event matches the payment's credit FTs with the debit and credit FTs from bill segments and adjustments. The FTs that are linked to the match event are controlled by the payment's **match type** and **match value** (payments made by open-item taxpayers must reference a match type and match value). Refer to *Payments And Match Events* for more information.
- The system may create a match event when any type of financial transaction is cancelled. This match event groups together the original FT with its cancellation FT. Refer to *How Are Match Events Cancelled?* for more information.
- The system creates a match event when a bill is completed for taxpayers that pay automatically (i.e., direct debit taxpayers). The match event groups together the bill's new charges against the automatic payment's payment segments.
- The system creates a match event when a bill is completed where the new charges are offset by other financial transactions.

FASTPATH:
Refer to *Bill Lifecycle* for more information about what happens during bill completion.

- The system creates a match event when an obligation closes and the obligation has unmatched FTs. For example, consider an obligation for debt that is written off. This obligation closes when the system creates transfer adjustments to transfer the debt to a write-off obligation (or writes down the debt). The system creates a match event to match the original debt to the transfer adjustments used to write-off the debt.
- A user can create a match event manually at any time. Manual match events would be created under a variety of situations. For example:
- If a taxpayer disputes a charge. Refer to *Disputing Items* for more information about disputes.
- To handle unusual situations when the system is unable to automatically match FT's together.

Match Event Lifecycle

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



Match events are initially created in the **open** state. Financial transactions (FT's) linked to **open** match events affect arrears, but not in an open-item fashion. Rather, FT's linked to **open** match events affect arrears in a balance-forward fashion. Refer to *Open Item Versus Balance Forward Accounting* for more information about these two accounting methods.

A user may delete an **open** match event. When an **open** match event is deleted, its FT's may be linked to other match events.

The system automatically changes an **open** event's status to **balanced** when the sum of the debit financial transactions (FT's) equals the sum of the credit FT's for each obligation on the match event. It's worth stressing that a match event may contain FT's from many obligation's and each obligation's FT's must sum to zero before the match event can become **balanced**.

A user may re**open** a **balanced** event (by adding / removing FT's so that the match event becomes unbalanced).

A user may **cancel** a **balanced** or **open** match event. Refer to *How Are Match Events Cancelled?* for more information about cancellation.

Payments And Match Events

As described under *When Are Match Events Created?*, the system creates a match event when a payment is added for an open-item account. The system uses the payment's **match type** and **match value** to determine the FT's (e.g., bill segments and adjustments) that will be matched with the payment's FT's (i.e., the payment segments).

Another way to think of this is as follows:

- When most payments are distributed, the system calls the payment distribution algorithm that is plugged-in on the account's account type.
- However, a payment that is made in respect of a specific bill requires a different distribution algorithm because the payment should only be distributed amongst the debt associated with the specific bill being paid. This is accomplished by

referencing a match type / match value on the payment. The match type references the appropriate payment distribution algorithm. This algorithm is used rather than the account type distribution algorithm.

For example, if a payment were made in respect of bill ID **192910192101**, this payment would reference a match type of **bill ID** and a match value of **192910192101**. At payment distribution time, the system calls the override payment distribution algorithm associated with this match type. The base package bill ID distribution algorithm does several things:

- It distributes the payment amongst obligations associated with the bill.
- It creates a match event and links the bill's bill segment and adjustment FT's to it.
- Refer to the *Bill ID Match Type Algorithm* for more information about this algorithm.

NOTE:
The match type's distribution logic is not "hard coded". Because the match type's payment distribution logic is embedded in a plug-in algorithm, you can introduce new algorithms as per your company's requirements.

It's worth noting that payment *distribution* and *freezing* are two separate steps that typically happen in quick succession. The system's standard match event algorithms create the match event during payment distribution. This match event exists in the **open** state (because the payment segment's FT's have not yet been linked to the match event and therefore debit FT's do not equal credit FT's). The **open** match event references the debit FT's (the bill segments and adjustments) for which it pays. It is only at payment freeze time that the credit FT's (the payment segments) are linked to the match event thus allowing the match event to be become **balanced**.

If, at freeze time, the payment's credit FT's do not equal the debit FT's on the match event, the match event is left in the **open** state. An alert will appear in the alert zone to highlight the existence of **open** match events (if the appropriate alert algorithm is plugged in the installation record). In addition, you can also set up a To Do entry to highlight the existence of open match events.

How Are Match Events Cancelled?

A user can cancel an **open** or **balanced** match event at any time. When a match event is **cancelled**, the event's FT's again affect arrears (and they can be associated with new match events). In other words, when a match event is **cancelled**, its FT's are released from the match event and become open-items.

In addition to manual cancellation, the system may automatically cancel a match event when the last of its payment FT's, if any, is cancelled (if you plug-in the appropriate FT freeze plug-in on your open-item account types).

For example, consider a match event that was created when a payment was made. If the payment is subsequently cancelled, the match event is also cancelled (thus releasing the match event's FT's) if no other payment FT's are linked to the match event. Please be aware that FT cancellation also causes a new match event to be created. This match event matches the original FT (the payment segment) and its cancellation FT. This means that the only "open items" that will exist after a payment is cancelled are the debit FT's that were originally paid.

NOTE:
Reopening bills associated with automatic payment taxpayers. While many payments are cancelled due to non-sufficient funds, please be aware that if you reopen a bill for which an automatic payment was created, the system will cancel the associated payment. If this payment is associated with a match event (because the account is an open-item account), the match event will be cancelled and a new match event will be created to match the original automatic payment with its cancellation details. This is necessary because a new payment will be created with the bill is subsequently completed and this payment's FT's will be matched to the bill's FT's.

NOTE:
Canceling a payment can result in many match events being created. If a cancelled payment has multiple payment segments, a separate match event will be created for each payment segment.

While payment cancellation is the most common type of FT cancellation, be aware that bill segment or adjustment cancellation may also cause a new match event to be created. We don't necessarily want to always link the cancellation FT and its original FT to the same match event. For example, when the cancellation FT is swept on to the next bill it affects the next bill and not the original FT's bill. For cancellations that will not be swept on to the next bill (payment cancellation, cancellation of an adjustment that is not shown on bill, and bill segment cancellation before the bill is completed) the system creates a new match event that matches the original FT and its cancellation FT. This way, neither FT affects aged debt.
If the original FT was linked to an existing match event and no other FTs are left on this match event it is automatically canceled.

Current Amount Is Matched, Not Payoff

The system matches the current amount of financial transactions, not the payoff amount.

FASTPATH:
Please refer to *Current Amount versus Payoff Amount* for more information about current and payoff amounts.

Disputing Items

Open-item taxpayers may dispute FT's that they are not comfortable paying. For example, a taxpayer who receives a bill with an anomalous charge may decide to dispute it.

When an open-item taxpayer disputes a charge, a user creates a match event and links the disputed FT(s) to it. This match event will be in the **open** state (because it does not contain FT's that sum to zero). In addition, the match event's "disputed switch" is turned on.

NOTE:
Alerts. An alert is displayed in the alert zone to highlight the existence of disputed match events (if the appropriate alert algorithm is plugged in). In addition, you can also set up a To Do entry to highlight the existence of disputed match events.

While the dispute is being researched, the disputed amount will not affect aged debt, but it still forms part of the taxpayer's balance.

If the dispute goes in your company's favor, the disputed match event should be **cancelled** (thus allowing the FT's to again impact aged debt).

If the dispute goes in the taxpayer's favor,

- You may decide to cancel the offending bill segment(s) / adjustment(s). As described above, these cancellations are going to be swept on to the next bill. The system therefore will not automatically cancel the disputed match event. Notice that the cancellation effect of the disputed items is carried over on to the next bill. This means that the previously disputed items still need to be paid.

  NOTE:
  Cancel / rebill. If you cancel / rebill an offending bill segment, both the cancel and the rebill will become open-items that will be matched when the next bill is paid.

- You may decide to issue an adjustment to counter the effect of the disputed FT's. In this situation, you would simply link the adjustment FT to the disputed FT's (thus allowing the match event to become **balanced**). It is important to use in this case an adjustment that does not show on bill.

Overpayments

An overpayment, by definition, does not "match" to open items. However, the match type algorithms supplied with the base package will result in a **balanced** match event if an overpayment is made. The following points explain how this is achieved:

- The base package's match type algorithms will distribute the payment until the taxpayer's current debt is satisfied.
- The amount of the overpayment will be kept on a separate obligation (this only happens if you plug-in the appropriate Overpayment Distribution algorithm on your account types). Refer to *Overpayment Obligations* for more information.
- When the payment is frozen, the payment segments that satisfy current debt will be matched against their respective open-items. The payment segment used to book the overpayment (on the overpayment obligation) will not be matched.
- When future bills are completed, the credit balance on this "overpayment obligation" will be transferred to the "real obligation's" when future bills are completed (if you have plugged in the appropriate bill completion algorithm on the overpayment obligation's obligation type). If the overpayment satisfies all newly calculated charges, a match event is created that matches the new charges against the funds transferred from the overpayment obligation. Refer to *When Are Match Events Created* for information about how the system creates match events at bill completion time when the new charges on the bill are satisfied by other credits such as overpayments.
- At some point in the future, the overpayment will be exhausted (i.e., all funds will be transferred to "real obligations"). At that point in time, the overpayment obligation will close (assuming you set up the overpayment obligation's obligation type as a "one time"). At close time, the system creates a match event that matches the original overpayment payment segment with the adjustments that were used to transfer funds to the "real obligation's". Refer to *When Are Match Events Created* for information about how the system creates match events when an obligation closes.

Setting Up The System To Enable Open Item Accounting

The following section provides an overview of how to enable open-item accounting.

Match Type Setup

The number of match types that you will need is dependent on the number of ways you want payments to be matched to open items. At a minimum, you will probably need the following match types:

- **Bill ID.** This match type should reference an override payment distribution algorithm that distributes the payment based on the bill ID specified on the payment (in match value). Refer to *Payments And Match Events* for more information.
- **Obligation ID.** This match type should reference an override payment distribution algorithm that distributes the payment based on the obligation ID specified on the payment (in match value). Refer to *Payments And Match Events* for more information.
- **Pay Plan.** This match type should NOT reference an override payment distribution algorithm (if this algorithm is blank, the account type's payment distribution algorithm is used). Refer to *Pay Plans* for more information.

Match Event Cancellation Reason Setup

The number of match event cancellation reasons that you will need is dependent on the number of ways your organization can justify the cancellation of a match event. At a minimum, you will probably need the following match event cancellation reasons:

- **FT Cancellation.** This cancel reason should be referenced on the account type FT Freeze algorithm that is responsible for canceling match events when one of its financial transactions is cancelled.

- **Incorrect Allocation.** This cancel reason should be specified by users when they cancel match events that were created by the system erroneously.

Account Type Setup

The following points describe *account type* oriented set up functions:

- Turn on the open-item accounting switch.
- Set up the following algorithms for each division:
- Specify a **payment freeze** algorithm that causes a payment's FT's to be linked to the match event that was created when the payment was distributed. Refer to *Payments And Match Events* for more information.
- Specify a **FT freeze** algorithm that causes match events to be cancelled (and a new match event to be created) when a FT is cancelled. Refer to *How Are Match Events Cancelled* for more information about cancellation.
- We strongly recommend specifying an **overpayment** algorithm that causes overpayments to be segregated onto an "excess credit / overpayment" obligation. Refer to *Overpayments* for more information.

Overpayment Obligation Type Setup

Specify a **bill completion** algorithm that causes the credit amount on overpayment obligation's to be transferred to newly create debt (created when the bill is created). This algorithm transfers an overpayment obligation's balance to regular obligation's and creates a match event if the overpayment covers the entire bill. Refer to *Overpayments* for more information.

Installation Record Setup

Specify an **automatic payment** algorithm that causes a match event to be created when automatic payments are created for open-item accounts. The base package algorithm will do this for you if you specify the appropriate parameter on the algorithm. Refer to *APAY-CREATE* for more information about this algorithm.

If you want an alert to highlight when the current account has any open match events, plug in the appropriate **Alert** algorithm on your installation record. Refer to *C1-OPN-MEVT* for more information about this algorithm.

If you want to enable manual pay segment distribution for open item accounts, along with other functions, you will need to plug in an installation algorithm for bill balance calculation. Refer to *C1-OI-BI-AMT* for more information about this algorithm.

To Do Entry Setup

Two To Do types are supplied with the base package:

- **TD-MODTL**. This To Do type highlights the presence of open, disputed match events.
- **TD-MONTL**. This To Do type highlights the presence of open, non-disputed match events.

Each of the above To Do types should be configured with the roles that work on entries of each type.

In addition, the account management group and/or divisions from which the default roles are extracted should be updated to define the role that should be defaulted for each of the above To Do types.

FASTPATH:
Refer to *The Big Picture Of To Do Lists* for more information about To Do lists.

Setting Up Match Types

Most payments are distributed amongst obligations using the payment distribution algorithm specified on the payment's account's account type. This algorithm decides how to distribute a payment amongst an account's existing debt if the taxpayer doesn't specify how the payment should be distributed.

A taxpayer can specify how a payment is distributed by specifying a match type and match value on their payments. Consider the following examples:

- Taxpayers that are subject to open-item accounting (this is defined on the account's account type) tell the system exactly which debt is covered by their payments. For example, an open-item taxpayer might make a payment in respect of bill ID **123919101919**.
- Even non open-item taxpayers can direct payments to specific obligations. For example, the system allows a balance-forward taxpayer's payment to be directed to a specific obligation (however, they cannot direct payments to specific bills as only open-item taxpayers can do this).

Match types are used to define the specific type of debt that is covered by a payment. The match type contains the algorithm that effectively overrides the standard payment distribution algorithm defined on the account's account type.

NOTE:
Background information. Please refer to *Payments And Match Events* and *Match Type Setup* for more information about how match types are used.

To set up match types, select **Admin Menu** > **Match Type**.

Description of Page

Enter an easily recognizable **Match Type** and **Description**.

Define the **Pay Dist Override Algorithm** used to distribute payments that reference this match type. 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 overrides the normal payment distribution algorithm.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_MATCH_TYPE*.

Setting Up Match Event Cancellation Reasons

When a match event is canceled, a cancel reason must be supplied.

NOTE:
Background information. Refer to *How Are Match Events Cancelled?* and *Setting Up Match Event Cancellation* for more information about cancellation.

To set up match event cancellation reasons, select **Admin Menu** > **Match Event Cancel Reason**.

Description of Page

Enter an easily recognizable **Match Event Cancel Reason** and **Description**.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_MEVT_CAN_RSN*.

Fund Accounting

The topics in this section provide background information about fund accounting.

NOTE:
This section is only relevant for some organizations. The system configuration requirements described in this section are only relevant if your organization practices fund accounting. If your organization does not practice fund accounting, you need only indicate such on the *Installation Record*; no other setup is required.

Fund Accounting Overview

Regulations or other restrictions may require some organizations to account for the finances of each of its departments as a separate entity.

To track the finances of departments separately, the organization sets up a fund for each department. A fund is an accounting entity with its own self-balancing set of accounts. Each fund has its own "sub general ledger" with its own chart of accounts, and within each fund, its debits equal its credits at all times. This allows the organization to report on the financial state of each fund independently.

In addition to having a fund for each department, there is also a general fund, which is used to handle inter-fund transfers as well as shared accounts.

A single obligation may have debt related to different funds. For example, tax liability and penalty and interest may go to the same "revenue" fund, and a collection fee may go to a "collection" fund.

In fund accounting, debits and credits must balance for the whole general ledger and debits and credits within each fund must balance.

Fund Accounting Example

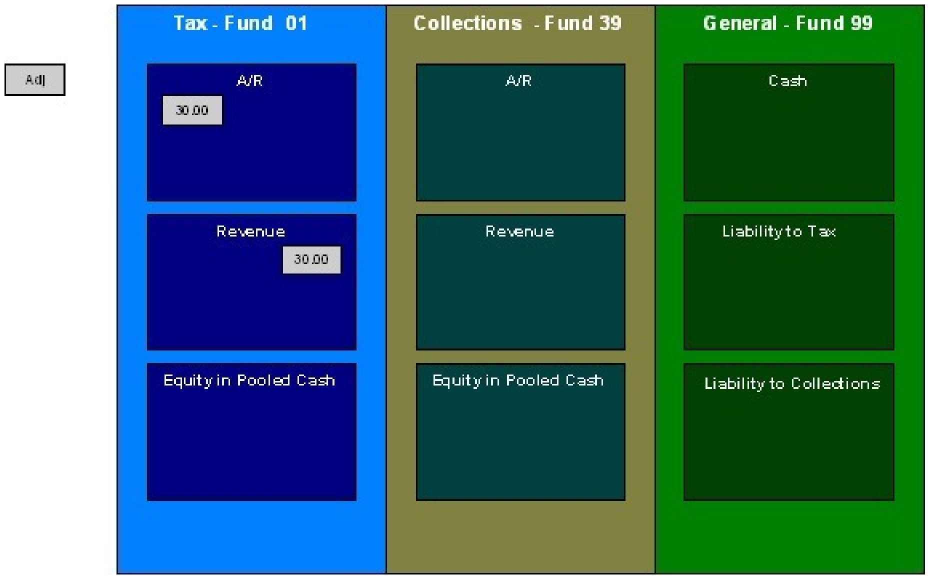
Consider an obligation that owes tax liability and then incurs a collection fee. Note that penalty and interest would also accrue, but are not shown in this example.

The accounts receivable (A/R) distribution code to use for financial transactions that affect A/R is linked to the obligation type. The assumption is that the fund for this distribution code is the same "revenue" fund.

For this example, three funds exist:

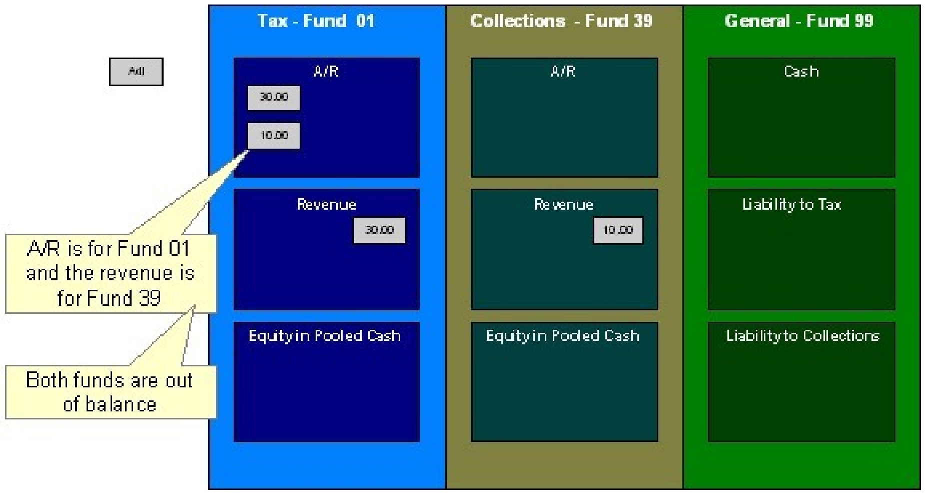
- Tax (fund 01)
- Collections (fund 39)
- General (fund 99)

When a tax is levied, the following adjustments are made to the tax fund account producing the following GL entries.



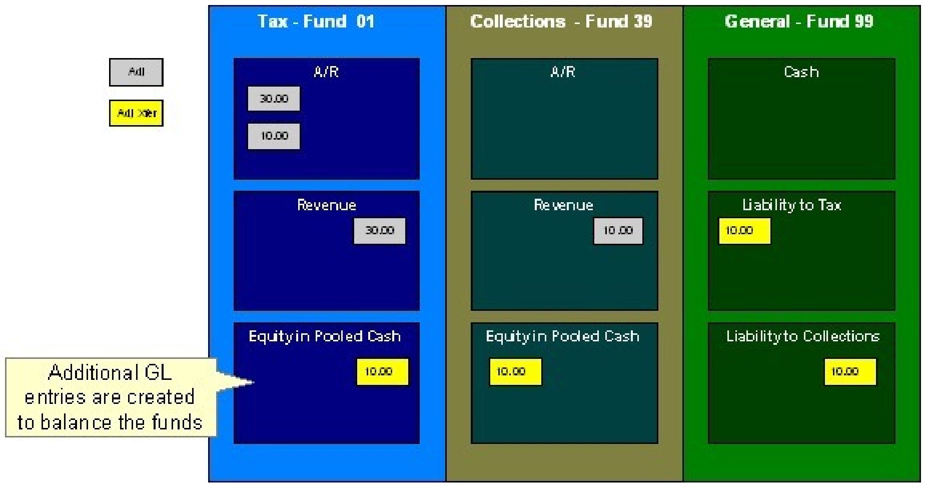
For the tax fund, the GL details of the bill include a debit to the accounts receivable (A/R) account and credits to the revenue account. The taxing authority is owed the entire portion of the adjustment by the taxpayer. The tax fund is balanced.

The following diagram illustrates the initial GL accounting that occurs when a collection fee is levied.

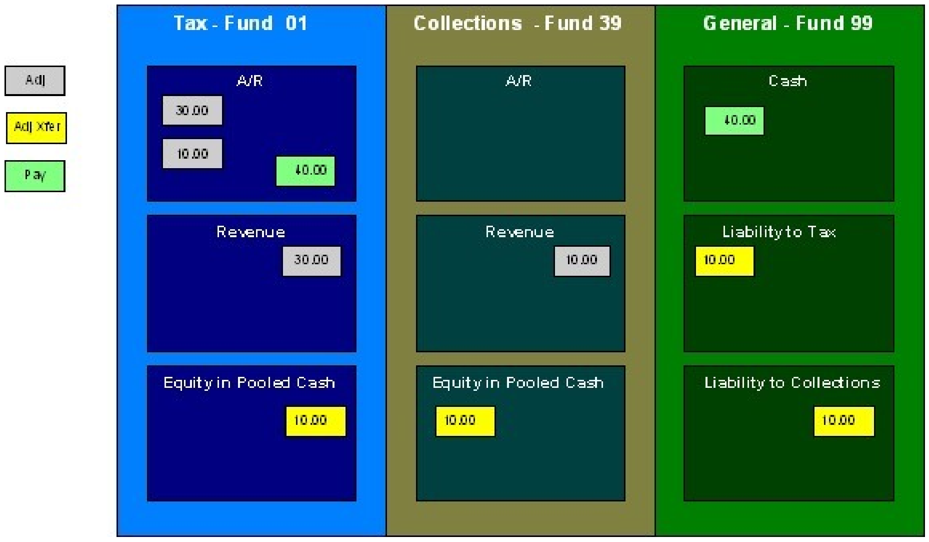


Because A/R is added to the tax fund and the revenue is added to the collections fund, the funds are out of balance.

As the tax authority takes on the responsibility to collect the fee, additional entries are created to balance the funds.



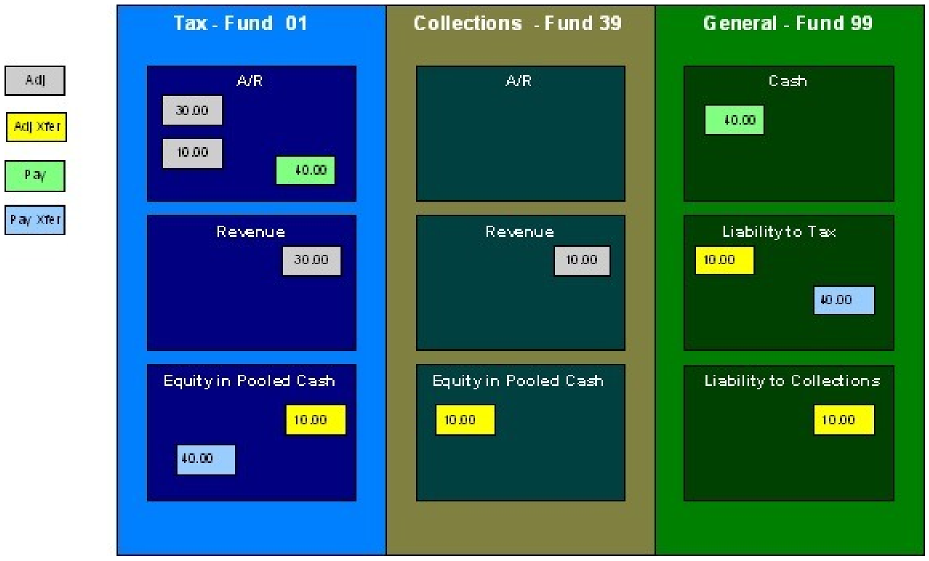
The following diagram illustrates the initial GL accounting that would occur when the payment arrives.



The tax authority's general cash account is debited, and the tax fund's A/R account is credited.

If the accounting were left in this state, the fund accounting principal - that each fund represents an independent entity with a self-balancing chart of accounts - would be violated. This violation is caused due to the fact that cash is recorded on the general fund, not the tax fund, causing the general fund to have an excess debit and the tax fund to have an excess credit.

From an organizational viewpoint, to make the tax fund whole, the tax fund needs to note what portion of the cash it owns, and correspondingly, the tax authority needs to note what portion of the cash is owed to each fund. The following diagram illustrates this point.



To maintain a balance of debits and credits within each fund, the tax and collection funds have an "equity in pooled cash" (EPC) account and the general fund has a liability account for each fund. In addition to debiting the general fund's cash account and crediting the tax fund's A/R accounts, the tax and collection funds' EPC accounts are debited and the general funds liability accounts are credited.

And so, with the additional GL entries, all funds have matching debits and credits.

An Example Of A Bill Segment That References Multiple Funds

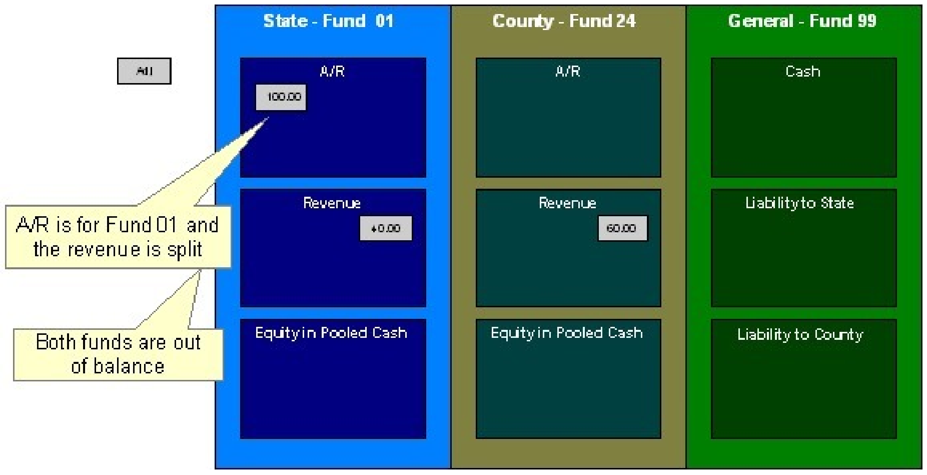
Consider a tax liability that is distributed to multiple jurisdictions where 60% is allocated to the county where the store is located and 40% is allocated to the state.

Note that in this example only two jurisdictions are used; however, a real scenario may break this down further into town, school districts, and so on.

For this example, three funds exist:

- State (fund 01)
- County (fund 24)
- General (fund 99)

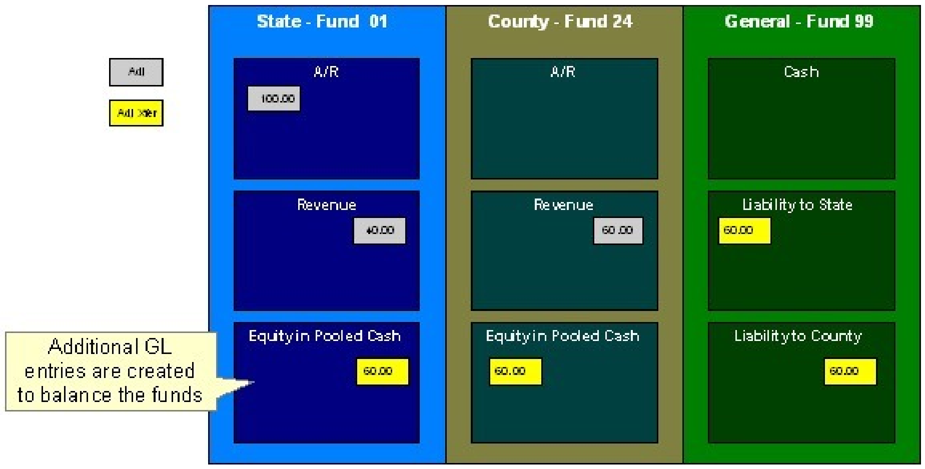
When a $100.00 tax is levied, the following diagram illustrates the initial GL entries for this scenario.



The state fund's A/R is debited, and the state and county funds' revenue accounts are credited.

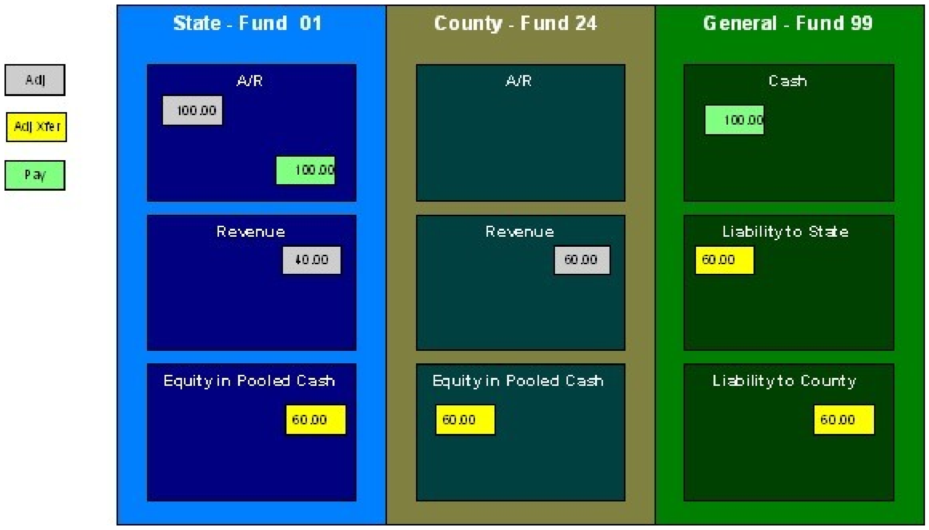
If left at this, the funds would be out of balance; the state fund would have an overall excess debit and the county fund would have an excess credit.

To balance the funds, the state fund accepts the responsibility for collecting the county taxes from the taxpayer but immediately remunerates the charges to the county fund.

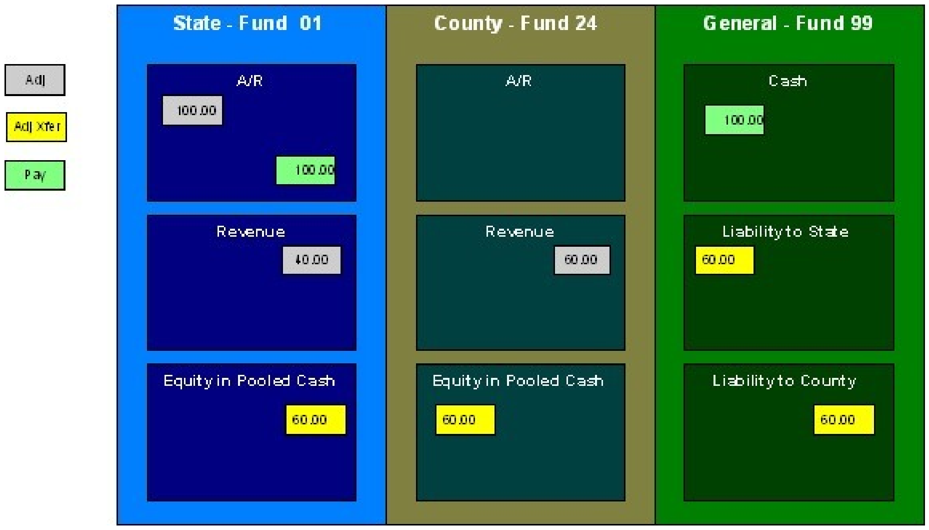


This transfer is done using the general fund. The state fund's EPC account is credited and the liability to state is debited with the amount of the county sales tax revenue. Also, the county fund's EPC account is debited and the general fund's liability to the county account is credited by the county sales tax revenue. In effect, the state fund owes the county charges to the general fund, and the general fund owes the county charges to the county fund.

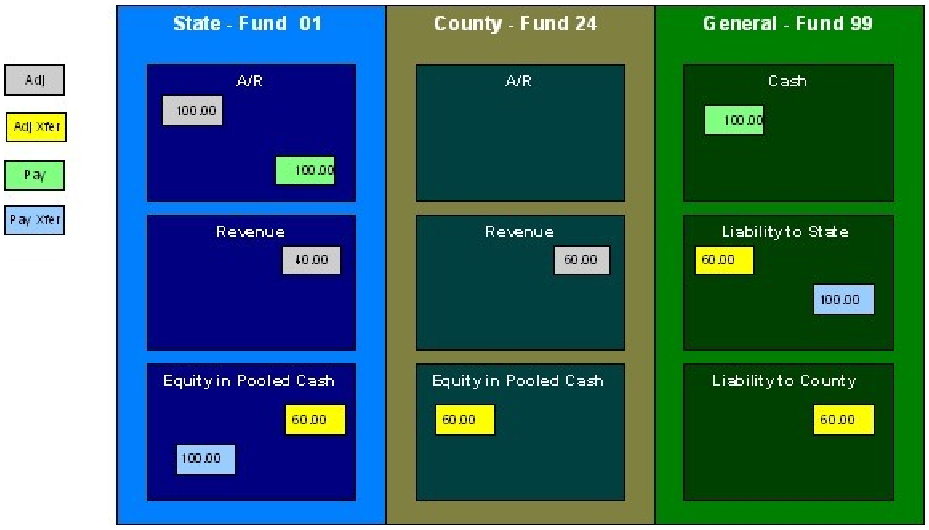
The following diagram illustrates the initial GL accounting that would occur when the payment arrives.



When the payment arrives, the cash is debited to the general fund's cash account, and the state fund's A/R is relieved. Again, the funds would be unbalanced if left in this condition; the state fund would have an excess of credits and the general fund would have an excess of debits.



To maintain each fund's balance of debits and credits, the general fund's liability to the state fund is credited by the amount of the fund's share of the cash, and the state fund's EPC is debited. Note that the payment has no effect on county fund's EPC and the general fund's liability to the county fund. The county fund "received" its money from the state department when the adjustment was created.



And so, all funds have matching debits and credits.

Accounting Method Is Defined On The Installation Options

You must turn on a switch on the *Installation Record* to enable fund accounting.

Fund Controls Fund-Balancing Entries

There are two levels of debit and credit balancing in fund accounting. There is the balancing required by double entry accounting: the total debits in the entire GL must equal the total credits. This is required regardless of whether fund or corporate accounting is used. The distribution codes for these entries come from varying sources, depending on the type of financial event.

FASTPATH:

Refer to *The Source Of GL Accounts On Financial Transactions* for information on the sources of the distribution codes.

The second level of balancing is specific to fund accounting. Within each fund-not just across the GL-the total debits must equal the total credits. The original distribution code from the financial event has a fund specified. For example, a bill would cause a debit to a fund's A/R distribution code, and included in that A/R distribution code is the fund. It is the definition of the fund that specifies whether fund-balancing entries are required and provides the distribution codes for these entries.

For a departmental fund, the fund-balancing debit and credit would be specified. When a debit is applied to a departmental fund's GL account, an additional account (typically the general fund's liability to the departmental fund) is debited and an account (typically the departmental fund's EPC) is credited. When a credit is applied to a departmental fund's account, an additional account (typically the general fund's liability to the departmental fund) is credited and an account (typically the department's EPC) is debited.

For the general fund, no fund-balancing debits and credits are specified.

Building Fund-Balancing GL Details

Building the GL details for a financial event is a two-step process.

- First, the system generates the regular GL details for a financial transaction (FT). This is done regardless of whether corporate or fund accounting is used.
- Second, with fund accounting activated, the system analyzes the distribution code on each GL detail associated with the FT. If a *fund* is specified on a distribution code, the system checks the definition of the fund. If fund-balancing entries are specified on the fund, two additional GL entries are added to the FT:
- An offsetting entry to the Equity in Pooled Cash account is created for the departmental fund (e.g., if the FT is debiting a given fund, an offsetting credit is created in the funds EPC account).
- Another entry to the departments liability account is created for the general fund.

The result is a consolidated set of GL entries for the FT, incorporating the regular entries as well as the fund-balancing entries.

Setting Up The System To Enable Fund Accounting

The following section provides an overview of how to enable fund accounting.

Turn On Fund Accounting

On the *Installation Record*, indicate that fund accounting is **Practiced**. This is the default setting.

Defining Funds

A fund must be setup for each specific fund in your organization. Don't forget to also set up a fund for the general fund. Navigate using **Admin Menu** > **Fund**.

Description of Page

Enter a **Fund** and a **Description** to identify the fund.

If this fund is used to balance other funds or to hold cash, indicate a **Fund Type** of **General**, otherwise indicate that it is **Specific**.

If the fund type is **Specific**, specify the **Equity Distribution Code** and **Liability Distribution Code**. These codes are used to balance financial transactions that span funds. The equity distribution code should belong to the same **Fund** as the one you are defining. The liability distribution code should belong to the general fund.

Distribution Codes Must Include Fund ID

All of your distribution codes must include their respective fund ID.

FASTPATH:
For more information, refer to *Setting Up Distribution Codes*.

Update Your Funds With Their Respective Equity and Liability Distribution Codes

After distribution codes have been setup, you must update your funds to indicate the equity and liability accounts used to balance inter-fund financial transactions.

Chapter 4

Defining Taxpayer Options

The definition of a taxpayer is someone (or something) with financial obligations with your revenue authority. Within the documentation, the terms "taxpayer" and "customer" may be used interchangeably.

The system subdivides taxpayer information into the following records:

- **Person.** The person record holds demographic information about your taxpayers and every other individual or business with which your revenue authority has contact. For example, in addition to normal registered taxpayers, person records may include contacts, accountants, related parties, mortgage brokers, related businesses in a corporate structure, overseas entities, persons of interest, and so on.
- **Account.** Accounts are the entities that group together tax roles and obligations where financial impacts are captured therefore you must create at least one account for every person who has financial obligations with your revenue authority.
- **Tax Role.** A tax role is used to define an instance of a specific tax type for a specific account. It includes information that is specific to the tax type for the account, for example, the dates that the tax is applicable for the account and the filing calendar that defines the filing frequency.
- **Obligation.** An obligation is a contract between the tax authority and the taxpayer. For example, an obligation may represent a specific filing period where a taxpayer is expected to file a return for a given tax type. An obligation may represent an ongoing charge such as a tax preparer fee. An obligation may be used to hold miscellaneous financial information such as an excess credit to later be applied to unpaid tax or debt from a third party source.

Before you can define persons, accounts, tax roles and obligations, you must set up the control tables defined in this section.

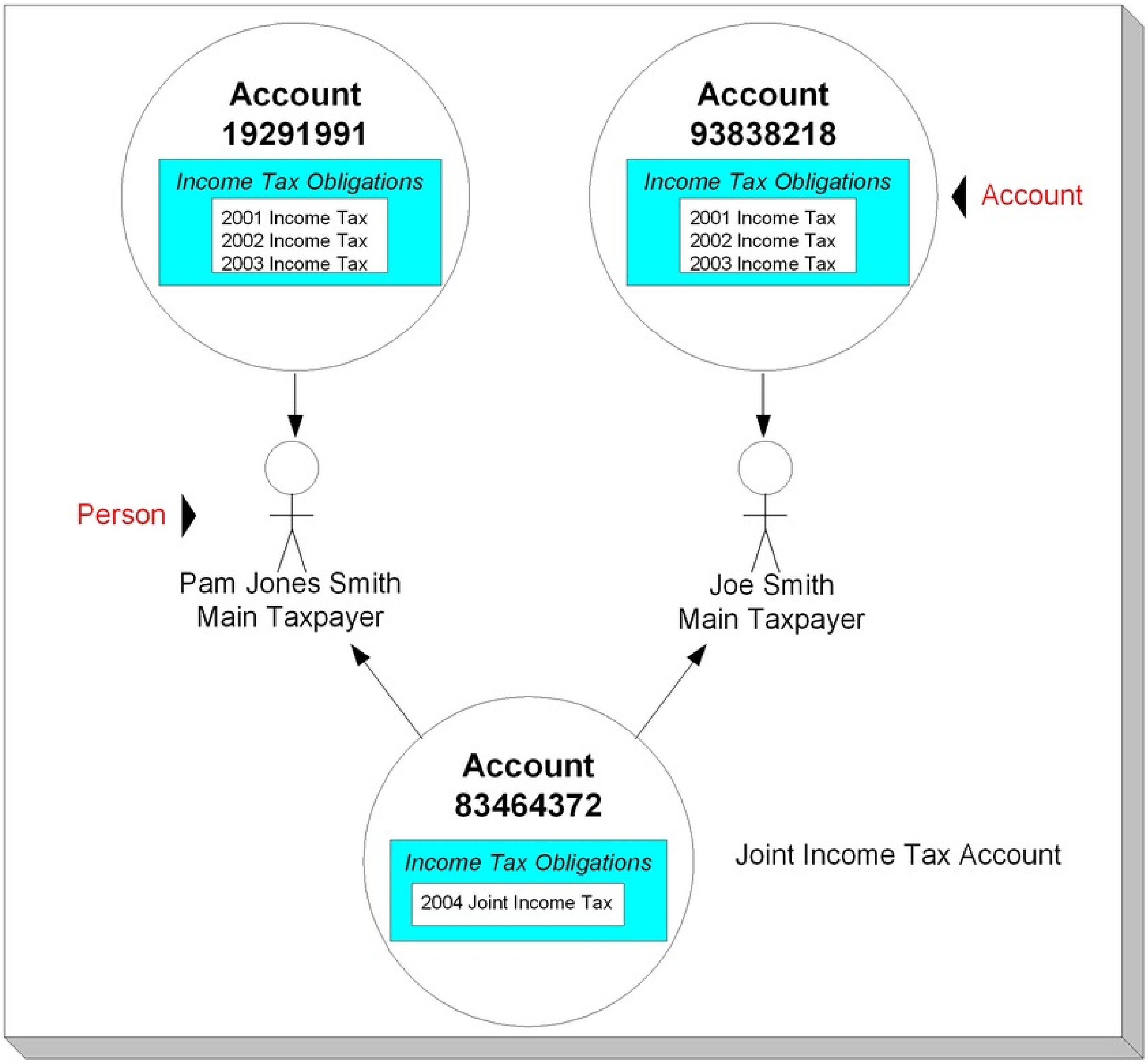
NOTE: In this section, we limit the discussion to tables that control basic demographic and financial information about your taxpayers. Other sections describe the tables that control other taxpayer related functions like forms processing, penalty and interest and collections.

Taxpayer Overview

This section describes how the person, account, tax role and obligation records are used to record your taxpayers' demographic and financial options.

A Simple Example Of Joint Taxpayers

The following picture illustrates two joint taxpayers: Joe Smith and Pam Jones Smith. Joe and Pam are the "main taxpayers" on their own Income Tax Accounts with their own obligations for tax years 2001, 2002, and 2003. Joe and Pam get married and file a joint tax return for tax year 2004. A new account is established for their 2004 obligation. They are both financially responsible for tax year 2004 while they each remain responsible for their prior tax years. Joe and Pam are each linked to two accounts for tax years 2001-2003 and to the new joint account.



Persons

Person records hold demographic information about the individuals and businesses with whom your organization communicates. Demographic information includes phone number(s), names and aliases, identification numbers, employment information, etc.

In the above example, 2 person records would be needed, one for Pam Jones and another for Joe Smith.

A new person is added when you first have contact with a person; the person does not have to be a taxpayer before he or she is added. For example:

- If John Doe is the main contact for ABC Corporation, John might not live in the taxing jurisdiction. You can add John as a new person and link him to any of the ABC Corporation accounts as a contact (but not financially responsible).
- In situations with non-filers and investigations, you can add a new person and associate it with the case. If the tax authority thinks that ABC Corporation is operating as a business but not registered and paying taxes, the tax authority may create a person record at the beginning of the investigation to send correspondence.

NOTE: **Businesses are persons too.** In addition to humans, you use person records to maintain basic information about the businesses with which your organization has contact.

FASTPATH: For a description of the control tables that must be set up before you can define a person, refer to *Setting Up Person Options*.

Person User Interface

The system supports two style of user interface. The original user interface is a fixed user interface that implementations are not able to customize. The business object driven user interface using portal / zone configuration allows for more customization at the implementation level. The Person user interface was originally implemented with the fixed user interface. However, the current implementation supports the BO driven portal / zone user interface. The product continues to support the original user interface for upgrading clients. However, functionality introduced to the person page after the introduction of the BO driven user interface are only implemented in base product business objects. Any functionality that is not supported the original user interface is noted in the documentation.

Person Names

The person can capture either a single name (Entity Name) or separate name constituents: first name, middle name, last name along with prefix and suffix. The name elements captured are driven by the person business object used. The product provides two person business objects out of the box.

- The Individual Person business object (**C1-PersonIndividual**) supports the separate name constituents. This is expected to be used for person types that represent human persons for implementations that wish to support separate name components.
- The Legal Entity business object (**C1-PersonLegalEntity**) supports a single name field. It may be used for implementations that wish to only support a single name for all persons and for businesses or other non-human persons for implementations that do want to support separate name components for human persons.

For business objects that support separate name components, the single person name field (Entity Name) must also be populated because searches throughout the system still search on the single person name. Refer to the Individual Person business object algorithms for this functionality.

The person business objects support a special option to indicate that the business object supports a single name or separate name constituents. This allows for algorithms to perform processing or validation related to the names based on this setting.

The assumed default is single name so business objects that do not configure the setting are treated as using the single Entity Name element.

A small number of searches in the system support searching for a person by separate first name and last name. The person search is one. The *360 Degree search* by person is the other. Most other searches use the single entity name person field to search for a person. If an implementation chooses not to support separate name constituents, but would prefer to use a single name for all persons, a *Feature Configuration* option allows you to configure support for a single name. This will hide the separate first and last name fields on the searches.

- Find the Feature Configuration record for the **General System Configuration** feature type. (It may need to be defined if it does not exist). Choose the option type **Person Name Support Type** and define a value of **C1FN**.
- For implementations that use the separate names may choose to create the above option with a value of **C1EN** to be explicit. However, if the option is not populated, the product assumes that separate names are supported.

NOTE: Only the BO driven portal / zone person user interface supports separate name elements. Refer to *Person User Interface* for more information.

Person Addresses

A person may refer to one or more addresses based on an address type. Configuration on the person type indicates the address types that may be linked to a person type. It also includes configuration for whether multiple are allowed for a given address type and whether seasonal addresses may be defined for a give address type.

When linking addresses to a person, the user may indicate a priority that may be used by other algorithms determining an address to use for correspondence. Refer to *Collection Letter* for an example of this functionality.

In addition, addresses for a person may be marked as undeliverable if it has been determined that mail has been returned for the person at the address.

NOTE: Only the BO driven portal / zone person user interface supports links to the centralized address. Refer to *Person User Interface* for more information.

Accounts

The system allows for a taxpayer to define one account for multiple types of taxes. For example, a corporation may have a single account for the corporate income tax, sales and use tax, withholding tax, various excise taxes, etc.

Alternatively separate accounts may be used for each type of tax.

Account ID

The unique number of an account is referred to as the "account ID". It is randomly generated and many other records that are child records of Account or related to any of the account's child records inherit part of its unique number from the Account ID.

NOTE:
Technical note. The non-intelligence of the account ID is also important from the perspective of the parallel processing that takes place when the system creates financial transactions.

Account / Person Cross Reference

A person may be linked to zero or more accounts. A person won't be linked to an account when they have no financial relationship with your organization. A person will be linked to multiple accounts when they have financial relationships with more than one account.

An account must reference at least one person (i.e., the main taxpayer), but may reference an unlimited number of individuals. Multiple persons are linked to an account when several parties have some type of financial relationship with the account (e.g., third party guarantors, account contact, etc.).

When Is An Account Created?

There a several instances when accounts must be created in the system. An account is created when:

- The taxpayer is registered for a tax obligation, or has any financial interactions with the tax authority.
- The taxpayer is registered for a new tax type. The account is created to track the obligations of the taxpayer to file or pay taxes.
- An estimated payment is received from a taxpayer for a tax that is not yet registered. The account must be created to hold the estimated payment.
- A non-filer is identified, and the tax authority decides to send a default tax assessment or penalty.

When Is An Account Expired?

Accounts never expire. Once a taxpayer has an account, the account remains in the system forever. Linked to the account are tax role records that define the applicable time period that a taxpayer is responsible for a given tax type. Obligation records are created for each revenue period where the taxpayer is obliged to file a tax return, and/or make a payment to the revenue authority. When an account has obligations, the system pursues unpaid and overdue debt, refunds credits overdue to the taxpayer, and pursues obligations to file returns that have not been fulfilled. If the account doesn't have active tax roles / obligations, the system will not consider the account for any outstanding issues. An account without active tax roles / obligations is "dormant". It may be used again in the future when the same taxpayer is obliged to file taxes again. If the taxpayer never re-files, the account (along with its financial history) remains dormant forever.

Tax Roles

Tax Roles represent the ongoing obligations of a given tax type within an account at a given point in time. It may be also be thought of as the reason a taxpayer must interact with the revenue authority, or one of the taxpayer's relationships with the revenue authority.

Tax roles include the dates that the tax is effective. If you consider a business, some tax types are required for the entire time a company is in business, such as corporate income tax. Other tax types may only be in effect if the company is engaging in certain activities, which may change over time.

When Is A Tax Role Created?

Most tax roles are created when the revenue authority is aware that a tax type is applicable for an account. For example, when a business taxpayer registers their business and indicates the various tax types that are applicable or when an individual taxpayer files a tax form.

Filing Calendars for Filing Tax Types

For tax types that are filing period based, the onus is on the taxpayer to file the return and pay the appropriate assessment on time. The tax type typically defines the valid filing calendar(s) that govern the frequency of filing and the dates to file for each filing period. For some tax types a single calendar is valid for all tax roles of that tax type. For other tax types, one or more different filing frequencies (and therefore filing calendars) may be applicable and may change over the life of the tax role for an account.

Revenue Calendars for Variable Schedules

Some tax types may need to define more than one schedule of due dates within a give revenue period. The most common example is real property taxes which are often payable in installments. Each installment may have a separate due date and be subject to different penalty and interest rules. These tax types typically have a single calendar which is valid for all tax roles of that tax type.

Tax Roles Can Maintain Obligations

For some tax types there is an ongoing obligation to file, such as sales tax or to pay billed taxes such as property taxes. For these types of taxes, it may be beneficial to include logic to create future obligations based on the start date, end date, and schedule frequency of the tax role. If the tax role dates or effective calendars change, it may be necessary to adjust the tax role's existing obligations to match.

Configuration on the tax type controls whether obligations should be maintained for tax roles of this type and how and when the obligations are created. Algorithms related to the tax role detect changes to the tax role that may affect the related obligations. For any changes that require creation of obligations, the system determines what obligation types are associated with the tax type and for each one that indicates that filing periods are applicable, calls its **Generate Obligations** algorithm plugged in on the obligation type.

This section describes the processing that is triggered by key tax role changes and provides a summary of the configuration required to implement it.

Adding A New Tax Role

When creating a new tax role, it may be necessary to create historic obligations to bring the tax role current as at the business date.

When a tax role is added, the system first determines if the tax role's tax type is configured to maintain obligations. If so, it checks the start date of the tax role.

- If the start date is in the past, the system performs the obligation creation processing real time.
- If the start date is in the future, obligations are created by batch processing when required

Changing A Tax Role Start Or End Date

A tax role's start and end dates define the period for which it is in effect. If a tax role's end date is populated or either date is changed, existing obligations for periods prior to or beyond the new dates may need to be adjusted or canceled.

When a tax role end date is changed, the following obligation processing occurs:

- If a filing period obligation exists whose start date is less than the new tax role end date and whose end date is greater than that date, the obligation end date will be updated to equal the tax role end date.

- If filing period obligations exist with a start date greater than the new tax role end date, those obligations must be canceled. If there has been any activity on the obligation that prevents cancellation, the process will report an error that must be resolved before continuing.
- If the tax role's end date was populated and has been cleared (indicating that the tax role is still in effect), the assumption is that the latest filing obligation was previously updated to match the tax role end date. That obligation's end date will be updated to align with the obligation's filing period end date.

If a tax role start date is changed, the following obligation processing occurs:

- If a filing period obligation exists whose start date is less than the new tax role start date and whose end date is greater than that date, the obligation start date will be updated to equal the tax role start date.
- If a filing period obligation exists with an date earlier than the new tax role start date, those obligations must be canceled. If there has been any activity on any of the obligations that prevents cancellation, the process will report an error that must be resolved before continuing.
- If the tax role's start date has been changed to an earlier date and the first obligation's start date is verified to be the later of the new tax role start date and the obligation's filing period's start date.
- If the tax role's start date has been changed to an earlier date such that there are no obligations covering the period from the start date to the first existing obligation, appropriate obligations are created to fill the period from the new tax role start date until the first existing obligation.

If there are non-filing period obligations linked to the tax role (obligations whose type indicates that filing is **Not Allowed**), the algorithm will not attempt to cancel them or adjust the dates. The algorithm will check if the obligation start and end dates are still within the tax role start and end dates. If so, no obligation changes are required. If not, the algorithm will report an error which the user must resolve before continuing with the update.

Effective Filing Calendar Changes

Some tax types, such as sales and use, support multiple filing frequencies: typically monthly, quarterly or annually. A tax authority may request a company filing quarterly or annually to file more frequently if their revenue increases. It's not as common for a taxpayer filing monthly to switch to quarterly or annually if the revenue decreases, but it is possible. If a tax role's effective filing calendar is changed, existing obligations for the previous filing frequency need to be adjusted or canceled.

When a change is made to the effective filing calendar for a tax role, the following obligation processing occurs:

- If a new calendar is added or an existing calendar has its effective date moved back, filing period obligations with a start date on or after the new effective date are canceled. If a filing period obligation exists with a start date prior to the new calendar effective date and an end date after that date, the obligation end date is adjusted to be the new calendar effective date minus one day.
- If a calendar is deleted or an existing calendar has its effective date moved forward, filing period obligations with a start date on or after the changed or deleted calendar's prior effective date are canceled. If a filing period obligation exists with a start date prior to the old calendar effective date and an end date after that date, the obligation end date is adjusted to align with the obligation's filing period end date.
- New filing period obligations are created to bring the tax role up to date for the effective calendar change as at the current business date, if required. Note that the system executes the same obligation creation logic that is invoked when the tax role is added. In this situation, the system creates obligations starting from the end date plus one day of the latest valid filing period obligation after the adjustments described above are complete.

NOTE: The system does not allow changes to both the filing calendar collection and the tax role start / end dates. These changes must be done in separate distinct updates.

Periodic Obligation Creation

Once the initial obligations have been created for a tax role, the system assumes that a batch process is used to create obligations on a periodic bases. Configuration controls how / when this occurs:

- If the tax type indicates that obligations are maintained via a monitor, the standard monitor process, configured for **Active** tax roles is used to create the next expected obligation for each obligation type. The tax role monitor is date driven so that not all tax roles are reviewed every time the monitor is run. The **Generate Obligations** algorithm that creates the obligation periodically is responsible for determining the monitor date when the next obligation should be created. This configuration is useful when a tax type supports multiple calendars with different frequencies and where the creation of the new obligations should happen at a predictable time, such as just before the next filing period start date.
- If the tax type indicates that obligations are maintained adhoc, a separate stand-alone batch job may be used to create the obligations. This configuration is useful when a tax type has a standard calendar for all tax roles and when the creation of the obligations is not predictable. For example, for property taxes the frequency is typically annual where the filing period covers a fiscal year. Sometimes the obligations are created several months in advance of the start of the fiscal year, sometimes after the start of the fiscal year.

Regardless of which batch mechanism is used, the system executes the same obligation creation logic that is invoked when the tax role is added, bringing the tax role up to date with the business date.

The system also supplies a base monitor algorithm which transitions the tax role to the **Expired** state when the tax role end date is passed. For monitored tax roles, once the tax role end date is reached and the tax role is expired, no further obligations are created.

Canceling A Tax Role

A tax role may be canceled if it was created in error. In this situation, the system will attempt to cancel all obligations linked to the tax role. If any of the obligations cannot be canceled - for instance, if there are non-canceled financial transactions or a form has been filed for the obligation - the process will issue an error. The user must resolve the specific issue with the obligation before continuing with the cancellation.

Configuration Controls Obligation Management

The processing described in the previous sections is carried out by a number of algorithms, which in turn invoke algorithms in other plug-in spots. The key elements are as follows:

- A Business Object Post Processing algorithm is responsible for the logic that cancels or adjusts any obligations whose periods are affected by changes to the tax role start date, end date or effective calendars.
- A second Business Object Post Processing algorithm is responsible for the logic that creates obligations to bring the tax role's obligations up to date for the current business date. This algorithm caters for the processing required when a tax role is added for the first time and also for the processing required to recreate obligations for a new filing frequency, in circumstances where a change to the tax role effective calendar has been back dated. This algorithm should be configured to execute after the algorithm that cancels unwanted obligations, described above.
- An Obligation Type - Generate Obligations algorithm is required for each obligation type linked to a tax type that is configured to maintain obligations. The base post processing algorithm that creates current and historic obligations for a tax role will invoke this plug in for each obligation type associated with the tax role's tax type. A base algorithm is supplied to generate filing period obligations based on the associated tax role's effective calendar(s).
- For business objects that support obligation maintenance based on a monitor, a Business Object Status Monitor algorithm is responsible for the logic that periodically creates the next obligation for the tax role when the prior obligation end date is reached. It should be plugged in on the Active state of the tax role business object. The base tax role monitor algorithm

invokes the same logic as the post processing algorithm that creates obligations real time. It therefore also relies on the Obligation Type - Generate Obligations plug-in spot.

- A second Business Object Status Monitor algorithm is responsible for the logic that transitions a tax role to the Expired state when the tax role end date is reached. It should be plugged in on the Active state of the tax role business object.
- A Business Object Status Enter algorithm is responsible for the logic that cancels all obligations for a tax role when the tax role is canceled. It should be plugged in on the Canceled state of the tax role business object.
- Various Business Object Validation algorithms are supplied to enforce rules related to obligation maintenance that are governed by tax type controls. Refer to *Configuring Tax Types* for more information on tax role related configuration and other tax role validations controlled by tax type settings.

Refer to the base Tax Role *business objects* for example of how to configure the base algorithms that are supplied for the business object plug-in spots described above.

Refer to *Setting Up Obligation Types* for more information on the Generate Obligations plug-in spot and the available algorithms.

Obligations

An obligation is a contract (either formal or implied) between the revenue authority and a taxpayer. An obligation is linked to an account. There is no limit to the number of obligations that may be linked to an account.

When Is An Obligation Created?

Obligations may be created at different times for different tax types. Some tax types may require obligations to be created in advance. For tax types such as individual income taxes and excise taxes, obligation records may be created when the revenue authority receives a tax return.

Refer to *Tax Roles Can Maintain Obligations* for a description of how tax roles can govern the creation of future obligations.

Due Dates for Filing Period Based Obligations

For tax types that are filing period based, the onus is on the taxpayer to file the return and pay the appropriate assessment on time. This section describes the configuration provided for defining and determining due dates.

When should the taxpayer file the return for a given filing period? When defining filing periods for a *filing calendar* you also define the statutory Due Date of the return along with the Grace Date. For taxpayers that request an extension to their filing date, an Override Filing Due Date may be defined on the specific *obligation*. An Override Due Date may also be automatically calculated by the system when an obligation's end date differs significantly from the filing period end date. The obligation type defines an Override Filing Grace Days. The base logic that determines the statutory and override due dates for an obligation uses the override filing grace days in its calculation of the override filing grace date.

Typically the statutory payment due date for a filing period is the same date as the statutory filing due date. However, the dates may be extended independently. Extensions to a payment date are also defined on a specific obligation using the Override Payment Due Date. Also note that the grace date for the payment due date is calculated differently. Instead of using the explicit Grace Date on the filing calendar, the obligation type defines a Payment Grace Days. The logic in the base algorithms that determine if a payment is made on time (whether it be for the statutory due date or the override payment due date) considers the grace days in its calculations.

Due Dates for Billable Obligations

For billable tax types, the revenue authority notifies the taxpayer of any obligation amounts due by issuing a bill.

Bill-based calendars generally have statutory due dates and grace dates but differ from filing based taxes in that the debt may be payable in more than one installment. The due dates of the installments may be extended independently on a bill by bill basis. The base logic that determines the due dates for the financial transactions of billable obligation will reference the related bill segment to determine if an override due date applies.

Financial Transactions Are Linked To Obligations

FASTPATH:
For more information about how financial transactions are linked to obligations, refer to *The Financial Big Picture*.

When Is An Obligation Closed?

For an obligation to be closed, its balance must be zero. Different obligation types may apply additional criteria to ensure that all activity for the obligation is complete. For a filing based obligation type, it may be necessary to check that the a tax form has been posted. For a bill-based obligation, it may be necessary to ensure that there is a completed bill.

When an account is monitored for overdue debt, the system also checks the account's obligations to determine if any of them can be closed. If the obligation balance is zero, the process invokes an optional Obligation Type algorithm that checks additional closure criteria. If those criteria are also met, the obligation is closed.

Refer to *Setting Up Obligation Types* for more information on the Close/Reactivate Criteria plug-in spot and the available algorithms.

Setting Up Person Options

This section describes tables that must be set up before you can define persons.

Defining Name Prefixes

Any implementation that supports separate person names and would like to support a name prefix / title must define valid name prefixes. Go to **Admin Menu** > **Lookup** and search for lookup **NAME_PREFIX_FLG**.

Defining Identifier Types

When you set up a person, you may define the various types of identification associated with the person, e.g., their driver's license number, their tax identity, etc. Every piece of identification associated with a person has an identification type. These identifier type codes are defined using **Admin Menu** > **Identifier Type**.

NOTE:
How are person identifiers used? The reason why identifiers are defined on a person is so that users you can look for a taxpayer using one of their person identifiers. In addition, person identifiers help prevent duplicate persons from being added to the database. This is because the system warns a user before they add a new person when a person exists with the same identifier.

NOTE:

Person identifier types are optional. An *installation option* controls whether at least one identifier type is required on every person.

Description of Page

Enter an easily recognizable **ID Type** and **Description** for the Identifier Type.

If the identifier type has a format against which validation can be performed, use **Identifier Format** to define the algorithm. To do this:

- Create a new algorithm (refer to *Setting Up Algorithms*).
- On this algorithm, reference an Algorithm Type that validates identifier types. Click *here* to see the algorithm types available for this plug-in spot.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_ID_TYPE*.

Defining Person Relationship Types

It is possible to associate persons to other person. For example,

- You might want to define the subsidiaries of a parent corporation
- You might want to define spouses as separate persons and then link each person to another person

When you link a person to another person, you must define in what way the person is related to the other person by using a person relationship type code. These codes are defined using **Admin Menu** > **Person Relationship Type**.

Description of Page

Enter the following for each relationship type:

- Enter an easily recognizable **Relationship Type** code.
- Use **Description (Person1=>Person2)** to describe how the first person is related to the second person.
- Use **Description (Person2=>Person1)** to describe how the second person is related to the first person.

NOTE:
Person 1 versus Person 2. When you link persons together, you do it in respect of one of the persons (which we call Person 1). For example, if you want to link the subsidiaries to a parent company, you do this in respect of the parent company (i.e., you define the parent company's subsidiaries using the *Person - Persons* transaction.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_PER_REL_TYPE*.

Defining Person Types

A person exists for every individual or business with which your tax authority has contact. Besides individual and business taxpayers, persons exist for contractors, accountants at corporate taxpayers, third party guarantors, collection agencies, etc.

The person type is used to define business rules for different types of persons. For example:

- The valid external Identifier type.

- A business that is a limited liability partnership, you may define the valid relationship type to use for additional responsible parties.
- The business object that governs the person record, including its user interface behavior.
- The valid address types to use when linking addresses to persons. Refer to *Configuring Your System for Centralized Address* for more information.

NOTE: This is only applicable for systems that are not configured for *legacy address* functionality.

A person type is governed by a *business object*. The base product provides business objects that you may use or extend. Refer to the BO definitions in the application. Or you may create your own.

Open **Admin Menu** > **Person Type** to define the person types used to categorize your persons.

The topics in this section describe the base-package zones that appear on the person type portal.

Person Type List

The Person Type *List zone* lists every person type. The following functions are available:

- Click the *broadcast* icon to open other zones that contain more information about the adjacent person type.
- The standard actions of **Edit, Duplicate** and **Delete** are available for each person type.

Click the **Add** link in the zone's title bar to add a new person type.

Actions

This is a standard actions zone. The **Edit, Delete** and **Duplicate** actions are available.

Person Type

The Person Type zone contains display-only information about a person type. This zone appears when a person type has been broadcast from the Person Type List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_PER_TYPE*.

Defining Person Information

The person information displayed throughout the system is controlled by a plug-in.

The system first looks to see if the person type references a person business object and if the business object defines an information plug-in. If a BO is not provided or if that BO does not define an information algorithm, the system looks for an information algorithm plugged into the person maintenance object.

If no plug-ins are found on the BO or the MO, the system looks for a Person Information plug-in algorithm on the *installation record*.

If you prefer different formatting logic, your implementation should provide a plug-in at one of the above plug-in spots, as appropriate for your business rules.

Setting Up Account Options

This section describes tables that must be set up for an account.

Setting Up Account Management Groups

Users are informed that something requires their attention by entries that appear in To Do lists. For example, consider what happens when:

- A tax return fails validation for a math error.
- A tax return meets the business rule criteria to trigger a desk audit.
- A taxpayer refund exceeds the business rule review thresholds.
- A letter to a taxpayer is in error because the taxpayer does not have an address record.

You can optionally use account management groups (AMG) to define the respective role to be assigned to To Do entries that are associated with an account and a given To Do type. For example, you can create an AMG called **Credit Risks** and assign this to accounts with suspect credit. Then, whenever an account-oriented To Do entry is created for such an account, it will be assigned a role based on the **Credit Risks** AMG. Refer to *Assigning A To Do Role* for more information.

NOTE:
Account management groups are optional. You need only set up account management groups (and link them to accounts) if you wish to address specific To Do entries associated with specific accounts to specific roles.

Account management groups are defined using **Admin Menu** > **Account Management Group**.

Description of Page

Enter an easily recognizable **Account Management Group** code and **Description** for each account management group. Use the grid to define the **To Do Role** to be assigned to entries of a given **To Do Type** that are associated with accounts that reference the Account Management Group.

NOTE:
Only To Do entries that are account-oriented take advantage of the roles defined for an account management group (because only accounts reference an account management group).

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_ACCT_MGMT_GR*.

Setting Up Account Relationship Codes

When you link a person to an account, you must define in what way the person is related to the account by using an account relationship code. These codes are defined using **Admin Menu** > **Account Relationship Type**.

Description of Page

Enter an easily recognizable **Relationship Type** and **Description** for each relationship type.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_ACCT_REL_TYP*.

Setting Up Alert Types

Permanent or temporary alerts may be linked to an Account and will therefore appear in the alert zone. Each alert linked to an account must have an Alert Type. To define valid alert types, navigate to **Admin Menu** > **Alert Type**.

Description of Page

Enter an easily recognizable **Alert Type** code and **Description** for each alert type. Specify the **Alert Days** to indicate the amount of time that alerts of this type will be effective by default. Specify a value of zero to indicate that alerts of this type will be effective indefinitely by default.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_ALERT_TYPE*.

Setting Up Account Types

When you set up an account, you must assign it an account type. The topics in this section describe the account type control table.

Account Type - Main

To set up account types, navigate to **Admin Menu** > **Account Type**.

Description of Page

Enter a unique **Account Type** code and **Description** for every account type.

Use **Collection Class** to define the collection class that defaults onto new accounts that belong to this account type. An account's collection class may be subsequently modified if the account has special collection problems or needs.

Use an **Account Business Object** to define a *BO* that may govern additional rules related to accounts of this type.

Turn on **Business Activity Required** if obligations linked to accounts with this account type require a Business Activity description to be entered.

Turn on **Open Item Accounting** if accounts belonging to this account type are subject to open-item accounting. Refer to *Open Item Accounting* for a complete explanation of the significance of this switch.

> **NOTE:** This switch is only applicable to implementations supporting the *classic billing* functionality.

Turn on **Non Tax Agency Payment** if accounts belonging to this account type are used for payments made to reduce non-tax authority debt. For example, if your tax authority sends collection notices or offsets taxpayer refunds on behalf of other agencies, such as child support, college tuition, or parking violations, you should set up the following information to accept such payments:

Create a new account type called "Non Tax Authority Customer".

- Create an obligation type for each type of non-tax authority payment that taxpayers can make. Make sure to enter a distribution code on each obligation type that references the appropriate revenue (or payable) account. If your implementation uses **Classic Billing** you must indicate on each obligation type that it is not billed.

- Create an account to which you'll book such payments. Have this account reference the new account type. We recommend creating a separate account for each obligation type that you created in the previous step.
- Create and activate an obligation for the new account(s).

When someone pays for non-tax authority debt, the operator will add a payment for the above account. On the payment, the operator should record reference information in order to know exactly why the payment was made. Refer to *Payment Event - Main* for more information.

You must define a variety of business rules for every division in which an account type has taxpayers. You do this on the *Account Type - Controls* page. The grid that follows simply shows the divisions for which business rules have been set up.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_CUST_CL*.

Account Type - Bill Messages

When an account type has bill messages, the system will sweep these messages onto bills created for accounts belonging to the account type. Use this page to define an account type's bill messages. Navigate to **Admin Menu** > **Account Type** > **Bill Messages** tab to maintain this information.

NOTE: Optional information. This tab is only visible if your implementation has not *turned off* the **BI — Billing (Classic)** module. The information is used only for classic billing functionality.

Description of Page

Use the bill messages collection to define **Bill Message** codes that should appear on bills that created for accounts that belong to a given account type. For each message, also specify the **Start Date** and **End Date** when such a message should appear on the bill (leave **End Date** blank if the message should appear indefinitely).

Where Used

The system snaps account type bill messages on a bill during bill completion. For more information about bill messages, refer to *The Source Of Bill Messages*.

Account Type - Controls

You must define a variety of business rules for every division in which an account type has taxpayers. Open **Admin Menu** > **Account Type** > **Controls** tab to maintain this information.

Description of Page

The **Account Type Controls** scroll contains business rules governing accounts that belong to a **Division** and **Account Type**. The following fields should be defined for each **Division**:

- Use **Min Compliance Review Freq (Days)** to define the maximum number of days that can elapse between the reviews of an account's debt by the *overdue monitor*. Note, a value of zero (**0**) means that accounts in this account type will be reviewed every day.
- Use **Compliance Review Grace Days** to define the number of days that algorithms may add to a given due date when requesting that an account should be reviewed by the *overdue monitor*. For example, when completing a bill, an algorithm inserts a record into the account review dates collection setting the review date to the bill due date plus the number entered here.

The grid that follows 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 (see the table that follows for a description of all possible events).
- 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.

CAUTION:
These algorithms are typically significant system processes. The absence of an algorithm may prevent the system from operating correctly.

You can define algorithms for the following **System Events**:

System Event	***Optional / Required***	***Description***
Autopay Amount Over Limit	Optional	This algorithm is called to handle the situation when a system-initiated *automatic payment* is created that exceeds the taxpayer's *maximum withdrawal limit*. Specifically, this algorithm is called when: - The account has a maximum withdrawal limit on their *automatic payment options* - The system attempts to create an automatic payment that exceeds this amount - The automatic payment algorithm that's plugged into the installation record has logic that invokes this algorithm when the above conditions are true. For more information refer to system event **Automatic Payment Creation** in *Installation Options — Algorithms*. If you do not plug-in this type of algorithm and the above situation is detected, the automatic payment will be created and no error will be issued. Refer to *How To Implement Maximum Withdrawal Limits* for more information. Click *here* to see the algorithm types available for this system event.
FT Freeze	Optional	When an FT is frozen, this algorithm is called to do additional work. For example, if you wish to schedule compliance review when an account’s balance changes, you may need such an algorithm to handle creating an account review trigger depending on the financial transaction type. Click *here* to see the algorithm types available for this system event.
Levy an NSF Charge	Optional	This algorithm is called when a payment is canceled with a cancellation reason that indicates an NSF. Refer to *NSF Cancellations* for more information about what happens when a payment is canceled due to non-sufficient funds. **Only One Algorithm**. Only one algorithm to levy an NSF charge may be defined for an account type / division combination. Click *here* to see the algorithm types available for this system event.
Overpayment Distribution	Required	When a taxpayer pays more than they owe, this algorithm is called to determine what to do with the excess funds. Refer to *Overpayment Obligations* for a description on

		how to configure the system to handle your overpayment requirements. **Only One Algorithm**. Only one overpayment distribution algorithm may be defined for an account type / division combination. Click *here* to see the algorithm types available for this system event.
Payment Cancellation	Optional	Algorithms of this type are called when a payment is canceled. Click *here* to see the algorithm types available for this system event.
Payment Distribution	Required	This algorithm is called to distribute a payment amongst an account's obligations. Refer to *Payment Distribution* for more information about how payment distribution works. **Only One Algorithm**. Only one payment distribution algorithm may be defined for an account type / division combination. Click *here* to see the algorithm types available for this system event.
Payment Freeze	Optional	When a payment is frozen, this algorithm is called to do additional work. For example, you may need such an algorithm to recalculate penalty and interest charges on receipt of a payment Click *here* to see the algorithm types available for this system event.

Additional Details

The following controls are only applicable if your implementation uses **Classic Billing** functionality.

Use **Days Till Bill Due** to define the number of days after the bill freeze date that the taxpayer's bill is due. If the due date is a weekend or company holiday, the system will move the due date forward to the next workday (using the workday calendar defined on the account's division).

You can define algorithms for the following billing related **System Events** :

System Event	***Optional / Required***	***Description***
Bill Cancel	Optional	This algorithm provides the ability to include additional cancel logic when canceling online. Algorithms of this type can be called in two modes: D (Determine Bill Page Buttons) and X (Cancel Bill). Mode 'D' governs whether an action button to cancel the bill will appear on the Bill page and mode 'X' performs the actual cancellation logic. Click *here* to see the algorithm types available for this system event.
Bill Completion	Optional	When a bill for an account is completed, bill completion algorithms are called to do additional work. Refer to the description of the Complete button under *Bill Lifecycle* for a description of when this algorithm is called during the completion process. Click *here* to see the algorithm types available for this system event.
Bill Eligibility	Optional	Algorithms for this plug-in spot are called when generating a bill in batch billing. It provides the ability to determine if an account

		is ineligible for billing and should therefore be skipped from further processing. If an eligibility algorithm is not used, a bill is created for any account in the open bill cycle and is later deleted by the billing process if it detects that there in no information linked to the bill. Click *here* to see the algorithm types available for this system event.
Bill Segment Freeze / Cancel	Optional	When a bill segment for an account in this account type / division is frozen or canceled, an algorithm of this type may be called to do additional work. Refer to *Bill Segment Lifecycle* for more information about freezing and canceling bill segments. Click *here* to see the algorithm types available for this system event.
FT Freeze	Optional	If you practice *Open Item Accounting*, you will need an FT Freeze algorithm to handle the cancellation of match events when a financial transaction is canceled that appears on a match event. Refer to *How Are Match Events Cancelled?* for more information about cancellation. **NOTE:** Open Item Accounting is only applicable to **Classic Billing** functionality. Click *here* to see the algorithm types available for this system event.
Override Due Date	Optional	An account's bill due date will be equal to the bill date plus its account type's Days Till Due. If you need to override this method for accounts in a specific account type, specify the appropriate algorithm here. **Only One Algorithm**. Only one due date override algorithm may be defined for an account type / division combination. Click *here* to see the algorithm types available for this system event.
Post Bill Completion	Optional	When an account type has algorithms of this type, they are called after the completion of a bill for an account linked to this account type. Refer to the description of the Complete button under *Bill Lifecycle* for a description of when this algorithm is called during the completion process. Click *here* to see the algorithm types available for this system event.
Pre Bill Completion	Optional	When an account type has algorithms of this type, they are called immediately before completion starts for an account linked to this account type. These algorithms have the potential of: • Deleting a bill. You might want a pre completion algorithm to delete a bill if a condition is detected that should inhibit the sending of a bill to a taxpayer (e.g., the bill just contains information about recent payments). • Aborting the completion process and creating a bill exception. If the algorithm indicates this should be done, the bill is left in the **pending** state and a bill

		exception is created describing why completion was aborted. You might want a pre completion algorithm to do this if, for example, integrity checks detect there is something wrong with the account or its obligations. If the integrity check fails, the bill can be left in the **pending** state and a bill exception created describing why. Refer to the description of the Complete button under *Bill Lifecycle* for a description of when this algorithm is called during the completion process. Click *here* to see the algorithm types available for this system event.
Payment Freeze	Optional	If you practice *Open Item Accounting*, you will need a Payment Freeze algorithm to link the payment's financial transactions to the match event that was originally created when the payment was distributed. Refer to *Payments and Match Events* for more information. **NOTE:** Open Item Accounting is only applicable to **Classic Billing** functionality. Click *here* to see the algorithm types available for this system event.

NOTE: Optional information. The Days Till Bill Due field is only visible if your implementation has not *turned off* the **BI — Billing (Classic)** module but the billing related system events will appear in the drop down list regardless of whether the module is enabled or not.

Setting Up Collection Classes

FASTPATH: Refer to *Setting Up Collection Classes* for a description of how to set up collection classes.

Defining Account Information

The account information displayed throughout the system is controlled by a plug-in.

The system first looks to see if the account type references an account business object and if the business object defines an information plug-in. If a BO is not provided or if that BO does not define an information algorithm, the system looks for an information algorithm plugged into the account maintenance object.

If no plug-ins are found on the BO or the MO, the system looks for an Account Information plug-in algorithm on the *installation record*.

If you prefer different formatting logic, your implementation should provide a plug-in at one of the above plug-in spots, as appropriate for your business rules.

Configuring Tax Types

Tax roles and obligation types must reference a tax type. The base Tax Type business objects includes configuration options that govern certain aspects of tax role and obligation processing.

This section describes the effect of these configuration options within the system.

External ID Usage

Some tax types use external IDs for additional taxpayer identification on tax forms. These IDs may be assigned by the tax authority or by another agency. The variable calendar tax type business objects allows configuration to indicate whether an external ID is required, optional or not allowed on tax roles that reference this tax type.

A base *business object* validation algorithm is supplied to validate the external ID field on a tax role based on this configuration. Refer to the base Variable Calendar Tax Role business object for an example of how to configure this validation.

Maintaining Obligations

When you set up a tax type, you must indicate whether the system is responsible for automatically *maintaining obligations for tax roles* of that tax type. The logic to create and maintain obligations is governed by plug-ins on the tax role business object. Only tax types that whose tax role applicability is set to **Required** may be configured to maintain obligations.

The product provides two tax type business objects that support maintaining obligations: Variable Calendar Tax Type and Asset Based Tax Type. The following points describe maintain obligation configuration supported by both business objects:

- Different tax type may have different requirements for when their obligations are created. The Obligation Maintenance Control option value of **Monitored** indicates that the deferred monitor on the tax role periodically reviews active tax roles and creates the next obligation. A value of **Adhoc** indicates that a separate Adhoc batch job is responsible for creating the next obligation for all the tax roles in the tax type when run.
- Different tax type may have different requirements for whether the next obligation to create should be the one that covers the current period or not. The Last Obligation Control option value of **Through Current Period** indicates that obligations are created up to and including the period that ends after the current date. A value of **Through Next Period** indicates that obligations are created up to and including the period that starts after the current date.
- Different tax type may have different requirements for whether the next obligation to create should be the one that covers the current period or not. The Last Obligation Control option value of **Through Current Period** indicates that obligations are created up to and including the period that ends after the current date. A value of **Through Next Period** indicates that obligations are created up to and including the period that starts after the current date.

The following points highlight additional configuration available on the Variable Calendar Tax Type:

- For tax types that allow different filing frequencies, the different filing calendars are typically set up such that there is a point when calendars of different frequencies have a common Filing Period Start Date. If the tax type is configured to maintain obligations, you may also need to specify whether filing calendar changes for the tax role can occur 'off cycle'.
- Setting the Change Filing Frequency Off Cycle option to **Allowed** will allow users to specify a filing calendar change that becomes effective before the next common filing period start date of the new calendar is reached. This will result in an obligation for either the old or new calendar with a period shorter than the filing period for the effective filing frequency. Note that the system will not allow two obligations to be created where the period covered does not align with the associated calendar's filing. Therefore, the effective date for an off-cycle filing calendar change must align with either a filing period start date of the new calendar or a filing period end date of the old calendar.
- Setting the Change Filing Frequency Off Cycle option to **Not Allowed** will only allow users to specify a filing calendar change that aligns with a filing period start date of both the old and new calendars.
- A base business object validation algorithm is supplied to validate filing calendar changes for a tax role based on this configuration.
- A base business object validation algorithm is supplied to validate that none of the effective calendars for a tax role whose tax type is configured to maintain obligations allows overlapping periods.

The assumption is that tax types that maintain future obligations do not allow overlapping filing periods within the same filing calendar. The base logic to create future obligations assumes that filing periods within a given a filing calendar have contiguous period dates.

Refer to the base Variable Calendar Tax Role business object and the Asset Based Tax Role business object for examples of how to configure the associated validations for this tax type configuration.

FASTPATH: Refer to *The Big Picture of Business Objects* for more information about business objects and the validation algorithm.

Overriding Obligation Filing Due Dates

For filing based tax types, the obligation's filing due date is normally determined by the due date configured on the applicable filing period. However, some situations may result in an obligation where the end date of the obligation does not coincide with the filing period end date. This may occur if the tax type supports multiple filing frequencies and the taxpayer changes to a more frequent filing schedule - for example, from an annual filing to a quarterly filing. If a business ceases to operate and the associated tax role is end dated, this may also result in a filing obligation whose end date is prior to the expected filing period. Tax types may be configured to automatically calculate an override due date for the obligation that covers the shortened period.

The Variable Calendar Tax Type provides the following configuration options for overriding an obligation due date:

- **Override Due Date** indicates if an override due date should be calculated for an obligation of this tax type when the end date of the obligation is different from the filing period's end date.
- **End Days Difference Tolerance** is the minimum number of days difference between the obligation end date and the filing period end date that should cause the due date to be overridden.
- **Months After Obligation End Date** is added to the obligation end date to calculate the override due date month.
- **Day Of The Month** is the day of the month of the new due date, adjusted for differing maximum days in the month.

A base *business object* post processing algorithm is supplied to perform the date calculations described above, based on this configuration. This algorithm should be plugged in on the associated business object for each obligation type associated with the tax type for which due date override is applicable. Refer to the base Filing Period with Tax Role Calendar Obligation business object for an example of how to configure this processing.

Valid Address Types

If tax roles for a tax type may define addresses, the valid address types must be configured on the tax type. Refer to *Configuring Your System for Centralized Address* for more information.

NOTE: This is only applicable for systems that are not configured for *legacy address* functionality.

Valid Calculation Control Types

Tax roles for billable tax types may need to define the effective calculation controls to be used in generating bills. If so, the valid calculation controls must be configured on the tax type, one of which should be designated as the tax type default. Refer to *Defining Calculation Engine Options* for more information about calculation controls.

Setting Up Tax Types

The tax type is used to define business rules for each type of tax you support. Tax type is required when defining any financial obligation so a catchall tax type of **Other** for write-offs, payment plans and other miscellaneous uses may be warranted.

A tax type is governed by a *business object*. The base product provides business objects that you may use or extend. Refer to the BO definitions in the application. Or you may create your own.

Open **Admin Menu** > **Tax Type** to define the tax types used to define information about the tax types you support.

The topics in this section describe the base-package zones that appear on the tax type portal.

Tax Type List

The Tax Type *List zone* lists every tax type. The following functions are available:

- Click the *broadcast* icon to open other zones that contain more information about the adjacent tax type.
- The standard actions of **Edit, Duplicate** and **Delete** are available for each tax type.

Click the **Add** link in the zone's title bar to add a new tax type.

Tax Type

The Tax Type zone contains display-only information about a tax type. This zone appears when a tax type has been broadcast from the Tax Type List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_SVC_TYPE*.

Overpayment Process Types

The Overpayment Process Type zone displays the list of overpayment process types that are valid for this tax type. Click the Add/Edit button to maintain this list.

The Usage flag may be used to mark a given overpayment process type to use as the default for a given scenario. For example, the product provides a usage value for indicating the **Default for Tax Form Creation** to indicate the overpayment process type to use when one is created as the result of a tax form. This is helpful if a given tax form rule may be used by different tax types.

NOTE: The usage flag is a customizable lookup. Implementers may add their own values. The lookup name is **OP_PROC_TYPE_USG_FLG**.

Setting Up Tax Type Options

This section describes tables that are related to tax types.

Setting Up Revenue Calendars

Revenue calendars are used to define the expected revenue period dates for a given type of obligation. The valid revenue calendar that governs an obligation's revenue period is typically defined on the obligation's *Tax Role* or its *Tax Type*.

The product supports two types of usage for revenue calendars. Filing based calendars are designed to support return based taxes and map a single set of filing related due dates per revenue period. Variable schedule calendars support multiple schedules of dates within a revenue period. Variable schedule calendars are used for billing based tax types which are typically payable in installments with differing due dates.

For more information, refer to the base product business objects provided for filing based taxes (**C1-FilingCalendar** and **C1-FilingPeriod**) and variable schedule taxes (**C1-TaxBillCalendar** and **C1-TaxBillingPeriod**).

To maintain or set up a revenue calendar and its revenue periods, open **Admin Menu** > **Revenue Calendar**.

The topics in this section describe the base-package zones that appear on the Revenue Calendar portal.

Revenue Calendar List

The Revenue Calendar *List zone* lists every Revenue Calendar. The following functions are available:

Click the *broadcast* icon to open other zones that contain more information about the adjacent Revenue Calendar.

The standard actions of **Edit**, **Duplicate** and **Delete** are available for each Revenue Calendar.

Click the **Add** link in the zone's title bar to add a new Revenue Calendar.

Revenue Calendar

The Revenue Calendar zone contains display-only information about a revenue calendar. This zone appears when a revenue calendar has been broadcast from the Revenue Calendar list, or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_FILING_CAL*.

Revenue Periods List

The Revenue Periods *List zone* list all revenue periods for the broadcast calendar. The revenue periods are shown in descending order by date. The zone details vary according to the calendar usage.

For filing based calendars:

- The list includes the period start and end dates, filing due date and filing grace date and revenue year
- Standard edit and delete actions are available.
- Clicking **Add** link in the zone's title bar will add a new filing period.
- Clicking **Add Multiple Periods** link in the zone's title bar will open a pop-up window in which multiple filing periods may be added using a grid input format or uploaded from a spreadsheet

For other calendar usages:

- The list includes the period start and end dates and revenue year
- Standard edit and delete actions are available.
- Clicking **Add** link in the zone's title bar will add a new revenue period.

- Clicking the Begin Date hyperlink in the zone's title bar will navigate to the revenue period portal, where the full details of the period can be maintained, including the schedule of due dates

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_FILING_CAL_PERIOD*.

Revenue Period Portal

This portal appears when a revenue period has been selected from the Revenue Periods list zone.

The topics in this section describe the base-package zones that appear on this portal.

Revenue Period

The Revenue Period zone contains display-only information about a specific revenue period. This zone appears via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_FILING_CAL_PERIOD* .

Setting Up Tax Role Relationship Types

The tax role relationship type is used to define the types of relationships that persons other than the primary taxpayer can have to a tax role. Tax role relationship type is required when linking a related person to a tax role.

A tax role relationship type is governed by a *business object*. The base product provides a business object that you may use or extend or you may create your own. Refer to the BO definition in the application.

Open **Admin Menu** > **Tax Role Relationship Type** to define the tax role relationship types used to classify persons related to a tax role.

The topics in this section describe the base-package zones that appear on the tax role relationship type portal.

Tax Role Relationship Type List

The Tax Role Relationship Type *List zone* lists every tax role relationship type. The following functions are available:

- Click the *broadcast* icon to open other zones that contain more information about the adjacent tax role relationship type.
- The standard actions of **Edit, Duplicate** and **Delete** are available for each tax role relationship type.

Click the **Add** link in the zone's title bar to add a new tax role relationship type.

Tax Role Relationship Type

The Tax Role Relationship Type zone contains display-only information about a tax role relationship type. This zone appears when a tax role relationship type has been broadcast from the Tax Role Relationship Type List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *C1_TAX_ROLE_REL_TYPE*.

Setting Up Financial Relationship Types

In addition to a tax role relationship type, miscellaneous persons related to a tax role may also have an optional financial relationship type. You can configure those relationship types as Financial Relationship Type extendable lookup values.

- Open **Admin Menu** > **Extended Lookup**.
- Search for and select the **Financial Relationship Type** extendable lookup business object.
- The list of existing financial relationship types are displayed in a standard *List zone*.
- Choose an existing financial relationship type to view, edit, delete or duplicate.
- Use the **Add** link in the zone header to create a new financial relationship type.

Setting Up Industry Codes

Tax roles and obligations can reference industry codes. This code is used to categorize tax roles and obligations for reporting purposes. To define an industry code, open **Admin Menu** > **Industry Code**.

Description of Page

Enter a unique **Industry Code** and **Description**.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_SIC*.

Financial Controls

There are also a number of control tables that must be set up to control the bills, payments, and adjustments that are linked to an obligation. For more information about these tables, please refer to *Defining Financial Transaction Options*.

Configuring Obligation Types

This section provides information on how to configure obligation types.

Designing Obligation Types

The topics in this section provide guidelines describing how to design obligation types. When designing obligation types you will want to consider what the obligation will be used for. Generally speaking, obligation types can be separated into two categories.

- The first type of obligation type relates to the specific tax types of a tax authority. Examples of these include corporate tax and property tax. The specific tax type dictates how bills are handled and what collections activities are available to the tax authority.
- The second type of obligation relates to specific events or transactions that require certain follow-up actions from the tax authority or taxpayer (or both). Examples of these include overpayment obligations and payment plan obligation. With overpayments, the tax authority has the responsibility to properly handle the excess credit a taxpayer has created with a payment. Payment plans are the responsibility of the taxpayer in that they are required to send in a series of payments on a predefined schedule. Each obligation must be monitored to ensure proper compliance.

Filing Period Obligations

Filing period obligations are one of the tax type specific obligation types. Examples of these tax types include individual income, sales and use, corporate, withholding, and fuel taxes. These are examples of return based taxes in which the taxpayer has the responsibility to file a return for each filing period obligation. There are usually statutory considerations when configuring these obligation types. You will want to consider the following when designing your obligation for these types of taxes.

- The obligation type must reference a defined *tax type*. The specific tax type will dictate the rules for that obligation. It should indicate that a Tax Role is required and should include the valid filing calendars for the tax type.
- The obligation type should be configured to indicate that a filing period is required.
- Distribution code and general ledger division for the obligation directs how the money is accounted. Different taxes may be moved to different accounts. While an individual income tax may go to the general treasury fund, a fuel tax may go the highway construction fund. The distribution code will reference an algorithm that directs how the money will be accounted for.
- Adjustment profile must be defined for the obligation. Adjustment profiles contain the individual listing of adjustment types that are available for that obligation type.
- An appropriate P&I control and P&I algorithms should be defined for the obligation type. Refer to *P&I Rules for a P&I Control Define the Calculation* for more information.
- Define the default debt category priority for credits linked to obligations of this type. Refer to *Debt Categories and their Priorities* for more information.

Each tax type will have its own unique attributes that will change how the obligation is set up. The more details you add to your obligation type, the more robust your processing of that obligation will be. Before configuring for this obligation type you should thoroughly review the statutory considerations and make sure they are all addressed when setting up your obligation type.

Property Tax Obligations

Property tax obligations are another type of tax specific obligation type. Property tax obligations are based off the assessed value of a taxpayer's asset. Typically they are items such as real estate or automobiles but can include boats, RV's, and other high value items. They differ from filing period obligations in that it is the tax authority's responsibility to assess the value and calculate the amount of tax owed. It is then the taxpayer's responsibility to either pay or appeal the assessed value. The considerations are very similar to filing period obligations.

- The obligation type must reference a defined tax type. The specific tax type will dictate the rules for that obligation.
- Payment plans are often utilized for property tax obligations. Due to their infrequent assessment and billing (once or twice a year) the amount is usually large enough where a significant number of taxpayers would have difficulty paying. If payment plans are utilized then make sure to check that option under the billing tab.
- Distribution code and general ledger division for the obligation directs how the money is accounted. Different taxes may be moved to different accounts. While an individual income tax may go to the general treasury fund, a fuel tax may go the highway construction fund. The distribution code will reference an algorithm that directs how the money will be accounted for.
- Rates may be utilized more often for property taxes. Property taxes tend to be a locally administered tax. The breakdown of how the tax is applied to local services can be done using the rate schedule option under the rate tab.
- Adjustment profile must be defined for the obligation. Adjustment profiles contain the individual listing of adjustment types that are available for that obligation type.

Overpayment Obligations

When a taxpayer pays more than they owe, you must decide what to do with the excess money. The following points describe two possibilities:

- You could create a new obligation to hold the excess (let's call it an overpayment obligation). The credit would need to be transferred from this obligation to the tax obligations at an appropriate time.
- You could direct the excess payment on one of the existing tax obligations.

You control which method is used by plugging in the appropriate **Overpayment Distribution** algorithm on each *account type* (i.e., you can choose a different method for different account types). If you choose to hold overpayments on a separate obligation, then you must set up an obligation type.

The following points highlight interesting information about overpayment obligation types:

- It should have an accounts payable distribution code. This is because when a payment segment is created for this type of obligations, the system must credit a liability (an overpayment is a liability).
- It's important to indicate that the overpayment obligation is a one-time obligation. Why? Because this means that the system will automatically close the obligation when it's balance falls to zero (i.e., when appropriate business processes transfer all of the overpayment to tax obligations).
- A bill segment type is not needed because the system never creates bill segments for such obligations (they exist only to hold excess credits).
- P&I controls and the P&I related algorithms are not needed.
- You may also want to turn on the alert message
- You must reference this overpayment obligation type as the parameter value on your overpayment algorithm (this algorithm is plugged in on the desired account types). Refer to *Overpayment Algorithm* for more information about this algorithm.
- You must design appropriate business procedures to use this overpayment when additional debt is incurred on tax obligations.

Write Off Obligations

Some agencies may choose to transfer written of debt from the "normal" obligation onto one or more write-off obligations. When the debt is transferred to a write-off obligation, the distribution code on the "normal" obligation is credited (typically an A/R GL account), and the distribution code on the write-off obligation is debited.

You will almost always need a write-off obligation whose distribution code is the write-off expense. However, if you don't practice cash accounting, you may have uncollectible debt for liabilities (and you don't owe the liability if you don't get paid). In this case you'll need another obligation type for the liabilities.

The following points highlight interesting information about write-off obligation types:

- The distribution code is either an expense or a liability account. This is because when debt is transferred to these types of obligations, the system must debit either an expense account (i.e., write-off expense) or a liability account. It's important to note that in The Ramifications of Write Offs in the General Ledger we explain how this liability account may be overwritten with the liability account that was originally booked.
- They do not need a bill segment type because the system never creates bill segments for such obligations (they exist only to hold uncollectable debt)

NOTE:

The adjustment type used to set the offending obligation's current balance equal to its payoff balance is defined on each write-offable obligation type. The adjustment type used to transfer the delinquent debt to the write-off obligation is defined on the write-off obligation type.

NOTE:
An Alternative. If you have a limited number of liability accounts, you may decide to have a separate write-off obligation for each liability account. Doing this would proliferate the number of obligations created at write-off time. However, it would simplify the remittance of payment to the third party if the reversed liability is ever paid.

Pay Plan Obligations

If you allow your customers to pay overdue debt using a payment plan, you need to set up obligation types for pay plan obligations.

FASTPATH:
For more information about pay plans, refer to *Defining Pay Plan Options*.

Over/Under Cash Drawer Obligations

In order to balance a tender control that is out-of-balance, your organization must set up an account with an obligation whose obligation type references the over/under expense account. You will probably only have one obligation that references this obligation type, but you still must have it if you remit funds via a cash drawer.

FASTPATH:
For more information about over/under processing, refer to *How To Get An Unbalanced Tender Control In Balance (Fixing Over/Under)*.

The following points highlight interesting information about this obligation type:

- It has an expense distribution code. This is because when a payment segment is applied to this type of obligation, the system must debit an expense account for under amounts (and credit it for over amounts).
- It doesn't need a bill segment type because the system never creates bill segments for such obligations (it only has over/under payment segments linked to it).

Payment Upload Error Obligations

If the payment upload process detects an invalid account on a payment upload record, it will create a payment for the suspense obligation defined on the upload process' tender source (see *Setting Up Tender Sources*). You should create a special obligation type for this obligation.

FASTPATH:
For more information about the payment upload process, refer to *Phase 3 - Create Payment Events, Tenders, Payments and Payment Segments*.

The following points highlight interesting information about this obligation type:

- It has an expense distribution code. This code should probably be a suspense account. All payment segments that are created for this obligation will eventually be transferred to a "real" obligation and therefore this GL account's balance should be zero when no payments are in suspense.

- It doesn't need a bill segment type because the system never creates bill segments for such obligations (it only has invalid account payment segments linked to it).

Billable Charge Obligations

You create a billable charge whenever a taxpayer should be charged for a service that occurs outside the normal course of business. For example, if a taxpayer requires a review of their property assessment, you may charge them an administration fee. You can also use billable charges to "pass through" other bill ready charges generated outside the system, by another application, or by a 3 rd party supplier.

A billable charge must reference an obligation. This obligation behaves just like any other obligation:

- **Bill segments are created for the obligation.** Whenever billing is performed for an account with billable charge obligations, the system creates a bill segment for each obligation with unbilled charges. If multiple unbilled charges exist for a given obligation, only one bill segment will be created and it will contain details about all of the billable charges.
- **Payments are distributed to the obligation.** Payments made by an account are distributed to its billable charge obligations just like any other obligation.
- **Overdue debt is monitored.** The credit and collections process monitors billable charge obligations for overdue debt and responds accordingly when overdue debt is detected.

Therefore, you must set up at least one obligation type to hold your billable charge debt. You may have multiple charges based on billing frequencies, A/R booking, debt monitoring, etc. It's really up to you.

The easiest way to determine how many billable charge obligation types you'll need is to define every conceivable billable charge (which you should have done when you designed your billable charge templates). Then ask yourself if they have the same billing and payment behavior, if so, you'll have one obligation type. If not, you'll need one obligation type for each combination.

Expiring Obligations

It is possible to assign an expiration date to an obligation. This wouldn't apply to an obligation used to capture amounts owed for a specific tax period. But it could apply to an obligation that covers an ongoing financial relationship, such as a license fee.

For obligations that have an expiration date, an Obligation Expiration background process (batch control **SAEXPIRE**) is used to automatically 'stop' the obligation when the expiration date arrives.

Renewing Obligations

An obligation type may be configured to allow obligations of that type to renew. This would only apply to an obligation that supports the concept of expiration. It is also supported on pay plan if the pay plan is being used as a proactive payment collection mechanism (rather than as a method of recovering unpaid debt for other obligations). Renewal may be performed manually or via a background process. When an obligation is created, the expiration date and renewal date may be defined for the obligation. A renewal flag on the obligation 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 obligation. The renewal date is defaulted on to an obligation based on the valued of the Days Before Expiration for Renewal field on the obligation type.

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

The Obligation Renewal background process (batch control **SARENEW**) does the following:

- Executes the obligation renewal algorithm (specified on the obligation type) when the renewal date is reached (i.e., it is on or before the process date). This algorithm is responsible for determining the new expiration and renewal dates.
- If the renewal algorithm is successful, the renewal and expiration date fields on the obligation / pay plan are updated with the new values.

- For pay plan obligations, a new payment schedule may be returned from the renewal algorithm and is appended to the existing schedule.

A user can manually launch the renewal process for a pay plan obligation by clicking **Renew** on the pay plan maintenance page.

Setting Up Obligation Types

This section explains how to create and maintain obligation types.

Obligation Type - Main Information

Open **Admin Menu** > **Obligation Type** to define core information about your obligation types.

Description of Page

Enter a unique combination of **Division** and **Obligation Type** for every obligation type.

Enter a **Description** for the obligation type.

Tax Type defines the tax type for the obligation type.

FASTPATH: For more information about tax types, refer to *Setting Up Tax Types*.

Use an **Obligation Business Object** to define a *BO* that may govern additional rules related to obligations of this type.

Indicate if the **Revenue Period Applicability** for obligations of this type is **Required** or **Not Allowed**.

Select the **Distribution Code** and **GL division** that defines the receivable account for receivable-oriented obligations. For non-receivable oriented obligations, this distribution code depends on the obligation type's function. Refer to *Designing Obligation Types* for some examples.

Select the **Revenue Class** associated with the obligation type (and its obligations). The revenue class may affect the revenue account(s) generated by the obligation's rate.

NOTE: The revenue class field is only visible if your implementation has *turned off* the **C1TB — Billing** module. This information is used only for classic billing and rates functionality.

FASTPATH: Refer to *Rate Component - GL Distribution* for more information about revenue class.

Set the **Eligible for Pay Plan** flag to **Eligible** if you want obligations of this type to be eligible to be covered by a pay plan.

Select the **Payment Segment Type** that defines how payment segments linked to obligations of this type affect the obligation's payoff and current balances and how the general ledger details for the payment are built.

FASTPATH: For more information about payment segment types, refer to *Setting Up Payment Segment Types*.

When a tender is canceled, a cancellation reason must be supplied. If the cancellation reason indicates a NSF (non sufficient funds) charge should be levied, the system invokes the Levy an NSF Charge algorithm specified on the tender's account's *account type*. Because adjustments must be linked to an obligation, the algorithm must determine the appropriate obligation to use to levy the adjustment based on business rules. The charge is levied using the **NSF Adjustment Type** of the appropriate obligation's obligation type.

CAUTION: You must specify adjustment type profiles on the obligation type (on the Adjustment Type window) to ensure that the adjustment type defined here is valid.

FASTPATH: For more information about adjustment types, refer to *Setting Up Adjustment Types*.

Define the **Payment Grace Days** and **Override Filing Grace Days** applicable for obligations of this type. Refer to *Due Dates for Filing Period Based Obligations* for more information about due dates.

Select the **Payment Priority**. This field is available for use by the algorithms that distribute partial payments amongst an account's obligations. Higher priority obligations will have their debt relieved before lower priorities. Refer to *Distribution Based on Payment Priority* and *Delinquent Payment Distribution Algorithm* for information about payment distribution algorithms that use this field.

NOTE: The values for this field are customizable using the Lookup table. This field name is PAY_PRIORITY_FLG.

FASTPATH: For more information about distribution priority, refer to *Distributing A Payment Amongst An Account's obligations*.

Select the **Delinquent Payment Priority**. This field is available for use by the algorithms that distribute partial payments amongst an account's obligations. Higher priority obligations will have their debt relieved before lower priorities. Refer to *Delinquent Payment Distribution Algorithm* for information about a payment distribution algorithm that uses this field.

NOTE:
The values for this field are customizable using the Lookup table. This field name is DEL_PRIORITY_FLG.

Define the default **Debt Category** to assign to bill segment type financial transactions for obligations of this type. Refer to *Debt Categories and their Priorities* for more information.

NOTE: **Required for base algorithms**. The base P&I calculation algorithm and the base *Determine Detailed Balance* algorithm require that every debit financial transaction reference a debt category. This field applies to both standard and classic bill segment financial transactions.

Define the default **Debt Category Priority** for credit financial transactions linked to obligations of this type. Refer to *Debt Categories and their Priorities* for more information.

Turn on **Do Not Overpay** if the system is not allowed to distribute an overpayment to this type of obligation (i.e., the obligation is not allowed to have a system-created credit balance). This field is available for use by algorithms that distribute overpayments. Refer to *Overpayments Held On Highest Priority Obligation* for information about an overpayment algorithm that uses this field.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_SA_TYPE*.

Obligation Type - Detail

Open **Admin Menu, Obligation Type** and navigate to the **Detail** tab to define additional details about a given obligation type.

Description of Page

Turn on **Display As Alert** if an alert should appear in the alert zone if an account has an obligation of this type that isn't **Closed** or **Canceled**. If this switch is on, also enter the **Alert Information** to appear in the zone. We recommend only using this feature on unusual obligation types (e.g., write-offs) so that a user is not presented with an alert for every obligation type.

If this obligation type is used for any of the special roles defined in the drop down of **Special Role Flag**, indicate which one. Valid values are: **Pay Plan**, **Write Off** and **Billable Charge**. This information is used on windows with functionality that can only be used by obligations used for specific roles. For example, the Pay Plan window group can only reference **Pay Plan** obligations.

NOTE: The values for this field are customizable using the Lookup table. This field name is SPECIAL_ROLE_FLG.

NOTE: The **Billable Charge** special role value is only visible if your implementation has not *turned off* the **BI — Billing (Classic)** module. This role applies only to classic billing functionality. Refer to the **Where Used** section below for more information on **Billable Charge** obligation types.

If the Special Role is **Write Off**, you must also define the following adjustment type:

- Use **Adjustment Type (Xfer)** to specify the type of adjustment used to transfer funds from the uncollectable obligations to the write off obligation.

Turn on **One Time Charge** if this obligation type is used for one-time invoices. When a one-time invoice obligation is created, the system sets the stop date of the obligation to be equal to the start date.

NOTE: The One Time Charge field is only visible if your implementation has not *turned off* the **BI — Billing (Classic)** module.

Set **Renewal** to **Not Allowed, Optional** or **Required** depending on the business rules for obligations of this type. Renewal logic is not common for revenue period based obligations because those types of obligations typically cover a specific revenue period. Renewal may be applicable for special obligations that are used to track a long term financial obligation like a license and have an expiration date.

If renewal is required or optional, specify the **Renewal Days Before Expiration**.

Where Used

The alert information is used by the alert zone to alert a user when unusual obligations exist for an account. Refer to *Alert Zone* for more information.

Only obligation types designated as being **Billable Charge** may have billable charges linked to them. Refer to *Maintaining Billable Charges* for more information.

Only obligation types designated as **Pay Plan** may be used to set up pay plans. Refer to *Pay Plans* for more information.

Only obligation types designated as being **Write Off** may be specified as the write off obligation type on distribution codes. Refer to *Setting Up Distribution Codes* for more information.

Obligation Type - Billing

Open **Admin Menu, Obligation Type** and navigate to the **Billing** tab to define how the system manages classic bill segments for obligations of a given obligation type.

NOTE: **Optional information.** This tab is only visible if your implementation has not *turned off* the **BI — Billing (Classic)** module. The information is used only for classic billing functionality.

Description of Page

Turn on **Eligible for Billing** if the system should create bill segments for obligations of this type.

Define the minimum number of days a bill segment (other than the final segment) must span using **Minimum Days for Billing**. This is useful to prevent initial bill segments that span only a few days.

FASTPATH: For more information about minimum days, refer to *Preventing Short Bill Segments*.

Select the **Bill Segment Type** that controls both how classic bill segments for this obligation type will be created and how the related financial transaction affects the general ledger and the taxpayer's debt.

FASTPATH: For more information about classic bill segment types, refer to *Setting Up Bill Segment Types*.

Use **Default Description on Bill** to define the verbiage that should print on the taxpayer's bill.

NOTE: **Rates overwrite this description.** The Default Description on Bill is not applicable for obligations whose charges are calculated using a rate. Why? Because the description that appears on the bill segment is defined on the rate schedule's rate version.

NOTE: **Billable charges overwrite this description.** The Default Description on Bill is not applicable for obligations whose charges are calculated using a billable charge. Why? Because the description that appears on the bill segment is defined on the billable charge.

Use the **Billing Processing Sequence** if you need to control the order in which obligations linked to this obligation type are processed by classic billing processes.

Use **Bill Print Priority** to define the order in which the obligation type's bill segments should appear on bills (relative to the other obligation types that appear on a bill).

NOTE: The values for this field are customizable using the Lookup table. This field name is BILL_PRT_PRIO_FLG.

Use **Max Bill Threshold** if you want the system to generate a bill error when a bill segment is produced in batch that exceeds a given value. These bill errors will appear on the standard billing queries and To Do lists. If, after reviewing the high value bill segment, an operator truly intends to send the bill out, they should regenerate the bill. Refer to *How To Correct A Bill Segment That's In Error* for more information.

CAUTION: The value entered in this field will DEFAULT onto obligations of this type when they are first created. An operator may change the default value on an obligation in case a specific taxpayer has unusually high bills that continually error out. It's important to be aware that if you change the value of High Bill Amount on an obligation type and there already exist obligations of this type, the existing obligations will contain the original value (the new value on the obligation type will not be propagated on the existing obligations).

Turn on **Characteristic Location Required** if a characteristic location must be linked to the obligation when the obligation is activated. The characteristic location is used to define the taxing authorities associated with the obligation's classic bill segments. It is also used to identify where the obligation's service is located on various windows.

NOTE: This is only applicable for systems that are configured for *legacy address* functionality.

FASTPATH: For more information about how characteristic location is used, refer *An Illustration Of A Rate Factor And Its Characteristics*.

Use the **Initial Start Date Option** to control how billing should calculate the calculation period for the very first bill for obligations of this type. This field is important if your obligation has rate components that include daily charges. Set the field to **Include First Day** if the obligation's start date should be included in the daily charges. Set the value to **Add 1 Day Always** if the obligation's start date should never be included in the daily charges. Set the value to **Add 1 Day for Back-to-back** if the obligation's start date should only be included in the daily charges if no previous taxpayer was responsible for the charge (for example for property tax). This field is not applicable for obligation types with a special role of **Billable Charge**.

Obligations may have the end date of their bill segments defined on a user-maintained bill period schedule. This option is used when bill segments must fall on strict calendar boundaries (e.g., quarterly bills that end on the last day of the quarter). If this obligation type should be billed like this, select **Use Bill Period** in the **Use Calendar Billing** field. When this option is used, you must define the **Bill Period** whose schedule defines the bill segment end dates.

FASTPATH: For more information about bill period schedules, refer to *Defining Bill Cycles and Bill Periods*. For more information about other bill end date methods, refer to *Ways To Control The End Date Of A Bill Segment*.

Instead of the **Use Bill Period** method, obligations may have their bill segment end date based on a specified date. If this obligation type should be billed like this, select **Anniversary Future Billing** or **Anniversary Past Billing** in the **Use Calendar Billing** field. When either option is used, you must define the **Anniversary Bill Frequency**. This frequency defines the amount of time between bill segments.

FASTPATH: For more information about anniversary billing, refer to *Using The Anniversary Method*. For more information about other bill end date methods, refer to *Ways To Control The End Date Of A Bill Segment*.

Total Bill Amount indicates whether obligations of this type can use the total amount to bill field on the obligation page. Valid values are **Not Allowed** and **Required**.

If **Required** is selected, you must enter the **Total Amount To Bill Label**. The **Total Amount To Bill Label** defines the label that prefixes the total bill amount on the obligation page for obligations of this obligation type.

Recurring Charge indicates whether obligations of this type can use the recurring charge field on the obligation window. Valid values are **Not Allowed**, **Optional** and **Required**. If either **Optional** or **Required** are used, you must enter:

- **Recurring Charge Amount Label**. This defines the label that prefixes the recurring charge amount on the obligation window for obligations of this obligation type.
- **Recurring Charge Frequency**. This defines the following:
  - Specifies the frequency at which the Recurring Charge Amount specified on obligations of the obligation type is to be billed.
  - Serves as the basis for proration of the Recurring Charge Amount.
  - Specifies the frequency at which obligations of the obligation type without a rate will be billed.

Obligation Type - Rate

Open **Admin Menu, Obligation Type** and navigate to the **Rate** tab to define the rates that may be referenced on obligations of a given type.

NOTE: **Optional information.** This tab is only visible if your implementation has not *turned off* the **BI — Billing (Classic)** module. The information is used only for classic billing functionality.

Description of Page

Turn on **Rate Required** if the bill segment creation algorithm for the obligation type expects a rate schedule to be referenced on obligations of this type.

FASTPATH:
For more information, refer to *Rates*.

Define the date the system uses when selecting an effective-dated rate (from the obligation's rate history) using **Rate Selection Date**. Selecting **Bill Start Date** will cause the system to use the rate effective on the first day of the bill segment's calculation period. Selecting **Bill End Date** will cause the system to use the rate effective on the last day of the bill period. Selecting **Accounting Date** will cause the system to use the rate effective on the accounting date associated with the bill.

The information in the **Rate Schedules** collection defines the rates that may be referenced on obligations of this type. The following fields are required for each obligation type:

Rate Schedule	Specify the rate schedule; its description is displayed adjacent.
Use Rate As Default	Turn on this switch for the rate to be defaulted on new obligations.

Where Used

This information is used to default and validate the rate specified on an obligation. Refer to *Obligation - Rate Info* for more information.

Obligation Type - Adjustment Profiles

Open **Admin Menu, Obligation Type** and navigate to the **Adj Profile** tab to define the adjustment profiles that define adjustment types that may be referenced on obligations of a given type.

Description of Page

Define the **Adjustment Type Profiles** that, in turn, define adjustment types that may be referenced on obligations of a given type.

FASTPATH:
For more information about adjustment type profiles, refer to *Setting Up Adjustment Type Profiles*.

Where Used

This information is used to validate the adjustments linked to the obligation. Refer to *Adjustments - Main Information* for more information.

Obligation Type - Characteristics

To define characteristics common to all obligations of a given type, open **Admin Menu, Obligation Type** and navigate to the **Characteristics** tab.

Description of Page

Use the characteristics collection to define a **Characteristic Type** and **Characteristic Value** common to all obligations of this type.

NOTE:
You can only choose characteristic types defined as permissible on an obligation type record. Refer to *Setting Up Characteristic Types & Their Values* for more information.

Obligation Type - Algorithms

Open **Admin Menu** > **Obligation Type** and navigate to the **Algorithm** tab to define the algorithms that should be executed for obligations of a given type.

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.

CAUTION: Warning! These algorithms are typically significant processes. The absence of an algorithm may prevent the system from operating correctly.

The following table describes each **System Event** for which you can define algorithms.

System Event	*Optional / Required*	*Description*
Bill Completion	Optional	These algorithms are executed whenever a bill is completed for an account that contains a non-canceled obligation of this type. **Note**. Algorithms of this type are called for all non-**Canceled** obligations, regardless of whether or not they are billed. If your algorithms should only be processed under certain conditions (for example, only process this algorithm for **Active** obligations), then it is the responsibility of the algorithm to check the conditions before continuing. Click *here* to see the algorithm types available for this system event.
Break Pay Plan Obligation	Optional	These algorithms are executed when a *pay plan* is manually stopped via the *pay plan maintenance page*. Click *here* to see the algorithm types available for this system event.
Cash Accounting True Up	Optional	This algorithm is responsible for making all necessary adjustments to the general ledger to transfer amounts from "holding" general ledger accounts to true payable accounts. It is executed when the Calculate P&I service has been called in Update mode, regardless of whether or not P&I is applicable for the obligation type. Refer to *P&I and Cash Accounting* for more information. Click *here* to see the algorithm types available for this system event and for more information about the behavior of this plug-in spot.
Close / Reactivate Criteria	Optional	This algorithm is invoked from the common routine that performs closing / reactivation logic for an obligation. The common routine is invoked from the close action on the obligation maintenance page, when an account's obligations are monitored for overdue debt and whenever an FT is added, canceled or frozen for the obligation. It checks any additional conditions that are necessary to determine if an obligation of this type can be closed or reactivated. Only the first algorithm plugged in on this spot will be executed. Click *here* to see the algorithm types available for this system event.
Determine Detailed Balance	Optional	This plug-in spot is responsible for breaking down an obligation's balance into balances by separate assessments and debt categories (for example: tax, penalty, interest, fees, etc). It is used in the base **P&I Calculation** plug-in and several other services that require the balance of an obligation to be broken down by assessment and debt category.

		Click *here* to see the algorithm types available for this system event and for more information about the behavior of this plug-in spot.
FT Freeze	Optional	These algorithms are executed whenever a financial transaction is frozen that is linked to an obligation of this type. Click *here* to see the algorithm types available for this system event.
Generate Obligations	Optional	This plug-in spot is responsible for generating Obligations of the associated Obligation Type for a Tax Role. It is invoked by the common service that generates obligations for a tax role. In addition to creating obligations for periods up to and including the current business date, algorithms plugged into this spot must indicate the date on which the next obligation of this type is due to be created for the tax role. Refer to*Tax Roles Can Maintain Obligations* for more information. Click *here* to see the algorithm types available for this system event and for more information about the behavior of this plug-in spot.
Obligation Activation	Optional	These algorithms are executed when an obligation status changes from **Pending Start** to **Active**. It performs any additional activities that are necessary to activate an obligation. Click *here* to see the algorithm types available for this system event.
Obligation Cancel	Optional	These algorithms are executed when an obligation status changes to **Canceled**. It performs any additional activities that are necessary when an obligation is canceled. Click *here* to see the algorithm types available for this system event.
Obligation Creation	Optional	These algorithms are executed when an obligation is created. Click *here* to see the algorithm types available for this system event.
Obligation Information	Optional	We use the term "obligation information" to describe the basic information that appears throughout the system to describe an obligation. The data that appears in "obligation information" may be constructed using an algorithm plugged in here. Refer to *Defining Obligation Information* for more information on when this algorithm is used. Click *here* to see the algorithm types available for this system event.
Obligation Renewal	Optional	These algorithms are executed by the Obligation Renewal background process whenever an obligation is due for renewal or when the user clicks the Renew button. It performs any activities that are necessary to renew an obligation and returns the new renewal and expiration dates for the obligation. Click *here* to see the algorithm types available for this system event.
Obligation Stop	Optional	These algorithms are executed whenever an obligation's status changes from **pending stop** to **stopped**. Click *here* to see the algorithm types available for this system event.

Obligation Stop Initiation	Optional	These algorithms are executed whenever an obligation's status becomes **pending stop**. Click *here* to see the algorithm types available for this system event.
P&I Calculation	Optional	This plug-in spot is responsible for *calculating penalty and interest* for the input obligation. Click *here* to see the algorithm types available for this system event and for more information about the behavior of this plug-in spot.
P&I Eligibility	Optional	This plug-in spot is invoked by the base Calculate Penalty and Interest plug-in. It receives an obligation id as input and returns an indication of whether the obligation is eligible or ineligible for P&I. Refer to *P&I Eligibility* for more information. Click *here* to see the algorithm types available for this system event and for more information about the behavior of this plug-in spot.
P&I Post Processing	Optional	This plug-in spot is invoked by the base P&I Calculation plug-in after all the penalty and interest calculations are finished. Algorithms plugged into this plug-in spot are responsible for additional logic performed at the end of the P&I process. Refer to *P&I Post Processing* for more information. Click *here* to see the algorithm types available for this system event and for more information about the behavior of this plug-in spot.
P&I Pre-processing	Optional	This plug-in spot is invoked by the base P&I Calculation plug-in and is used to populate information needed during P&I calculations. Refer to *P&I Pre-Processing* for more information. Click *here* to see the algorithm types available for this system event and for more information about the behavior of this plug-in spot.
P&I Prepare Periodic Balance	Optional	This plug-in spot is called from the Calculate Penalty and Interest plug-in spot for each date range being calculated. It is responsible for passing the appropriate FTs to the **Determine Detailed Balance** plug-in for the current date range. Click *here* to see the algorithm types available for this system event and for more information about the behavior of this plug-in spot.
Payment Freeze	Optional	These algorithms are executed whenever a payment is frozen. Click *here* to see the algorithm types available for this system event.
Process Pay Plan Scheduled Payment	Optional	These algorithms are executed by the Pay Plan Scheduled Payment Processing background process whenever a scheduled payment is due. If the pay plan obligation is unmonitored, this algorithm is not called. This algorithm should be specified on pay plan obligation types to create the necessary adjustments for the pay plan obligation. Click *here* to see the algorithm types available for this system event.
Retrieve FT List	Optional	This plug-in spot is responsible for retrieving the existing FTs for an obligation. It obtains all the data for each FT that is needed for **P&I Calculation** logic and the **Determine Detailed Balance** logic. Refer to the *Big*

Picture of Penalty and Interest for more information.

Click *here* to see the algorithm types available for this system event and for more information about the behavior of this plug-in spot.

Obligation Type - Pay Plan Recommendation Rule

Open **Admin Menu, Obligation Type** and navigate to the **Pay Plan Rec'n Rule** tab to define the recommendation rules that are valid for pay plan obligations of this type.

Description of Page

If the obligation type's special role is **Pay Plan**, you may define the **Recommendation Rules** that are valid on pay plan obligations of this type. Check the **Use As Default** box to indicate the default recommendation rule for obligations of this type.

FASTPATH:
For more information about pay plans, refer to *Defining Pay Plan Options*.

Where Used

The pay plan recommendation rules are used to recommend the payment amount and payment schedule for pay plan obligations. Refer to *Maintaining Pay Plans* for more information.

Obligation Type - P&I Control

Open **Admin Menu, Obligation Type** and navigate to the **P&I Control** tab to define the penalty and interest control record that governs the *penalty and interest calculation* rules for obligations of this type.

Description of Page

The P&I Control and its P&I Rules define the configuration governing the tax authority's penalty and interest calculations. As these rules change over time, a new P&I Control record with the new applicable P&I Rules is created and linked to the applicable obligation types for the effective date of the change.

The following fields display:

- **Effective Date** is the date the P&I Control rate becomes effective.
- **P&I Control** defines the P&I rules used to calculate the individual penalty and interest charges.

NOTE:
P&I configuration. Refer to *Setting Up Penalty and Interest Options* for more detail about the configuration required for calculating penalty and interest.

Where Used

This information is used by the P&I Calculation plug-in to apply specific rules.

Obligation Type - BC Template

Open **Admin Menu, Obligation Type** and navigate to the **BC Template** tab to define the billable charge templates that can be used on obligations of a given type.

NOTE: Optional information. This tab is only visible if your implementation has not *turned off* the **BI — Billing (Classic)** module. The information is used only for classic billing functionality.

NOTE:
Only billable charges have billable charge templates. Only obligations that are defined as Billable Charges (in the Special Role on the Details window) may use the grid on this window.

Description of Page

The information in the **Billable Charge Template** collection defines the obligation type's permissible billable charge templates. A billable charge template contains the default bill lines, amounts and distribution codes used to levy a one-off charge. The following fields are required for each template:

Billable Charge Template	Specify the billable charge template. Its description is displayed adjacent.
Use As Default	Turn on this switch for the template to be defaulted on new billable charges linked to obligations of this type (if any).

Where Used

This information is used to limit the billable charge templates that can be used for a given obligation type.

Obligation Type - BC Upload Overrides

The *BCU2 - Create Billable Charge* background process is responsible for creating billable charges for each billable charge upload staging record interfaced into the system. This process will override the values of the various switches referenced on a billable charge upload staging line if the respective obligation's obligation type has an override value for the billable charge upload staging line's billable charge line type.

NOTE: Optional information. This tab is only visible if your implementation has not *turned off* the **BI — Billing (Classic)** module. The information is used only for classic billing functionality.

NOTE:
If you don't need to override the values of a *Billable Charge Line Type* you don't need to set up this information.

Open **Admin Menu, Obligation Type** and navigate to the **BC Upload Override** tab to define override values for a given Obligation Type / Billable Charge Upload Staging Line Type.

Description of Page

Use the **Billable Charge Overrides** collection to define values to be overridden on billable charge lines uploaded from an external system (refer to the description above for the details). The following switches may be overridden for a given **Obligation Type** and **Billable Charge Line Type.**

- Use the **Show on Bill** switch to define the value to be defaulted into the Show on Bill indicator on billable charge upload lines that reference this line type.
- Use the **Appear in Summary** switch to define the value to be defaulted into the App in Summary indicator on billable charge upload lines that reference this line type.
- Use **Memo Only, No GL** switch to define the value to be defaulted into the Memo Only, No GL indicator on billable charge upload lines that reference this line type.

- Use **Distribution Code** to define the value to be defaulted into the Distribution Code on billable charge upload lines that reference this line type.

Defining Obligation Information

The obligation information displayed throughout the system is controlled by a plug-in.

The system first looks to see if the obligation type references an obligation business object and if the business object defines an information plug-in. If a BO is not provided or if that BO does not define an information algorithm, the system looks for an information algorithm plugged into the obligation maintenance object.

If no plug-ins are found on the BO or the MO, the system looks for a plug-in algorithm on the *Obligation Type*.

If such an algorithm is not plugged-in on the Obligation Type, the system looks for a corresponding algorithm on the *installation record*.

If you prefer different formatting logic, your implementation should provide a plug-in at one of the above plug-in spots, as appropriate for your business rules.

Setting Up Customer Contact Options

This section describes tables that must be set up before you can define customer contacts.

FASTPATH:
Refer to *The Big Picture Of Customer Contacts* for more information about taxpayer (customer) contacts.

Letters Via Customer Contacts

Customer contacts may be configured to cause a letter to be printed. These types of customer contacts reference a letter template, which configures various aspects of how the letter is printed. This section contains additional information about configuration for customer contacts that are letters.

Letter Printing

Letters are printed as a result of a customer contact that is specially configured for letters. The contents of this section describe the technical implementation of letter production.

Letter Templates Control The Information Merged Onto Letters

Customer contacts that trigger letters reference a *customer contact type* that, in turn, references a *letter template*. The letter template controls the following:

- It contains an algorithm that is responsible for extracting the information merged onto your letters. Algorithms of this type are called under the following scenarios:
    - The *background process* that builds the flat file that's passed to your letter print software calls these algorithms to construct the supporting details for each letter.
    - If your letter print software has the ability to construct a real-time image of a letter (in a PDF), you can plug-in an **Online Letter Image** algorithm on the *Installation Record*. This type of algorithm will call the letter's letter template's extract algorithm to extract the information that is merged onto the letter. Refer to *Technical Implementation Of Online Letter Image* for the details.

NOTE: The product provides a sample extract algorithm to integrate with Documaker. The sample integration uses the letter template code to map to the Documaker template.

- It contains the ID of the background process that builds the flat file that's passed to your letter print software. The base package example of this process (known by the batch control ID of **LTRPRT**) simply calls each customer contact's letter template's extract algorithm to format the information placed onto the flat file.

Technical Implementation Of Batch Letter Production

The batch process that extracts letter information (known by the batch control ID of **LTRPRT**) reads all customer contact records in a given run number that are marked with its batch control ID. For each customer contact, it creates numerous records on a flat file. These records contain the taxpayer information that is merged onto your letters.

The information that is placed on each record is controlled by the customer contact's letter template's extraction algorithm. Refer to *Letter Templates Control The Information On Letters* for more information.

The records on the flat file can be constructed in either a fixed-position or field-delimited format. The format that's used is controlled by a parameter supplied to the **LTRPRT** background process.

NOTE:
Print / routing methods. If you require different print formats, you must create new letter extract algorithms and plug them in on your *letter template*.

Technical Implementation Of Online Letter Images

Users can view an image of any letter that is sent to a taxpayer if you set up the following:

- Plug-in an **Online Letter Image** construction algorithm on the *Installation Record*. Refer to *dataDictionary?type=algtype&name=C1-LT-DISP* for an example of such an algorithm. Note, if your letter print software is not capable of producing a PDF containing an image of a letter, users will not be able to view images of letters.
- Plug-in the appropriate extract algorithm on each *letter template*. Algorithms of this type format the records that contain the taxpayer information that is passed to your printing software.

When you plug-in these algorithms, a button appears on *Customer Contact - Main* for customer contacts whose customer contact type references a letter template. When a user clicks this button, the following takes place:

- The installation record's **Online Letter Image** construction algorithm is executed.
- This algorithm calls the customer contact's letter template's extract algorithm. This algorithm constructs the information that's merged onto the letter and returns it to the **Online Letter Image** algorithm. This algorithm, in turn, passes it to your letter print software.
- Your letter print software renders an image of the letter in a PDF and returns it to the **Online Letter Image** algorithm.
- And finally, the **Online Letter Image** algorithm displays the PDF in a separate Adobe session.

Letter Extract Record Structure

This section describes the structure of the record layouts provided with base product flat file letter extract algorithms. All algorithms follow a common pattern and each specific letter extract algorithm includes the details specific for its letter.

The following points highlight the common structure:

- The first 4 bytes are called the **Record Type**. This field is used to identify the type of record.

- The next 26 bytes are called the **Sort Key**. This field is used to ensure the various records used to construct a printed letter appear in the correct order in the interface file.
- The next 12 bytes are called the **Mailing Key**.
- The remaining bytes contain the **Letter Information** that appears on the printed letter. The type of information differs for each record type.

The topics in this section describe each component.

Common Record Types

The topics in this section describe the record structure and values for the common record types. Common record types appear on all letters. There are four types:

- Global Information Record
- Letter Record
- Mailing Information Record
- End of Letter Record

Record Type

The following table lists the common record type values.

NOTE: Specific types of letters will have their own record types. Refer to *Specific Record Types* for letter extracts provided by the base product.

Record Type	***Description***	***Notes***
0010	Global Information Record	Required. 1 per flat file.
1000	Letter record	Required. 1 per letter.
1100	Mailing Information	Required. 1 per letter.
Note	Additional record types are defined for specific letters. Refer to *Specific Record Types* for details.	
9990	End of Letter Record	Required. 1 per letter.

Sort Key

The following table describes the structure of the sort key and the values for the common record types.

Field Name	***Format***	***Source / Value / Description***
LTR_TMPL_CD	A12	
CC_ID	A10	
CC Record Group	A2	This field forces the letter’s records into the correct order. Values: 10 – General Letter information 20 – Mailing information Note: Specific letter record types define their own CC Record Group. Refer to *Specific Record Types* for examples.

		99 – End Letter
Sub Record Group	A2	Available for specific letter record types to further control the sort of records. Refer to *Specific Record Types* for examples.

Mailing Key

Mailing key contains the postal code.

Letter Information

The following shows the letter information for the common record types together with their associated record type, sort and mailing key.

Global Information Record

This is a “header” record that contains information about the extract run as a whole.

NOTE: Please be aware that if you do not sort your extract file by Sort Key after it is produced, this record will physically be the last record on the extract file. Also note that this record will only be produced by the various letter extract processes if you supply the appropriate parameter value (CNTL-REC-SW=Y).

Field Name	***Format***	***Source / Value / Description***
Record Type	A4	"0010"
Sort Key	A26	This is blank for this record type only.
Mailing Key	A15	This is blank for this record type only.
BATCH_CD	A8	CI_BATCH_INST / The Batch Process that called the Letter Extract COBOL routine
BATCH_NBR	N10	CI_BATCH_INST / A sequence number for serial instances of a Batch Process over time
BATCH_RERUN_NBR	N10	CI_BATCH_INST / The number of times a particular batch has been re-run since its initial run
Letter Count	N10	Total number of letters in this extract
Date/Time	A26	System time of extraction.
CC Id Start	A10	The lowest CC ID in a particular thread in the extract
CC Id End	A10	The highest CC ID in a particular thread in the extract
BATCH_THREAD_NBR	N10	CI_BATCH_THD / Identifies a thread distinctly within a group of threads
Thread Count	N10	Total number of threads in the extract run

Letter Record

One record is produced for every letter to be printed.

Field Name	***Format***	***Source / Value / Description***

Record Type	A4	"1000"
Sort Key	A26	
Mailing Key	A15	
CC_ID	A10	CI_CC / System-assigned identifier for a Customer Contact.
PER_ID	A10	CI_CC / System-assigned identifier for the Person who will receive the letter.
CC_DT	D10	CI_CC / The date of the Customer Contact.
CC_TM	T15	CI_CC / The time of the Customer Contact.
CC_CL_CD	A4	CI_CC / The category of contact.
CC_TYPE_CD	A12	CI_CC / The type of contact within the category.
LTR_TMPL_CD	A12	CI_CC / The template of the letter.
LTR_DT	D10	From the current system date/time - the date to print on the letter.

Mailing Information Record

One record is produced for every letter to be printed.

NOTE: Please note that there may be space and display limitations in the designed letter with respect to field lengths. For example, the system supports 64 bytes for the taxpayer name and address and supports several lines of each. The designed letter may not allow for this much information and could cut off information. Please work closely with the letter template designers for your letters to ensure proper display of data.

Field Name	***Format***	***Source / Value / Description***
Record Type	A4	"1100"
Sort Key	A26	
Mailing Key	A15	
ENTITY_NAME1	A64	If an override mailing name has been specified on the CI_PER row, this field contains the first line of the person’s override mailing name (OVRD_MAIL_NAME1). Otherwise, the person’s primary name is used (ENTITY_NAME).
ENTITY_NAME2	A64	If an override mailing name has been specified on the CI_PER row, this field contains the 2nd line of the person’s override mailing name (OVRD_MAIL_NAME2). Otherwise, this field is blank.
ENTITY_NAME3	A64	If an override mailing name has been specified on the CI_PER row, this field contains the 3rd line of the person’s override mailing name (OVRD_MAIL_NAME3). Otherwise, this field is blank.

ADDRESS_SBR		Address Constituents. Total number of bytes = 361. Refer to How The Address is Determined below for the source of the address information.
COUNTRY	A3	
ADDRESS1	A64	
ADDRESS2	A64	
ADDRESS3	A64	
ADDRESS4	A64	
CITY	A30	
NUM1	A6	
NUM2	A4	
HOUSE_TYPE	A2	
COUNTY	A30	
STATE	A6	
POSTAL	A12	
GEO_CODE	A11	
IN_CITY_LIMIT	A1	

End of Letter Record

Field Name	*Format*	*Source / Value / Description*
Record Type	A4	"9990"
Sort Key	A26	
Mailing Key	A15	

Specific Record Types

This section describes the functionality of specific letters provided by the base product.

Collection Letter

The product supplies a base collection letter extract algorithm *C1-LTR-CLLDC*. It includes the common record types as described above as well as additional records with new record types. These records are described below.

NOTE: Please note that there may be space and display limitations in the designed letter with respect to field lengths. For example, several control table descriptions are included and are 60 bytes. The designed letter may not allow for 60 bytes for a description and could cut off some information. Please work closely with the letter template designers for your letters.

Record Types

The following table lists the record types used for the collection letter.

Record Type	Description	Notes
1600	Overdue Process record	Required. 1 per letter.
Note	The subsequent record types depend on the 'collecting on' object for the overdue process. The algorithm supports 'collecting on' object of obligation or assessment. If it's obligation, then record types 1620 and 1622 are populated. If it's assessment, then record types 1650 and 1652 are populated.	
1620	Overdue Process Obligation record	A record will exist for each obligation linked as a 'collecting on' object to the overdue process.
1622	Overdue Process Obligation Debt Category record	A record will exist for each combination of obligation and debt category for outstanding debt.
1650	Overdue Process Assessment record	A record will exist for each assessment linked as a 'collecting on' object to the overdue process.
1652	Overdue Process Assessment Debt Category record	A record will exist for each combination of assessment and debt category for outstanding debt.

Sort Key

The following table describes the structure of the sort key and the values for the collection letter record types.

Field Name	Format	Source / Value / Description
LTR_TMPL_CD	A12	
CC_ID	A10	
CC Record Group	A2	30 - Collection type letters
Sub Record Group	A2	This field forces the subordinate records into the correct order. Values: 10 – Overdue Process record 20 – Overdue Process Obligation record 30 – Overdue Process Obligation Debt Category record 40 – Overdue Process Assessment record 50 – Overdue Process Assessment Debt Category record

Mailing Key

Mailing key contains the postal code.

Letter Information

The following shows the letter information for the collection letter record types together with their associated record type, sort and mailing key.

Overdue Process Record

This record contains information about the overdue process and event that triggered the letter.

Field Name	*Format*	*Source / Value / Description*
Record Type	A4	"1600"
Sort Key	A26	
Mailing Key	A15	
OD_PROC_ID	A12	CI_OD_PROC_LOG / Overdue process that triggered the letter.
OD_EVT_SEQ	N3	CI_OD_PROC_LOG / Overdue event that triggered the letter.
ACCT_ID	A10	CI_OD_PROC / Account being collected on.
OD_PROC_TEMPL_CD	A12	CI_OD_PROC / Overdue process template.
DESCR	A60	CI_OD_PROC_TMP / Description of the overdue process template (using the language of the customer contact's Person).
CRE_DTTM	DT25	CI_OD_PROC / Date/time that the overdue process was created.
CURRENCY_CD	C3	CI_OD_PROC / Currency for the amount.
CHAR_TYPE_CD	C8	CI_OD_PROC_TMP/Char Type of Objects Collecting On

Overdue Process Obligation Record

If the Overdue Process is 'Collecting On' Obligations, this record contains the obligation details. There will be 1 record for each obligation in the collection.

Field Name	*Format*	*Source / Value / Description*
Record Type	A4	"1620"
Sort Key	A26	
Mailing Key	A15	
SA_ID	A10	CI_SA / Id of the Obligation
START_DT	N10	CI_SA / Start Date of the Obligation
END_DT	N10	CI_SA / End Date of the Obligation
DESCR	A60	CI_SA_TYPE_L / Description of the Obligation Type (using the language of the customer contact's Person)

Overdue Process Obligation Debt Category Record

If the Overdue Process is 'Collecting On' Obligations, this record contains the breakdown of the credit allocation by debt category for the given obligation. This is done by calling the "determine balance details" business service (BS). A record exists for every debt category for the obligation's FTs that have outstanding debt.

NOTE: Please note that extract produces a row for each unique debt category that has a balance for each obligation. Your implementation may define granular debt categories, if desired. Please work closely with the letter template designers for your letters to ensure that the display of the debt details allows for the number of possible debt categories for your obligations.

Field Name	***Format***	***Source / Value / Description***
Record Type	A4	"1622"
Sort Key	A26	
Mailing Key	A15	
SA_ID	A10	CI_SA / Id of the Obligation
DEBT_CAT_CD	A8	From Determine Balance Details BS / Debt category
DESCR	A60	From Determine Balance Details BS / Description of Debt Category. (using the language of the Customer Contact's Person)
C1_DEBIT_AMT	N13.2	From Determine Balance Details BS / Assessed Amount
C1-PAY_AMT	N13.2	From Determine Balance Details BS / Payment Amount
C1_WV_AMT	N13.2	From Determine Balance Details BS / Waive Amount
C1_WO_AMT	N13.2	From Determine Balance Details BS / Write Off Amount
C1_OC_AMT	N13.2	From Determine Balance Details BS / Other Credit Amount
C1_BAL_AMT	N13.2	From Determine Balance Details BS / Balance Amount
DEBT_TYPE_FLG	A4	From Determine Balance Details BS / Debt Type

Overdue Process Assessment Record

If the Overdue Process is 'Collecting On' Assessments, this record contains the assessment details. There will be 1 record for each assessment in the collection.

Field Name	***Format***	***Source / Value / Description***
Record Type	A4	"1650"
Sort Key	A26	
Mailing Key	A15	
FT_ID	A12	CI_FT / FT Id of the Assessment

FREEZE_DTTM	N26	CI_FT / Freeze Date Time of Assessment
SA_ID	A10	CI_SA / Id of the Assessment's obligation
START_DT	N10	CI_SA / Start Date of the Assessment's obligation
END_DT	N10	CI_SA / End Date of the Assessment's obligation
DESCR	A60	CI_SA_TYPE_L / Description of the Assessment's Obligation Type (using the language of the customer contact's Person)

Overdue Process Obligation Debt Category Record

If the Overdue Process is 'Collecting On' Assessments, this record contains the breakdown of the credit allocation by debt category for the given assessment. This is done by calling the "determine balance details" business service (BS) for the assessment's obligation and then retrieving the detail specific for this assessment. A record exists for every debt category for the assessment's related FTs that have outstanding debt.

NOTE: Please note that extract produces a row for each unique debt category that has a balance for each assessment. Your implementation may define granular debt categories, if desired. Please work closely with the letter template designers for your letters to ensure that the display of the debt details allows for the number of possible debt categories for your assessments.

Field Name	***Format***	***Source / Value / Description***
Record Type	A4	"1652"
Sort Key	A26	
Mailing Key	A15	
FT_ID	A12	CI_FT / Id of the Assessment
DEBT_CAT_CD	A8	From Determine Balance Details BS / Debt category
DESCR	A60	From Determine Balance Details BS / Description of Debt Category. (using the language of the Customer Contact's Person)
C1_DEBIT_AMT	N13.2	From Determine Balance Details BS / Assessed Amount
C1-PAY_AMT	N13.2	From Determine Balance Details BS / Payment Amount
C1_WV_AMT	N13.2	From Determine Balance Details BS / Waive Amount
C1_WO_AMT	N13.2	From Determine Balance Details BS / Write Off Amount
C1_OC_AMT	N13.2	From Determine Balance Details BS / Other Credit Amount
C1_BAL_AMT	N13.2	From Determine Balance Details BS / Balance Amount

DEBT_TYPE_FLG	A4	From Determine Balance Details BS / Debt Type

How the Address is Determined

The address is retrieved as described in *Determining Address*.

NOTE: If an address is not found, the letter is not produced and a warning is issued by the background process responsible for creating the letter extract file.

Suppressing Letters

The following points highlight the “lifecycle” of a customer contact letter:

- **Not yet extracted**. When a customer contact is first created and the customer contact type refers to a letter template, the extract batch control is stamped onto the customer contact along with the current run number of the batch control. Note that internally, the customer contact also has a “print letter” switch that is set to “Y”.
- **Extracted**. When the extract batch process runs, the appropriate information related to the customer contact is sent to the printing software. The customer contact is updated with the extract date / time. At this point, the application does not have any additional information about the status of the printing of the letter as it is in the control of the printing software.

There are some implementations that may want to attempt to suppress a letter from printing if the process that created the customer contact is canceled for some reason. If the letter has not yet been extracted, then it is possible to suppress the letter from being printed. Once the letter has been extracted, it is possible to still prevent mailing of the letter by either stopping the print via the printing software or by intercepting the printed letters before mailing. However, the application does not have any control over intercepting letters after they are extracted.

The product provides the following support for attempting to suppress the letter via an algorithm. Currently there is no support for suppressing the letter using the customer contact page.

- Updating the “print letter” switch. The extract batch process only selects customer contacts where the “print letter” switch is set to “Y”. If an algorithm would like to suppress a letter, it should mark this switch as “N”. Note that if a letter has already been extracted, the switch may still be set to “N” if desired by the implementation. This may be done for audit purposes to indicate that the process attempted to suppress the letter but that it was already extracted.
- Suppression Reason. The product supplies a customer contact characteristic type that may be used to mark a reason for suppression. Algorithms that attempt to suppress the letter should populate the suppression reason characteristic with an appropriate value.
- The customer contact user interface displays letter information and will adjust the information based on whether the print letter switch is Y or N and whether or not the letter has already been extracted.

NOTE: For an example of an algorithm that attempts to suppress a letter, refer to the overdue type reversal algorithm **C1-CC-REVRSL** Reverse Customer Contact

Setting Up Letter Templates

You can set up a customer contact type to generate a form letter whenever a customer contact of this type is added. In fact, this is the only way to generate a letter in the system.

FASTPATH:
Refer to *Letter Printing* for more information about how letters are produced.

Every customer contact that causes a letter to be sent must reference a unique letter template. To define a letter template, open **Admin Menu** > **Letter Template**.

Description of Page

The following fields are required for each letter template:

- **Letter Template** is the unique identifier of the letter template.
- Use **Description** to enter a brief description of the letter.
- Use a **Customer Contact Business Object** to define a *BO* that may govern additional rules related to customer contacts of this type.
- Turn on **Special Extract** if this type of letter should only be created via a system generated event such as a collection letter. Turning on this switch is what prevents a user from adding a customer contact that references this type of letter template (because you don't want a user to be able to request a letter associated with a system generate event by adding a customer contact, rather, they must execute the appropriate process and it will generate the customer contact).
- The next two fields control how letters of this type are printed (both in batch and online). Refer to *Technical Implementation Of Batch Letter Production* for more information about producing letters in batch. Refer to *Technical Implementation Of Online Letter Production* for more information about online letter production.
  - Use **Batch Control** to define the process that creates the flat file that is passed to your letter printing software. If you use an **Extract Algorithm** to construct the downloaded information, you can use the **LTRPRT** process.
  - Use **Extract Algorithm** to define the plug-in component that constructs the "flat file records" that contain the information to be merged onto letters of this type. This algorithm is called when a user requests an online image of a letter on *Customer Contact - Main* and it may also be called by the batch letter extraction process defined above. Click *here* to see the algorithm types available for this plug-in spot.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_LETTER_TMPL*.

Setting Up Customer Contact Classes

Every customer contact record has a contact type that classifies the record for reporting purposes. And every contact type, in turn, references a customer contact "class". The class categorizes customer contacts into larger groupings for reporting purposes.

Open **Admin Menu** > **Customer Contact Class** to define your customer contact classes.

Description of Page

Enter a unique **Contact Class** and **Description** for each customer contact class.

After you have created your customer contact classes, you'll be ready to setup your *customer contact types*.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_CC_CL*.

Setting Up Customer Contact Types

Every customer contact record has a contact type that controls the behavior of the customer contact.

FASTPATH:

Refer to *The Big Picture Of Customer Contacts* for more information about customer contacts.

Open **Admin Menu** > **Customer Contact Type** to define your customer contact types.

Description of Page

Every customer contact type is identified by a unique combination of **Contact Class** and **Contact Type**.

Enter a brief **Description** of the customer contact type.

Only specify a **Contact Shorthand** if customer contacts of this type can be added in the *Customer Contact Zone*. The value you specify in this field is what the user selects to add a customer contact in this zone.

Use **Contact Action** if something should be triggered when customer contacts of this type are added. The only valid value in this release is **Send Letter**. If you select this option, you must also specify a **Letter Template**. Refer to *Letter Printing* for more information about how letters are produced.

Use the **Customer Contact Type Characteristics** collection to define characteristics that can be defined for contacts 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 customer contacts of a given type. Turn on the **Default** switch to default the **Characteristic Type** when customer contacts 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.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_CC_TYPE*.

Setting Up Letter Suppression Reasons

If your implementation includes algorithms that support attempting to suppress letters from printing, appropriate letter suppression reasons should be configured.

- Go to Admin > Characteristic Type and select the characteristic type **C1-REVST** , Letter Suppression Reason.
- Add appropriate values to the Characteristic Value collection as needed.

Chapter 5

Defining Pay Plan Options

Tax authorities use pay plans as a collections tool. When a taxpayer agrees to be on a payment plan, it is an acknowledgement of his/her tax debt. Pay plans are typically used to satisfy obligations in which the taxpayer cannot satisfy the total obligation with one lump payment.

The pay plan's terms are typically negotiated between the tax authority and taxpayer. The payment amount and specific date intervals of the pay plan include such factors as total obligation amount and the taxpayer's ability to pay. Payments received are used to satisfy tax, penalty, and interest.

The Big Picture of Pay Plans

A pay plan is a special type of payment plan that encompasses two major elements:

- A set of scheduled payments
- The business rules used to recommend the payment schedule

Pay plans are managed via an obligation whose obligation type has a special role of **Pay Plan**. This type of obligations must have one or more recommendation rules, and are allowed to have payment schedules.

A pay plan's status is just like any other obligation's status. An active pay plan implicitly has a "kept" status (i.e. all scheduled payments have been made). A stopped plan implies that the pay plan has been completed, i.e. paid off, or stopped due to non-payment. It is important to note that the scheduled payments have their own status.

A list of the obligations covered by a pay plan is maintained with the pay plan. This list is used at payment distribution to determine which obligation to pay. An obligation may be covered by a pay plan when its obligation type is flagged as **Eligible for Pay Plan**.

Creating Pay Plans

A payment plan can be created as a result of the following scenarios:

- A collections case is created as a result of overdue debt. One of the available actions for the user responsible for the collections case is setting up a pay plan. Refer to *The Big Picture of Collection Cases* for details on how a collection case can create a pay plan.
- A bankruptcy case seeking readjustment of debt may require the taxpayer (debtor) to fulfill an agreed-upon payment plan. For bankruptcy types that require a pay plan, one of the available actions for the user is to create a pay plan. Refer to *The Big Picture of Bankruptcy* for details on bankruptcy and pay plan creation.
- The taxpayer may also voluntarily seek to enter a payment plan. Refer to *Maintaining Pay Plans* for details on how to create a pay plan manually.

Activating Pay Plans

A pay plan is created in a **pending start** status. For the pay plan to become effective, it needs to be activated. If the pay plan does not require any special approval, the user can activate the pay plan once they have completed setting up the payment schedule. This can be done by navigating to the obligation page, and using the **activate** button.

If the pay plan requires approval, a different process might be implemented depending on the tax authority's rules.

When a pay plan is activated, you can perform special processing using an algorithm plugged in on the obligation Activation system event on the pay plan obligation 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 taxpayer of their payment amount and payment schedule.
- Initiate the creation of a payment coupon book for a taxpayer.
- In some cases, the taxpayer may be charged a setup fee for the pay plan. See the base algorithm (*C1-OT-LEVFEE*) for an example.

The system comes supplied with a sample algorithm type (*SAAT-CC*) that simply creates a customer contact to indicate that the pay plan is activated.

Breaking Pay Plans

If the taxpayer does not meet their terms of the pay plan, the pay plan is considered broken. A pay plan can be broken in the one of the following ways:

- The user has used the **Break** action on the pay plan page.
- An automated process, such as monitor algorithm has determined that the taxpayer has broken the terms of the pay plan.

When the pay plan is broken, the pay plan is transitioned to **pending stop** status.

A base package break pay plan algorithm type (*C1-PAYP-BRK*) can be used to create a characteristic value to indicate if a pay plan is manually stopped by a user.

In addition, the algorithm type (*C1-OT-TPPRVW*) will create an account monitor review trigger if the pay plan is not associated with a collections case.

You can plug in an algorithm (of type *SAIS-ST*) on the obligation Stop Initiation system event on the pay plan obligation type to automatically stop the obligation (i.e. transition it to **stopped** status).

To finalize a pending stop obligation, the system calls the stop obligation algorithm plugged-in on the obligation Stop system event on the obligation type.

Stopping Pay Plans

If the taxpayer pays off the outstanding debt, the pay plan is considered stopped. A pay plan can be stopped in one of the following ways:

- The collection case that created the pay plan was closed as a result of the taxpayer paying off the debt, and updates the pay plan to **pending stop**.
- An automated process, such as monitor algorithm has determined that the taxpayer has paid off the debt, and updates the pay plan status to **pending stop**.

As indicated above, an algorithm can be plugged it to transition the pay plan immediately from **pending stop** to **stopped**.

Canceling Pay Plans

A pay plan may need to be cancelled under certain circumstances that are initiated either by the tax authority or the taxpayer. For example:

- A user may need to cancel an erroneous pay plan after it has already been activated.
- A tax authority may cancel pay plans (en masse) in the event of natural disasters.
- When a taxpayer indicates additional hardship, the procedure may involve canceling the existing pay plan and renegotiating a new pay plan.

Canceling a pay plan causes the pay plan status to change to **Cancelled**. Any specific actions that need to take place can be plugged in on obligation Cancel system event. The base package algorithm type (*C1-OT-CRCC*) can be used to create a customer contact when the pay plan is cancelled. Alternatively, (*SACA-CRTODO*) can be used to create a to do entry when the pay plan is cancelled.

Distributing Payments for Pay Plans

Pay plan payments are distributed directly to the covered obligations using payment event distribution rules.

The base-package provides a **Pay A Plan** Distribution Rule — Create Payment algorithm type *C1-DR-PAYPP*. This algorithm assumes that the payment event has a distribution detail that specifies the Pay Plan ID.

To enable this algorithm type:

- Configure an algorithm for C1–DR-PAYPP with the appropriate parameters.
- Create a distribution rule for Pay Plan ID.

  Plug in the algorithm you created in this distribution rule.

Refer to *Payment Event Distribution Rules* for more information on configuring distribution rules.

Processing Scheduled Payments

In addition to the pay plan having a status, the scheduled payments each have their own status. A scheduled payment is created in a **pending** status when the pay plan is activated.

Each scheduled payment is reviewed by the base base-package batch **C1-PAYPS** on the payment due date. The batch will set the scheduled payment status to **processed** after executing the **Process Pay Plan Scheduled Payment** plug-in defined on the obligation type.

This plug-in can contain specific business logic that need to be executed when a scheduled payment is due. Based off the rules that will be configured by the tax authority, a multitude of actions can occur. They range from giving the taxpayer a few extra grace days to make the payment to canceling the current payment plan and forwarding the obligations to a collections authority. A tax authority's business rules as well as each taxpayer's personal history and circumstances should be taken into account when determining what actions to take next.

The base-package provides an algorithm (*C1-CC-CHKSPP*) that will create a to do if a scheduled payment has not been paid. It will also detect when actual payments have fulfilled the pay plan in advance.

If your implementation's rules require additional status values (e.g. kept, late, partial), you will need to do the following:

- Create your own copy of **C1-PAYPS**.
- Add custom values to lookup **NB_SCHED_PAY_STATUS_FLG**.

Automatic Payment and Pay Plans

If a taxpayer wants to pay their pay plan scheduled payments automatically, the account must be set up for automatic payment. In addition, the pay plan must indicate that automatic payment is being used.

FASTPATH:
Refer to *How To Set Up Automatic Payment For A Pay Plan* for more information.

When this is done, a background process referred to as **C1-PAYPA** creates automatic payments on the scheduled payment date by calling the automatic payment creation algorithm plugged in on the installation record.

The base-package provides two auto pay creation algorithms that support pay plans scheduled payments:

- Use algorithm type *APAY-CREATE* if your implementation distributes payments using the standard account type payment distribution.
- Use algorithm type *C1-APAY-CRDR* if your implementation distributes payments using distribution rules.

Alerts For Pay Plans

The system provides *alerts* to highlight the existence of pay plans. These alerts are important to assist the tax authority's users:

- An alert is displayed if the account has a pay plan that is not stopped (e.g. **pending start**, **active** or **pending stop**).
- For taxpayers who are permanently forbidden from having a pay plan, the user should put a permanent *alert on the account*.
- Use an algorithm to highlight cancelled pay plans with an entry in the alert zone. The algorithm type to do this is not provided. Use *C1-STOP-SA*as an example of how to create this type of algorithm.
- Configuring a specific *customer contact type* with the alert algorithm type (*CC BY TYPCL*) can cause alerts to be displayed whenever a user adds a *customer contact* of this type. This method can be used to highlight special conditions relating to the pay plan.

FASTPATH:
For more information about introducing alert conditions in the alert zone, refer to *Installation Options - Algorithms*.

Setting Up Pay Plan Options

The topics in this section describe functionality that you must consider when designing pay plans.

Designing Recommendation Rules

This section describes topics related to designing your recommendation rules.

What is a Pay Plan Recommendation Rule

Users ask the system to recommend the scheduled payments for a pay plan. In general, this recommendation process must establish:

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

A recommendation rule comprises of three elements:

- Payment schedule algorithms
- An algorithm to calculate an average daily amount. This is optional. If this plug-in must be used, it should be invoked from your payment schedule algorithms
- A collection of default parameter values for the payment schedule algorithm type

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 pay plans already in effect. Existing pay plans keep the parameter values that were used when the pay plan 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).

NOTE:
Normally parameter values for an algorithm type are kept with the algorithm. Because the parameters may vary for each pay plan, while the same algorithm logic is used, the parameter values are kept with the pay plan.

Examples of Recommendation Rules

The following are two common methods of calculating a schedule of payments:

- Taxpayer knows how much and when they can pay. The recommendation rule will calculate the schedule of payment dates given the fixed payment amount, frequency, and the first payment date. For example: monthly payments of $1,000 beginning on August 1, 2007.
- Taxpayer knows when they would like to pay the debt by, and how often they can pay. The recommendation rule will calculate the schedule of payment dates and payment amounts given the first payment date, the last payment date and the frequency. For example: monthly payments to be made starting on August 1, 2007 and ending on February 1, 2008.

Generate Payment Schedule Algorithm Types

The core of the pay plan recommendation rule is the Generate Payment Schedule plug-in. The base package provides two general methods for calculating a payment schedule: fixed amount and fixed duration.

One set of base-package algorithms assume that P&I controls are applicable to the pay plan’s covered obligations. These algorithms include penalty and interest charges in the calculated scheduled payments:

- Schedule Fixed-Amount Payments With P&I Forecasting: *C1-PP-FIXAMT*
- Schedule Fixed-Duration Payments With P&I Forecasting: *C1-PP-FIXDUR*

Another set of base-package algorithms assume that the covered obligations do not have P&I controls. These algorithms calculate the scheduled payments based on simple obligation balances:

- Schedule Fixed-Amount Payments Without P&I Forecasting: *C1-NOPI-FAMT*
- Schedule Fixed-Duration Payments Without P&I Forecasting: *C1-NOPI-FDUR*

Penalty and Interest Considerations

If *penalty and interest* (P&I) rules are applicable to the obligations covered by a pay plan, the payment schedule that is generated by the recommendation rule's generate payment schedule algorithm should factor expected penalty and interest calculations.

The penalty and interest plug-in spot may be called in *forecast* mode to forecast the charges to a certain date in the future. The plug-in spot also supports receiving financial transactions supplied by a calling service allowing for "what if" scenarios to be supplied. For example, when forecasting P&I to the future, the generate payment schedule algorithm can include proposed scheduled payments along with existing financial transaction to the P&I algorithm to get a more accurate estimate of the future P&I charges.

Setting Up The System To Enable Pay Plans

The above topics provided background information about how pay plans are supported in the system. The topics in this section describe how to set up the system to enable pay plan functionality.

Characteristic Type

If your implementation requires a characteristic added to the pay plan when the pay plan is broken, you will need to create a characteristic type that specifies **obligation** as its characteristic entity.

Customer Contact Class And Types

If your implementation requires certain customer contacts and/or letters created when the pay plan is activated or stopped, you will need to create appropriate customer contact class and customer contact type(s).

NOTE:
If you want to send letters to your taxpayers when a contact of any of these types is created, you must create an appropriate *letter template* and attach it to the customer contact type.

Algorithms

As described above, in *The Big Picture of Pay Plans*, the base package provides a number of algorithm types for different system events in the pay plan's lifecycle. In order to use any one of these algorithm types, you will need to create an associated algorithm and ensure any required parameters are populated.

If your implementation created additional custom algorithm types, you will also need to define associated algorithms.

Defining a Pay Plan Recommendation Rule

Recommendation rules are used to recommend scheduled payments for pay plans. To define recommendation rules, navigate to **Admin Menu, Pay Plan Recommendation Rule.**

Description of Page

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

If applicable, 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 pay plans that use this recommendation rule. The Payment Schedule Algorithm Type cannot be modified if a pay plan 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.
- **Pay Plan Rule ParameterValue** 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*.

Obligation Types

This section provides information about setting up the obligation types for the pay plan itself and the obligation types for covered obligations.

Obligation Type for Obligations Covered by Pay Plans

Consider which types of obligations need to be covered by a pay plan. For each obligation type that should be allowed to be covered by a pay plan, ensure that the **Eligible for Pay Plan** flag (on the Billing page) to **Eligible**.

Pay Plan Obligation Type

In order to enable pay plan functionality, you will need to define an obligation type that is suitable for a pay plan. Depending on the business rules of the tax authority, you may need multiple pay plan types, which will require you to define multiple obligation types. For example you might want to create a pay plan type for individual taxpayers, and a different pay plan type for a business.

The following points provide guidelines for creating a pay plan obligation type:

Obligation Type - Pay Plan Recommendation Rule

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

Obligation Type - Main

- **Tax Type** should either reference a specific tax type or a generic tax type (e.g. miscellaneous fees) depending on your authority's business rules.
- **Revenue Class** should be set to **N/A**. (Revenue classes are not applicable because pay plans do not apply a rate and revenue classes are only relevant for obligation types that use a rate.)
- The **Payment Segment Type** should reference the **Normal Payment**.
- **Do Not Overpay** should be on. Payments are not distributed directly against the pay plan.

Obligation Type - Detail

- **Special Role** is **Pay Plan**.
- **Renewal** may be optional, not allowed, or required depending on your business processes.
- If Renewal is required, specify the **Days Before Expiration for Renewal**.

Obligation Type - Billing

- **Eligible for Billing** flag should be off, as pay plans do not get billed.
- **Characteristic Location Required** should not be checked for pay plans.
- **Eligible for Pay Plan** should be set to **Ineligible**. This option only applies to the obligation types that can be covered by a pay plan.

Obligation Type - Rate

Pay plans do not use rates, so the **Rate Required** flag should be off.

Obligation Type - Algorithms

Plug in any algorithms defined above for the following system events:

- Break Pay Plan
- Initiate Stop
- Obligation Activation
- Obligation Cancel
- Obligation Renewal
- Obligation Stop
- Process Pay Plan Scheduled Payment

Pay Plan Background Processes

Ensure that the following background processes are scheduled:

- Pay Plan Scheduled Payment Processing (**C1-PAYPS**)

- Pay Plan Scheduled Payment Automatic Payment Create (**C1-PAYPA**)

Chapter 6

Defining Address Options

This chapter describes options related to addresses.

Centralized Address

The product supports a centralized address repository. All address records are stored in a common table and may be linked to persons, tax roles and assets. Refer to the documentation for each object for more information about how addresses are linked.

NOTE: Legacy address functionality. Implementations that existed prior to the introduction of the centralized address functionality may still be using the address functionality that was originally provided in the product. This is referred to as "legacy" address functionality. Refer to *legacy address* for more information about legacy address functionality.

The following points highlight some information about centralized address functionality:

- The address object displays and captures address constituents based on the configuration on the *country* defined for the address. Your implementation must define the valid countries and the expected address constituents to capture for that country.

  NOTE: The address object does include support for the elements House Type or In City Limit.

- The address object does not have a related Type object that governs the behavior of the address as is found with many other master and transaction objects in the system.
- There is an Address Type extendable lookup supplied. This is used to define how / why an address is linked to another object. For example, a person may have a "legal" address and a "mailing" address. In order to link addresses to a person or a tax role, the valid address types must be configured on the respective person types and tax types.
- The address object does not have a configurable business object lifecycle. Rather it has a simple Active / Inactive flag. When changing an address to be inactive, the system prompts for an inactive reason, which are defined using the Address Inactive Reason extendable lookup

- The system supplies a Find address algorithm plug-in spot on the *installation record*. If your implementation receives information from an external source related to a change to a taxpayer’s address, the plug-in spot may be used to determine if the address is already in the centralized address repository.

Legacy Address

The product was originally built with address functionality that is referred to now as "legacy" address functionality. This section describes topics related to that functionality.

Address Feature Configuration Setting

Implementations must opt to use either the centralized address functionality or the legacy address functionality. Because some functionality in the system only applies to one address option or another, the product uses a feature configuration option define which address functionality is being used.

If your implementation is using the legacy address functionality, you must also set up a *Feature Configuration* option to indicate that. Find the Feature Configuration record for the **General System Configuration** feature type. (It may need to be defined if it does not exist). Choose the option type **Address Support** and define a value of **LEGACY**.

For implementations that use the centralized address functionality may choose to create the above option with a value of **CENTRALIZED** to be explicit. However, if the option is not populated, the product assumes that centralized address functionality is supported.

Setting Up Location Options

This section describes tables that must be set up before you can define locations.

Defining Location Types

NOTE: This is legacy address functionality and does not apply to implementations that use a centralized address.

Open **Admin Menu** > **Location Type** to define the location types used to categorize your locations.

Description of Page

Enter a unique **Location Type** and a **Description** for every location type.

Use a **Location Business Object** to define a *BO* that may govern additional rules related to locations of this type.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_PREM_TYPE*.

Defining Location Information

NOTE: This is legacy address functionality and does not apply to implementations that use a centralized address.

The location information displayed throughout the system is controlled by a plug-in.

The system first looks to see if the location type references a location business object and if the business object defines an information plug-in. If a BO is not provided or if that BO does not define an information algorithm, the system looks for an information algorithm plugged into the location maintenance object.

If no plug-ins are found on the BO or the MO, the system looks for a Location Information plug-in algorithm on the *installation record*.

If you prefer different formatting logic, your implementation should provide a plug-in at one of the above plug-in spots, as appropriate for your business rules.

Setting Up Postal Defaults

Set up postal defaults if field values can be defaulted onto new addresses based on an address' postal code.

In addition, for implementations using the Location object that is part of the legacy address functionality, there are some additional fields that may be defaulted from postal defaults. The topics in this section describe how to maintain postal defaults.

Postal Defaults - Main

To define postal defaults, open **Admin Menu** > **Postal Code Default**.

Description of Page

Enter the **Country Code** and range of postal codes to which the default values apply using the **From Postal Code** and **To Postal Code**.

NOTE: You may not have postal defaults whose from / to postal codes overlap.

Enter the **County** to be defaulted onto new addresses located in this postal code range.

Enter the **City** to be defaulted onto new addresses located in this postal code range.

Enter the **Division** to be defaulted onto new locations located in this postal code range. This field is only visible if the *address support* is set to Legacy.

Enter the **State** to be defaulted onto new addresses located in this postal code range.

Enter the **Time Zone** to be defaulted onto new locations located in this postal code range. This field is only visible if the *address support*is set to Legacy.

Use the **Characteristic Types and Values** collection to define the **Characteristic Type**s and their respective **Characteristic Value**s to be defaulted on locations located in this postal code range. This field is only visible if the *address support* is set to Legacy.

Use the **Geographic Types and Values** collection to define the **Geographic Type**s and their respective **Value**s to be defaulted on locations located in this postal code range. This field is only visible if the *address support* is set to Legacy.

Where Used

The County, City and State are defaulted when a new address is added on the centralized address object. In addition, for implementations using legacy address, the same information is defaulted onto new addresses defined on location, person correspondence or account / person override.

For implementations using the location object in the legacy address functionality, characteristics and geographic values are also defaulted when the postal code for a location is changed.

Determining Address

There is some functionality in the system that may require an address for correspondence purposes. For example, when sending a letter or when providing the accounts payable system an address for issuing a check. Implementations may provide their own algorithms for determining the address. The base product provides logic to Retrieve Mailing Address that determines the address as follows:

- For implementations using centralized address functionality, the system looks for an address linked to the Person effective for the processing date where the priority flag is set to 10 (preferred for correspondence). If there are seasonal addresses defined for the current season, that address is used. If the functionality has an obligation or account and doesn't have a specific person, the main person on the account is used.

For implementation is using *legacy address* functionality, the following points describe the source of the address:

- If the account's main person has a *seasonal address* effective on the business date, the address is set to the respective seasonal address.
- Otherwise the address source on the *Account - Person* record for the account's main person is checked.
    - If the address source is **Mailing Location on Account**, the address is set to the *account's mailing location*.
    - If the address source is **Account Override**, the address is set to the override address on the *Account - Person* record.
    - If the address source is **Person**, the address is set to the *person's mailing address*.
    - If the address source is not defined, the *person's mailing address* is used.

Chapter 7

Defining Forms Processing Options

Forms processing is a core function for a tax authority. This section explains the forms definition and processing functionality of the system and describes how to set up the tables that control forms processing.

The Big Picture of Forms Processing

The product supports registration forms and tax forms.

Most forms have a common structure. The form consists of one or more sections, with each section containing one or more lines or groups of lines. Processing rules can be associated with any of these components.

As form changes are introduced from year to year, the changes usually entail adding or removing specific sections and / or lines, as well as adding / removing the processing rules associated with them.

Forms can come from one or more sources. They can be interfaced in batch from an external system or created internally.

Registration Forms

Registration forms are used for taxpayer registration and / or taxpayer demographic information updates. These forms are processed in the context of a taxpayer. When a registration form is processed, it usually results in the creation of new taxpayers / accounts / tax roles / obligations or updates to existing taxpayer information - e.g. name / address change.

A taxpayer is registered in any of the following ways:

- The taxpayer files a paper registration form
- The taxpayer submits a registration form online.
- A tax authority user registers the taxpayer using an online registration form
- For some tax types, the system can automatically register the taxpayer when the first tax form is processed.

Registration Forms Are Linked to Taxpayers

Registration forms create / update taxpayer information. When a registration form posts, it may include actions like adding / updating accounts, tax roles and obligations.

Unlike tax forms, registration forms typically don't result in assessments. Thus, these forms usually do not have a financial effect.

Registration Forms Have A Common Lifecycle

The base product supplies a parent business object for registration forms that defines the following lifecycle:

Pending	The registration form is created in an initial state where processing rules are not yet executed. This allows the form to be saved as work-in-progress.
Validated	A validation step verifies the information on the registration form. Validation form rules specific for the form type are executed to check the validity of the form data. Any issues detected at this step usually causes form processing to stop.
Suspended	A registration form goes into this state when the validation step detects errors on the form that a user may be able to investigate and resolve. Exceptions are stored in the system so that a user can resolve the issues accordingly. The form needs to be re-validated after the issues are resolved.
Waiting for Information	A registration form goes into this state when the validation step detects missing information, thus preventing the system from further proceeding with form processing. Exceptions are stored in the system so that a user can track the issues accordingly. In addition, this state usually triggers correspondence to the taxpayer, requesting for the missing information. The form is usually re-validated after the missing information is received.
Ready for Posting	When a registration form passes validation, it transitions to Ready for Posting. From here a user may choose to make further changes by re-editing the form. Or the form can transition to Posted.
Re-edited	When a user is creating or correcting a form and it passes validation, there may still be a need to review and correct the data further. A form in the Ready for Posting state may be updated by transitioning it to Re-edited. The form needs to be re-validated once it is re-edited.
Posted	When a registration form passes validation, it can transition to Posted. The posting of a form typically causes other data in the system to be added or updated based on the purpose of the registration form. Posting form rules specific to the form type are responsible for the posting logic.
Canceled	Forms that are not yet posted may be canceled if a user determines that the form was created incorrectly.

The base package provides a registration form business object **C1-ParentRegistrationForm** that has the lifecycle described above. Specific form business objects created using form generation are child forms for a common parent form BO so that the lifecycle is consistent for all forms. The base parent BO may be used or an implementation may create a custom business object. Refer to the business object meta-data for more details on the lifecycle, including the allowed transitions and the algorithms provided.

Tax Forms

Tax forms are the most common types of forms. These forms are typically processed within the context of a filing period. When a tax form is filed, it satisfies an obligation to file. Form processing results in an assessment, which determines either a tax amount due or an overpayment. For tax types that do not require pre-registration, the tax form's processing may also result in taxpayer creation.

Tax Forms Are Linked to Obligations

When a tax form is posted, it is associated with an obligation for a specific filing period. Any financial transactions relating to the filing period will be created under this obligation. The obligation can be created in advance in anticipation of filing, such as in the case of business taxes or created at the time the form is posted, such as in the case of individual income tax.

The form type references one or more obligation types, and this information is used when determining what type of obligation to create for a specific tax form. Typically a form type would only reference one obligation type. For example, taxes such as sales and use, individual income, and corporate income would typically have one obligation type defined per form type. Both the **C1–StandardTaxFormType** and the **C1–FilingCalendarTaxFormType** base business objects support the definition of one obligation type on a form type.

There may be situations where a form type would be associated with more than one obligation type. These include business activity statements, estimated payments, audit forms. A different business object can be used to support these situations. Alternatively, the validation on the base business object can be disabled to allow more than one obligation type for a form type.

The obligation types defined for a form type are used to restrict the form type list for a specific tax type, when a user adds a tax form.

Tax Forms Have A Common Lifecycle

The base product supplies a parent business object for tax forms that defines the following lifecycle:

Pending	The tax form is created in an initial state where processing rules are not yet executed. This allows the form to be saved as work-in-progress.
Validated	A validation step verifies the information on the tax form. Validation form rules specific for the form type are executed to check the validity of the form data. Any issues detected at this step usually causes form processing to stop.
Suspended	A tax form goes into this state when the validation step detects errors on the form that a user may be able to investigate and resolve. Exceptions are stored in the system so that a user can resolve the issues accordingly. The form needs to be re-validated after the issues are resolved.
Waiting for Information	A tax form goes into this state when the validation step detects missing information, thus preventing the system from further proceeding with form processing. Exceptions are stored in the system so that a user can track the issues accordingly. In addition, this state usually triggers correspondence to the taxpayer, requesting for the missing information. The form is usually re-validated after the missing information is received.
Ready for Posting	When a tax form passes validation, it transitions to Ready for Posting. From here a user may choose to make further changes by re-editing the form. Or the form can transition to Posted.

	NOTE: For implementations that use form control functionality, the tax form is only progressed to Posted by linking it to a form control that gets approved. Refer to The *Big Picture of Form Control* for more information.
Re-edited	When a user is creating or correcting a form and it passes validation, there may still be a need to review and correct the data further. A form in the Ready for Posting state may be updated by transitioning it to Re-edited. The form needs to be re-validated once it is re-edited.
Posted	When a tax form passes validation, it can be posted in the system. The posting of a form typically causes other data in the system to be added or updated. For example, adjustments are created for the assessments. Taxpayer data may also get updated based on newer information from the form. Posting form rules specific to the form type are responsible for the posting logic.
Adjusted	A posted tax form can be adjusted for any of the following common reasons: • Correcting data capture errors or data entry errors • Line item adjustments initiated by the taxpayer • Adjustments initiated from an audit When a tax form is adjusted, the financial effect of the existing tax form is canceled and new financial transactions are created for the updated line items. Note that the effects of adjusting a form are controlled by form rules.
Transferred	A tax form can be transferred to a different taxpayer/obligation if it previously posted to the incorrect taxpayer/obligation. When a tax form is transferred, the current tax form transitions to Transferred and a new tax form is created, allowing a user to indicate the correct taxpayer / obligation. The financial effect of the existing tax form is removed from the current obligation using form rules. The new form goes through the same lifecycle of validation and posting and new financial transactions are created for the 'transfer to' obligation via the new form's posting rules.
Reversed	Some exceptional cases may require that the financial effect of a tax form be canceled, without necessarily creating new financial transactions. A tax form reversal is a way of voiding a tax form that has already posted.
Canceled	Forms that are not yet posted may be canceled if a user determines that the form was created incorrectly.

The base package provides a tax form business object **C1-ParentTaxForm** that has the lifecycle described above. Specific form business objects created using form generation are child forms for a common parent form BO so that the lifecycle is consistent for all forms. The base parent BO may be used or an implementation may create a custom business object. Refer to the business object meta-data for more details on the lifecycle, including the allowed transitions and the algorithms provided.

Other Changes After a Tax Form Posts

The following sections describe other possible changes after a tax form posts.

Standalone Amendment Forms	A posted tax form can also be adjusted by processing a standalone amendment form. In this case, the amendment form is processed independently of the original form. New financial transactions are created for the difference between the financial effect of the original form and the financial effect of the amendment form.

Standalone Audit Forms	A standalone audit form can also adjust a previously posted tax form. In this case, the audit form is processed independently of the original form. New financial transactions are created for the difference between the financial effect of the original form and the financial effect of the audit form.

Payments Related to Tax Forms

Some tax types require taxpayers to make estimated payments based on specific conditions - e.g. estimated revenue, estimated tax liability, etc. The system allows for payments to be made in advance. These payments are usually reconciled when the associated tax forms are processed.

The system also caters for payments sent together with tax forms. The payments are usually created when the form posts, suspends or cancels (depending on the situation).

Payments in Advance / Estimated Payments

Some tax types require taxpayers to make estimated payments based on specific conditions -e.g. estimated revenue, estimated tax liability, etc.

When the payment is processed, the obligation may or may not exist yet.

- If the obligation exists, the payment is applied to that obligation
- If an obligation does not exist but the taxpayer has an existing account, the payment is applied to an excess credit obligation under that account.
- If the taxpayer does not have any accounts, the payment is applied to a company suspense account.

In the cases where the payment got applied to an excess credit obligation or company suspense, the payment needs to be transferred when the tax form is processed and the obligation is determined. To do this, a form rule needs to be configured to do this transfer at the point where the form's obligation is identified.

FASTPATH:
Refer to *Defining Form Rules* for details on how to configure rules.

It's important for the payment to capture details that will facilitate the transfer of the payment to the correct obligation when the tax form gets processed. The payment should capture details like the taxpayer's identifier and the tax type / filing period being paid.

Payments with a Tax Form

When payments are sent with a tax form, the payment is usually linked to the form by a unique identifier, such as a document locator.

The payment is processed in two possible ways, which are described in the following sections

Payment Is Processed Separately From The Tax Form

This is applicable for implementations that have separate processes for payments and forms.

In this case, the payment is applied depending on which of the payment and the tax from gets processed first:

- If the tax form is processed before the payment, the payment is applied to the obligation on the form.
- If the payment is processed before the tax form and:
  - The taxpayer has an existing account, the payment is applied to an excess credit obligation under that account

- The taxpayer does not have any accounts; the payment is applied to a company suspense account.

In the cases where the payment got applied to an excess credit obligation or company suspense, the payment needs to be transferred when the tax form is processed and the obligation is determined. To do this, a form rule needs to be configured to do this transfer at the point where the form's obligation is identified.

FASTPATH:
Refer to *Defining Form Rules* for details on how to configure rules.

It's important for the payment to capture details that will facilitate the transfer of the payment to the correct obligation when the tax form gets processed. The payment should capture details like the taxpayer's identifier and the tax type / filing period being paid.

Payment Is Processed With The Tax Form

This usually occurs when payments are uploaded with tax forms.

In this case, the payment is not created until the form is processed. In order to process payments within tax form processing, a form rule needs to be configured.

FASTPATH:
Refer to *Defining Form Rules* for details on how to configure rules.

Additional Forms Processing Information

The topics in this section highlight additional information about forms processing.

Categorizing States

The business objects provided in the base product include a status option for the states in the lifecycle called Status Category. This option allows one or more categories to be assigned to the states such that other processing in this system related to forms that are in a given state do not require hard-coding of the status values. Following are examples of status categories used in the product for forms.

- Values identify the states in which the form is considered **In Effect** and the states in which it is considered **Not In Effect**. This setting may be used in code to allow or prevent certain actions from occurring based on the state of the form. For example, a user may not cancel an Obligation if there are any tax forms for the obligation whose state is **In Effect**.
- Values identify states that may need to be reviewed as being in the state for too long (**Status Too Long**).
- The **Posted** status defines a status category value for **Posted**, allowing for validation or processing based on whether a form is posted or not.

Controlling Automatic Processing

The goal when processing forms is to automatically process as many forms that can be automatically processed as possible, without anyone needing to review the form. However, for a form being added or corrected by a user, they system should not inadvertently pick up the form for processing in case a user is not yet finished with it. The way the system controls this differs slightly for Tax Form and Registration Forms. Note that in both cases, the logic described here is the logic supplied with the base product parent business object for each type of form. Implementations may choose not to adopt this business practice.

Tax Forms

Tax forms include a field called Automatic Post that is set to **Y** only for scenarios where a user is not working on the form. The tax form background processes provided with the base product to process the forms only select forms with this flag set to **Y** along with the other selection criteria included in the background process.

- A form that is created manually does not set the flag to **Y** so it will never be picked up for batch processing while in the **Pending** state. This applies to audit tax forms, tax forms that were created as a result of a transfer, tax forms that were created as a result of an adjustment and tax forms that are simply created manually. A user is able to spend as much time as needed to create the form (and do interim saves). When the form is fully entered, the user clicks **Validate** or **Validate and Post** to progress the form. If the user clicked **Validate and Post** and the form does not have any exceptions, the user is finished. Otherwise,
    - If the user clicks **Validate** and the form passes validation, it transitions to the **Ready for Posting** status. The user can review the form at this point and make additional changes if needed using the **Re-edit** button. Once the form is in a state where the user is happy with it, the user can click **Post** to transition the form to **Posted** real-time or can click **Batch Post**, which sets the Automatic Post flag to **Y** such that the form will be processed the next time the batch monitor process runs.

      NOTE: For implementations that use form control functionality, the tax form is only progressed to posted by linking it to a form control that gets approved. Refer to The *Big Picture of Form Control* for more information.

    - If the form does not pass validation and transitions to the **Suspended** status, the user may correct the form as needed. Once the form passes validation and transitions to **Ready for Posting**, the logic as described above applies.
    - If a validation form rule indicates that additional information is needed and transitions to the **Waiting for Information** status, the same logic applies as for the **Suspended** status. When a user determines that the needed information is received and re-validates the form, the form eventually transitions to **Ready for Posting** and the logic as described above applies.
- The form upload process sets the Automatic Post flag to **Y** when creating a form. Therefore, forms uploaded into **Pending** state are processed by the batch monitor assuming the other selection criteria configured the batch monitor are satisfied.
    - If the form does not have any exceptions, it progresses to the **Posted** state and the form is finished.
    - If the form does not pass validation and transitions to the **Suspended** status or the **Waiting for Information** status, the automatic post flag is reset to null because at this point a user needs to get involved to progress the form. The logic as described for manually created forms applies. Once the form is corrected and is in **Ready for Posting**, it can be progressed further as previously described.
- A special background process exists to detect tax forms that have been waiting too long in a given state. This allows implementations to detect manually created or corrected forms that are not progressing; and does this outside of the standard form processing batch jobs. The background process is **C1-TXSTL**.

Registration Forms

The process for registration forms is a bit simpler. It does not include an Automatic Post flag.

- A form that is created manually is not picked up for batch processing while in the **Pending** state. The background monitor process plugged into the **Pending** state only looks for forms that were uploaded along with additional selection criteria. (Refer to the background process **C1-RGMU** for more information). A user is able to spend as much time as needed to create the form (and do interim saves). When the form is fully entered, the user clicks **Validate** or **Validate and Post** to progress the form. If the user clicked **Validate and Post** and the form does not have any exceptions, the user is finished. Otherwise,
    - If the user clicks **Validate** and the form passes validation, it transitions to the **Ready for Posting** status. The user can review the form at this point and make additional changes if needed using the **Re-edit** button. Once the form is in a state where the user is happy with it, the user can click **Post** to transition the form to **Posted** real-time or can simply let the form be picked up the next time the batch monitor process runs.

- If the form does not pass validation and transitions to the **Suspended** status, the user may correct the form as needed. Once the form passes validation and transitions to **Ready for Posting**, the logic as described above applies.
- If a validation form rule indicates that additional information is needed and transitions to the **Waiting for Information** status, the same logic applies as for the **Suspended** status. When a user determines that the needed information is received and re-validates the form, the form eventually transitions to **Ready for Posting** and the logic as described above applies.

- Forms uploaded into **Pending** state are processed by the batch monitor assuming the other selection criteria configured the batch monitor are satisfied.
    - If the form does not have any exceptions, it progresses to the **Posted** state and the form is finished.
    - If the form does not pass validation and transitions to the **Suspended** status or the **Waiting for Information** status, the automatic post flag is reset to null because at this point a user needs to get involved to progress the form. The logic as described for manually created forms applies. Once the form is corrected and is in **Ready for Posting**, it can be progressed further as previously described.
- A special background process exists to detect registration forms that have been waiting too long in a given state. This allows implementations to detect manually created or corrected forms that are not progressing; and does this outside of the standard form processing batch jobs. The background process is **C1-RGSTL**.

The Big Picture of Form Definition

Form definition refers to the process of defining a new form in the system. The product provides a single Form Type control table that is used to define details about each registration form and tax form in your implementation. The form type defines:

- The sections and lines that make up the definition of the form. This includes the format and some basic validation for the values entered in the form line.
- The form rules that should execute when a form of this type is processed.

Designing Form Types

A Form Type is configured for each type of form that the system needs to process.

The following are the steps in configuring a form type:

- Define basic information about the form type, including the business object that controls the behavior of the form type. The base product provides the **C1-StandardTaxFormType** and **C1–FilingCalendarTaxFormType** business objects, which can be used for automatic generation of tax form business objects. Also, determine the applicability of change reasons to the form type.
- Define the sections of the form.
- Define the lines that each section contains.
- Generate the form type.
- Define the valid form change reasons to the form type, if change reasons are applicable.
- Link the form type to form rule groups.

Designing Sections

When designing the sections of your form, determine the number of sections needed. Small or simple forms may contain only a few sections. Complex forms, such as forms with schedules, need a bit more design time, as they may include more sections, including sections that repeat.

Identify any recurring sections. This is common with tax forms that allow multiple occurrences of a schedule. For each recurring section, determine whether or not recurrence should be limited to a finite number.

CAUTION: Although the configuration allows for unlimited recurring sections, the HTML rendering for the user interface is unable to handle a large number of sections and lines. Be careful in using forms to capture data that results in a large number of sections and lines. Such forms need to be fully tested with expected volumes, to ensure that the forms are able to handle your business need.

Refer to the definition of business object **C1-FormSection** for more details.

Designing Lines

After establishing the sections, you need to define the information that is contained in each of the sections. Some sections could include one or more simple lines. Some sections may include one or more repeating groups of lines. Other sections may have a mix of simple lines and repeating groups.

Define each line. Most lines on a tax form are either character or numeric fields. These lines could simply contain a value or could capture additional information.

The base package supports three common types of lines: Standard Line, Group Line and Label Line

Standard Lines

Most form lines fall into this category. It provides the definition of a field on the form, including the label to display.

The base product provides the **C1-StandardLine** business object, which has the common structure of a form line.

A standard line can be configured to either create a new field or point to an existing field in the system. To create a new field, specify **Create New** in the Definition Field Usage and specify the data type, the applicable settings for the data type and the label definition. To point to an existing field, specify **Use Existing** in the Definition Field Usage and specify an existing field in Definition Field Usage.

The base business object supports most basic field data types that the Framework supports - i.e. character, number, money, date, boolean, etc.

Character field types can be validated using a **Validation Pattern** that uses an Extended Lookup, **C1-RegularExpressionLookup**. The Extended Lookup uses a regular expression to define the pattern. The validation is only performed if the Form Line Configuration Validation form rule, **C1-ChkReqSectionsLines** is included on the Form Type.

In addition, special data types are also supported: simple lookup, extendable lookup and foreign key reference.

Simple lookups are used when the line's possible values is a simple list of descriptions. To configure a form line to point to an existing lookup, specify **Use Existing** in the Definition Field Usage and specify an existing lookup field in the Definition Field Name. To configure a form line to create a new lookup field, specify **Create New** in the Definition Field Usage and define the lookup field and values.

NOTE: The initial list of values can be specified on the line configuration. However, once the lookup field exists, any changes to the lookup values must be done via Lookup maintenance.

Extendable lookups are used when the lookup values require additional information. For example, a list of counties may be set up using an extendable lookup. Each county can specify not only the description of the county, but also a distribution code. To configure a form line as an extendable lookup, the extendable lookups have to be defined first, via Extendable Lookup maintenance. When defining the form line, specify **Use Extendable Lookup** in the Definition Field Usage and specify the Extendable Lookup BO.

Foreign key references are used when the form line needs to contain a value that exists in one of the tables in the product. For example, a registration form may specify an industry code, which must be defined in the Industry Code table. To configure a form line as a foreign key reference, specify **Use Foreign Key Reference** in the Definition Field Usage and specify the Foreign Key Reference. In addition, specify the UI Map Usage, to indicate whether the values will be displayed as a dropdown list or as an input field with search.

NOTE: **Input Field** should be specified only if the foreign key reference has a custom search.

Form lines can be configured to map to a Related Form Field. The Related Form Field list consists of columns in the tax form or registration form. When a form line is mapped to a related form field, the line's value is copied to the corresponding table column. When mapping a form line to a related form field, specify the Form Line Mapping setting, to indicate whether the copy happens every time the form is updated or only when a form rule executes.

For each form line, the following elements may be configured:

- As Current - This is required. This element captures the line's current value.
- As Reported - This setting is optional. This element is used to display the original value that was reported for the line. It is populated when the form exits the Pending state.

  CAUTION: This setting should be carefully considered. This should not be enabled for forms with a large volume of lines because it can adversely impact performance. Users can refer to the Form Versions tab of the tax form, to view any of the form's versions, including the original version.

- Override Switch - This switch is used to bypass validation. For example, a form rule may be defined such that "If line 110 is populated, line 120 must have a value greater than 500." Suppose line 110 is populated and line 120 has a value of 499 and the business rule is that when this form suspends, a user can review it and make a decision that the value on the form is valid and that the rule should be bypassed. The user clicks on the Override switch and try to re-validate the form.

  CAUTION: This setting should only be enabled for form lines where a corresponding validation rule is designed to check this switch.

- Change Reason - This is optional. It allows a change reason to be supplied at the form line level (in addition to change reasons at the form level, if applicable). When a user changes the current value of a line, this change reason can also be supplied.

  CAUTION: Capturing a change reason at the line level requires an additional element and thus, adds to the size of the form. This should be carefully considered.

All of the above mentioned elements have a Display Label setting. To ensure the proper display of line details, this setting should always be the same for all applicable elements.

CAUTION: HTML rendering considers labels as elements or widgets too. Therefore, labels add to the form's size. Carefully consider when the labels for these extra elements should display. For instance, when the line only has As Current is applicable, it's probably not useful to display the 'As Current' label. More importantly, when enabling As Reported, Override and/or Change Reason on a repeating group of lines, the 'As Current', 'As Reported', 'Override' and 'Change Reason' labels should be enabled for the first line only.

Group Lines

This type of line is used for defining a repeating group of standard lines. This line contains header information about the group. Every line that belongs to the group refers to their group line.

Determine whether or not recurrence should be limited to a finite number.

Refer to the definition of business object **C1-GroupLine** for more details.

CAUTION: Although the configuration allows for unlimited recurring groups, the HTML rendering for the user interface is unable to handle a large number of repeating groups. Be careful in using forms to capture data that results in a large number of lines. Such forms need to be fully tested with expected volumes, to ensure that the forms are able to handle your business need.

Also, nesting of groups (i.e. groups within groups) is not supported.

Label Lines

This line type is used for displaying text on the form.

Refer to the definition of business object **C1-LabelLine** for more details.

Address Form Lines

The entities that capture addresses in the product break down the constituents into separate fields. An implementation defines the appropriate constituents for each Country and allows for each constituent to have an appropriate label. This way, when displaying addresses in the system, the appropriate fields appear with the appropriate labels.

FASTPATH: Refer to *Defining Countries* for details on how to configure address constituents.

When considering addresses captured by tax forms or registration forms, implementations should consider how the data is captured and how this data is later mapped to the appropriate constituents based on the country defined on the form. The following points should be considered:

- Should all possible countries be loaded in the Country table? For validation purposes, an implementation may choose to load all countries. However, for countries that are not common, it is recommended to choose a consistent set of constituents for mapping from the form and capturing in the master data in the system. If implementations do not wish to load all the countries in the country table, an alternative is to define a special "Other" or "Unknown" country. In this case one of the free format constituents available may be used to capture the country value.
- Consider the definition of State. The product provides a constituent for State and in this case it expects valid states to be defined in the State table (also on the Country page). Implementations will probably not want to define the valid states for all the countries defined in the Country table. However, the Country table provides many free-format fields that available to define the "state" for a country where the list of possible states is not maintained. For example, assuming County information is not captured for most foreign countries, the County field can be used. It's label can be configured to be "State" or "Province" or whatever is appropriate. Address 4 is another possibility if it is rarely used.
- Once the address constituents for different countries are defined, consider forms processing. Forms have a defined set of fields for capturing addresses. When attempting to map those form lines to the system's address components, the address columns on the Form need to be populated based on the country configuration. For countries where the State checkbox is checked, the data on the form representing the State should go into the STATE column and will be validated. For countries where the State checkbox is not checked, the data on the form representing the State should go into the agreed upon other column (for example. COUNTY).

Assigning Business Names to Sections and Lines

Sections and lines are assigned business names to give form rules configuration a way of referencing the sections and lines, without necessarily knowing the specific form types that contain them. This allows form rules to be configured for reuse across several form types.

The actual instances of the sections and lines within a form type are determined when the rules are executed.

FASTPATH:
Refer to *Defining Form Rules* for details on how to configure rules.

Designing Form Change Reasons

As part of designing your form types, you will need to decide if the form should require the user to supply a change reason when modifying a form. The form type includes the following configuration:

- **Overall Change Reason Applicability** - this controls whether change reason is used at the overall form level.
- Set this to **Required** if the user needs to enter a change reason when making changes to a form.
- Set this to **Optional** if the user can choose to enter a change reason but is not required.
- Set this to **Not Applicable** if this form does not use overall change reasons. This will suppress the change reasons grid and comments from the main section of the form's UI map.
- **Line Change Reason Applicability** - this controls whether change reasons are used at the individual form line level.
- Set this to **Applicable** if this form uses change reasons at the form line level for one or more of its form lines. The change reasons definition checkboxes will be enabled when defining a form line.
- Set this to **Not Applicable** if this form does not allow change reasons at the form line level. The change reason definition checkboxes will be disabled when defining a form line.

  In addition to controlling which change reasons fields appear on the form's UI map, the applicability flags also control the validation for change reasons on the form. See *Determining When a Change Reason is Required* for more information.

  If form change reasons are going to be used with this form type, then a list of one or more change reasons needs to be defined. This list defines the valid change reasons for the form type and is used to create the dropdowns on the form for the user to choose from.

  Some change reasons are used by algorithms that update a form, rather than a user. These change reasons will have a system default flag associated with them and will not appear on the dropdown presented to the user. A lookup defines which system transitions can be used. Additional values can be added by an implementation as needed.

  The base package provides the algorithm type *C1-DFLTFCR* that defaults a change reason when a form is suspended or is waiting for information. If your form's business object is using the business object **C1-ParentTaxForm**, you will need to have one change reason with the system default flag of **Form Transition**.

Generating Forms

The base package provides form type business objects that can be used for generating most tax form types.

- **C1-StandardTaxFormType** has a lifecycle that includes a generation step and that allows for regenerations when needed.
- **C1–FilingCalendarTaxFormType** extends **C1-StandardTaxFormType** with a collection of valid filing periods.

The Lifecycle of a Form Type

A form type is created and stays in a 'pending' state while the details about the form type, its sections and its lines are being configured.

Once the structure of the form type, sections and lines are defined, the next step is to generate the form business object and its UI maps.

If form generation encounters any problems, the form type goes into an error state. The user needs to fix the issue(s) and regenerate the form.

On the other hand, if form generation is successful, the form type goes into a 'generated' state until the user activates it.

Activated forms can be inactivated.

If changes need to be made after a form type is already generated, a user could put the form type back into the 'pending' state and change / regenerate the form type accordingly.

The same could be true for activated or inactivated form types - if changes are allowed when the form type is already in either of these states.

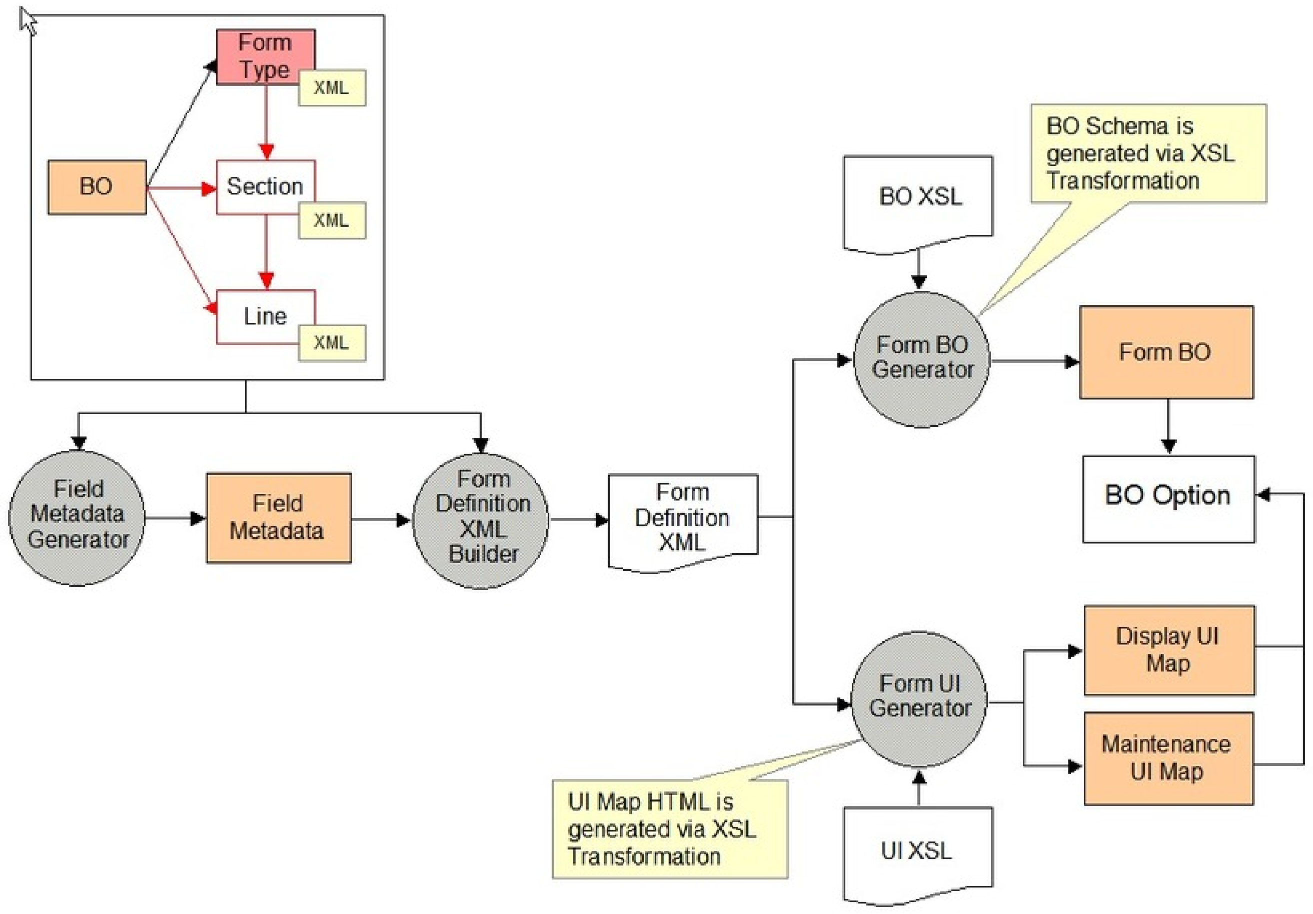
In addition to the actions that move the form type through the states described above, the **Edit, Delete** and **Duplicate** actions are available.

For form types that are activated, a special action of **Generate OPA Rule** is provided.

FASTPATH: Refer to ***Generating an OPA Rulebase Data Model for a Form Type*** for more information.

The Form Generation Process

The following diagram illustrates the form generation process flow.



Form Generation

The steps are discussed in the following sections.

Generating the Form Type

When using the base business object **C1-StandardTaxFormType** or **C1–FilingCalendarTaxFormType** for generating tax forms, the form generator is invoked when the form type goes into the Generated state.

An 'enter' algorithm (*C1-GEN-FORM*) invokes the following processes (in order):

- Form Business Object Generation
- Form Display Map Generation
- Form Maintenance Map Generation

All three processes have to run successfully.

If any of the processes fail, the algorithm creates an appropriate log entry to indicate which one failed. Any components that were generated prior to hitting the error are rolled back.

Form Business Object Generation

This process creates a form business object using the form definition data on the form type. It includes intermediate steps for: creating Field metadata, creating Lookup Fields / Values and extracting form definition data.

Field Metadata Creation

This process goes through the form type's section and line definitions and creates label fields, definition fields and override label fields based on the section and line configurations. For definition fields that are configured as lookups, this process has an additional step of creating the Lookup Fields and their Lookup Values. The generated lookup fields and field values are automatically populated with Java names, so that backend logic can reference them accordingly.

Form Definition Extract

This process extracts the form type / section / line definitions into one XML document that feeds into both the business object creation and UI map generation processes.

Form Business Object Creation

This process creates the form business object using the form definition XML data that was extracted in a prior step. The form business object is created as a 'child' BO that inherits from a parent form business object - as defined on the form type business object's 'Form Parent Business Object' option.

NOTE:
The **C1–FilingCalendarTaxFormType** and **C1-StandardTaxFormType** business objects reference **C1-ParentTaxForm** as their 'Form Parent Business Object'.

Any elements that are common to all forms should be placed in the parent form BO's schema. Other form elements that vary between form types should have corresponding Line definitions. Section and Line definitions are generated into the form BO schema accordingly. Elements in the BO schema will vary depending on the type of line. For ‘standard’ lines in particular, the schema attributes differ based on the Line’s Definition Field Usage:

- Simple input fields are mapped to existing or new simple Field metadata.
- Foreign key reference fields are mapped based on the related Foreign Key Reference code.
- Extendable lookup fields are mapped based on the related extendable lookup BO.

This process also creates the Application Service for the generated business object. The associated access modes are the same as the parent business object's access modes.

NOTE:

These generated Application Services must be configured on your user groups accordingly.

Form Display / Maintenance UI Map Generation

These processes create the form's display and maintenance UI maps and automatically link the maps to the generated form BO.

The 'Form Display/Maintenance UI XLS Stylesheet' specified on the form type business object controls how the HTMLs in these maps are generated.

Reserved Primary Keys

The form generator populates the primary keys of the generated components as follows:

- Form BO code (max length 30) = Form Type code (max length 12)
- Display UI Map code (max length 30) = Form Type code (max length 12) + 'Display'
- Maintenance UI Map code (max length 30) = Form Type code (max length 12) + 'Maint'
- BO Application Service ID (max length 20) = Form Type code (max length 12 - note: this is also the generated BO code) + 'BOAS'

If initial generation of BOs / UI maps / application services finds a conflict with any existing CM primary keys for the same objects, form generation will result in an error. Either the existing CM objects or the new Form Type code should be changed accordingly.

Generating an OPA Rulebase Data Model for a Form Type

If your implementation is planning to validate the whole form using OPA, proceed to generate the OAP rulebase model.

Use the action button that becomes available once the Form Type is successfully generated and activated.

Save the data model file locally. The system stores the data model as an attachment and links it to the Form Type via log entry. Use Form Type Log to locate and download the latest model at any time.

NOTE: Change the data model file extension to ***xsrc*** before making an attempt to import the data model OPA.

FASTPATH: Refer to *Using the OPA Validation Form Rule* for specifics on how to create the OPA Rule using Pre-generated Data Model.

Rule Details

There is no run-time synchronization between OPA determinations server and the base product, therefore no updates happens when new rulebase gets deployed on the integrated OPA instance. This is an advisable housekeeping practice to update the OPA Rulebase Data Model with name of the rulebase created with this model.

Use **Admin –> OPA Rulebase Data Mode**l search to locate the data model and update it’s collection of related rulebases.

Regenerating a Form Type

Refer to *The Lifecycle of a Form Type* for information on when / how a form type is regenerated.

Once the form business object / UI maps / application services exist, any subsequent form type regeneration will retain the primary key, as well as most of the references associated with the generated objects - e.g. BO options, BO plug-ins, etc. Only the following updates would take place:

- The existing BO's schema is replaced.
- The existing UI maps' HTML is replaced.
- If the parent form BO referenced on the form type has changed since the last generation:
- The Parent BO and Maintenance Object references on the existing BO are replaced.
- The access modes on the child application services are replaced.

NOTE: If the structure of the form (sections/lines) was changed, the corresponding OPA rulebase data model should be regenerated in order to reflect these changes. The system issues a warning if it finds an existing Form Type log entry that is referencing an attachment whose BO is **C1–OPARulebaseDataModel**.

Overriding the Generated UI Maps

The look-and-feel of the generated UI maps is controlled by an XSL stylesheet that is specified on the form type business object's 'Form Display/Maintenance UI XSL Stylesheet' option.

Overriding All Generated UI Maps

If your implementation needs to override the look-and-feel of all generated UI maps, you can specify your own XSL Stylesheet on the form type business object.

If you are using any of the base form type business object, you need to add a 'Form Display/Maintenance UI XSL Stylesheet' option to the **C1-StandardTaxFormType** BO and specify a higher sequence number on that option.

Overriding Specific Generated UI Maps

If your implementation needs to override the look-and-feel of specific UI maps or if you need to add implementation-specific logic to the generated UI maps, you can do the following:

- Make a CM copy of the generated UI maps and put the additional logic on those copies.
- Plug the CM UI maps into the corresponding BO option types on the generated form BO. Use a higher sequence number than the base-generated BO options. The 'UI map' BO options types are single-instance options, for which the FW knows to use the options with the highest sequence number.

NOTE:
Generated UI maps will use sequence number 10. CM maps must use sequence numbers higher than 10. If your CM maps do not follow this rule, you will lose your CM maps' links to the BO after regeneration.

When the BO and generated UI maps are regenerated, the form generator process simply replaces the XML in the BO schema and the HTML in the UI maps. The BO options added from the initial generation are retained. Thus, any CM UI maps would continue to be in effect.

After regeneration, your implementation must compare the generated maps with the override maps and copy the differences to the override maps accordingly.

Designing The Form's Processing Rules

The final step in configuring a form is the definition of processing rules.

FASTPATH:
Refer to *Defining Form Rules* for details on how to configure rules.

The Big Picture of Form Versions

A form is likely be modified several times during its lifetime. Whenever one or more changes are made to a form, a new form version is created. The rules that determine which changes constitute a new form version vary from one tax authority to another.

In general, there are two main categories of form changes:

- User correction. These are changes made to a form by a user for a variety of reasons such as correcting a scanning error, correcting a taxpayer error, update of a mailing address resulting from a taxpayer phone call, or correction of a form's receive date.
- Auto correction. These are changes made automatically by the system for reasons such as correction of a calculation error.

The tax authority requires the ability to track changes to a form for a couple of reasons:

- Audit Trail. The need to track who made the changes to the form and when the changes were made.
- Taxpayer Inquiry. The tax authority needs to be able to explain to the taxpayer any differences between the information that the taxpayer reported and the current information on the form.

How are Form Versions Created

A form version is created when the form is modified either by a user or by a backend process (e.g. algorithm, service). The base package business objects for tax form (**C1-ParentTaxForm**) and registration form (**C1-ParentRegistrationForm**) include two algorithms, which take a snapshot of the form that is being changed and store it in a special type of log record.

This log record, with a log type of **Form Version**, includes the details of the form version such as the user who made the change, the date and time of the change and a snapshot of the whole changed form.

The form versions get created in the following scenarios:

- A pending form is transitioned to the next state. When the form exits the **Pending** status, an algorithm creates the initial form version. This version is also known as the 'As Reported' version. Its purpose is to capture the form's data as reported by the taxpayer.
- A form enters a state that supports versions. The base business objects have configured an enter plug-in for states that enable versions which captures a version. This caters for scenarios where rules may have made updates to the form since the last version was captured.
- A form is modified and is in a status that indicates a version should be created. Each base business object includes the audit algorithm that creates a form version if the form is in a status that has the status option **Enable Versions** set to **Y** and any of the form's fields have changed.

Form Version Display

A form version is stored as a snapshot on the tax form log record. A user can view this form version on the **Tax Form - Versions** portal. This portal contains a list of the form versions for the tax form in context. A user can select a form version from the list and have it displayed below.

The form version is displayed using a derived map zone, which utilizes the same UI map used by the form itself. A derivation script looks at the form's business object options to determine which display UI map and display map service script should be used. This allows the form version to have the same look and feel as the form itself.

Form Changes Typically Require a Reason

Typically when a user makes a change to a form, they will be required to enter a reason for making the change. Some tax authorities require a user to enter a specific reason for each line that is changed, while others only require a more generic reason that describes the nature of the change for the overall form. The types of form changes supported by the base package are:

- Overall Change Reason. This change reason is specified at the form level and is used to describe the nature of the changes made to the form. Some examples include: corrected scanning error, data entry, and form rules auto correction.
- Line Change Reason. This change reason is specified at the form line level and is used to describe the specific change made to a form line. Some examples include: additional information provided by taxpayer, supporting documentation not accurate. It is possible for some form lines to be associated with a form change reason while others do not. For example, a change to an address might not require a change reason, while a change to the deductions amount will. The forms definition facility allows you to define the lines that need a change reason.

The form type controls which type(s) of change reason are applicable to the form, if any. A form type can be set up to require overall change reasons, form line change reasons, both or none. See *Designing Form Change Reasons* for more information on how a form type can be set up to use change reasons.

Determining When a Change Reason is Required

The base package business objects for tax form (**C1-ParentTaxForm**) and registration form (**C1-ParentRegistrationForm**) include a validation algorithm that use the following conditions to determine if a change reason is required:

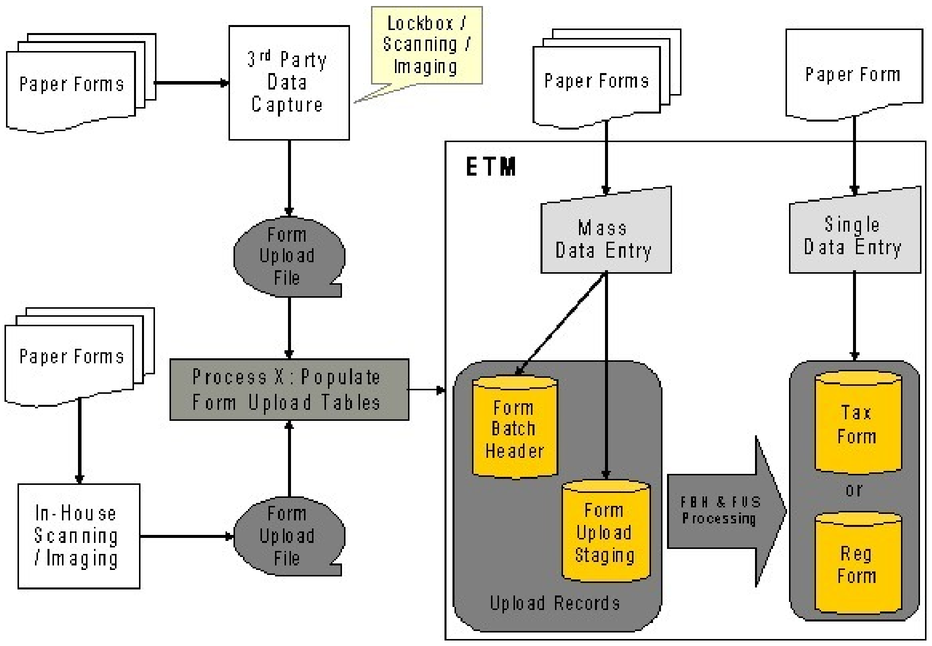
- Change reasons are only required if the status of the form is configured to enable versions.
- If the form type indicates that the Overall Change Reasons Applicability is **Required**, at least one form change reason should be supplied. On the other hand, if the form type indicates that the Overall Change Reasons Applicability is **Not Allowed**, form change reasons cannot be populated.
- If the form type indicates that the Line Change Reason Applicability is **Applicable**, a form change reason will be required for each line that has changed that is defined to have a form change reason. On the other hand, if Line Change Reason Applicability is **Not Applicable**, no line change reasons should be specified.
- Any form change reason that is populated on a form must be valid for the form's form type.

Uploading Forms

The topics in this section describe logic related to uploading batches of forms.

Forms Can Come From Different Sources

Forms are created in the system through a number of different ways. These methods of entering forms in the system are often referred to as 'channels'.



Most forms are uploaded in batch. This is a two-step process:

- Your implementation populates the form upload tables. That is referred to as 'Process X'.
- The base product processes the upload data and creates the corresponding tax forms and / or registration forms.

See *Designing Form Sources* and *Setting Up Form Sources* for details on how to configure your form sources.

Form Batch Header Groups Form Upload Records

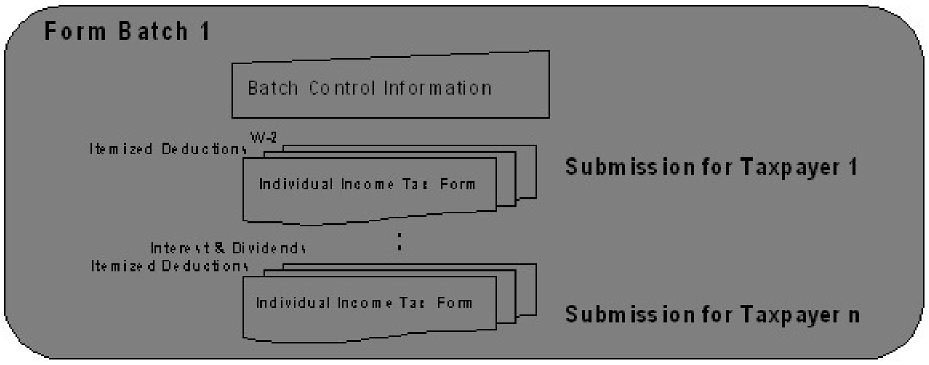
A form batch header is used to group form upload records together. It holds control information - e.g. number of form upload records, total payment amounts, etc.

A form batch header also controls the types of forms that can be put together in the same batch.

Refer *Designing Form Batch Headers* for details on how to configure form batch headers.

A Batch Can Group Forms Into Submissions

A batch can contain one or more submissions. A submission or a 'return' is a group of related forms that a taxpayer submits. It usually contains a main tax form and zero / one / more child forms. An example is an individual income submission where the income tax form is accompanied by a number of schedules and statements.



Form Batch 1
Batch Control Information
Itemized Deductions
W-2
Individual Income Tax Form
Submission for Taxpayer 1
Interest & Dividends
Itemized Deductions
Individual Income Tax Form
Submission for Taxpayer n

The product expects all documents supporting a single submission or 'return' to be included in a single form upload staging record that results in a single tax form or registration form, with sections representing the separate schedules and statements.

Form Upload Staging Holds Form Data

Actual tax / registration form data is uploaded via form upload staging record. A form upload staging record references a type, which determines how the upload data is mapped and stored in into the tax / registration form tables. The tax / registration form's type determines the target form business object to which the upload data will be mapped. For 'standard' form lines, the values from the upload record are mapped to the corresponding As Current elements in the target form business object schema. Refer to *Standard Lines* for more information on standard line configuration.

If tax / registration form creation is successful, the created form is linked to the form upload staging record.

Form Upload Staging Type Controls Everything

Each form upload staging record has a reference to its type. Form upload staging type controls how form upload data is stored into the product's tax form and registration form tables.

The bulk of your forms upload configuration will be spent configuring form upload staging types and defining the mapping of data from the different sources to your target form schemas in the system.

If your form sources each use a different record format / layout for interfacing a given form, a form upload staging type must be defined per unique combination of the destination form type and form source.

The mapping of the input form data into the target form schemas is done in the **Map Form Upload Data** algorithm plugged into the business object for the form upload staging type. The base product provides a form upload staging type BO **C1-FormUploadStagingType** that uses XSL transformation to map the raw XML data into the target tax / registration form schemas.

NOTE:
Base plug-ins. Click *here* to see the algorithm types available for this system event.

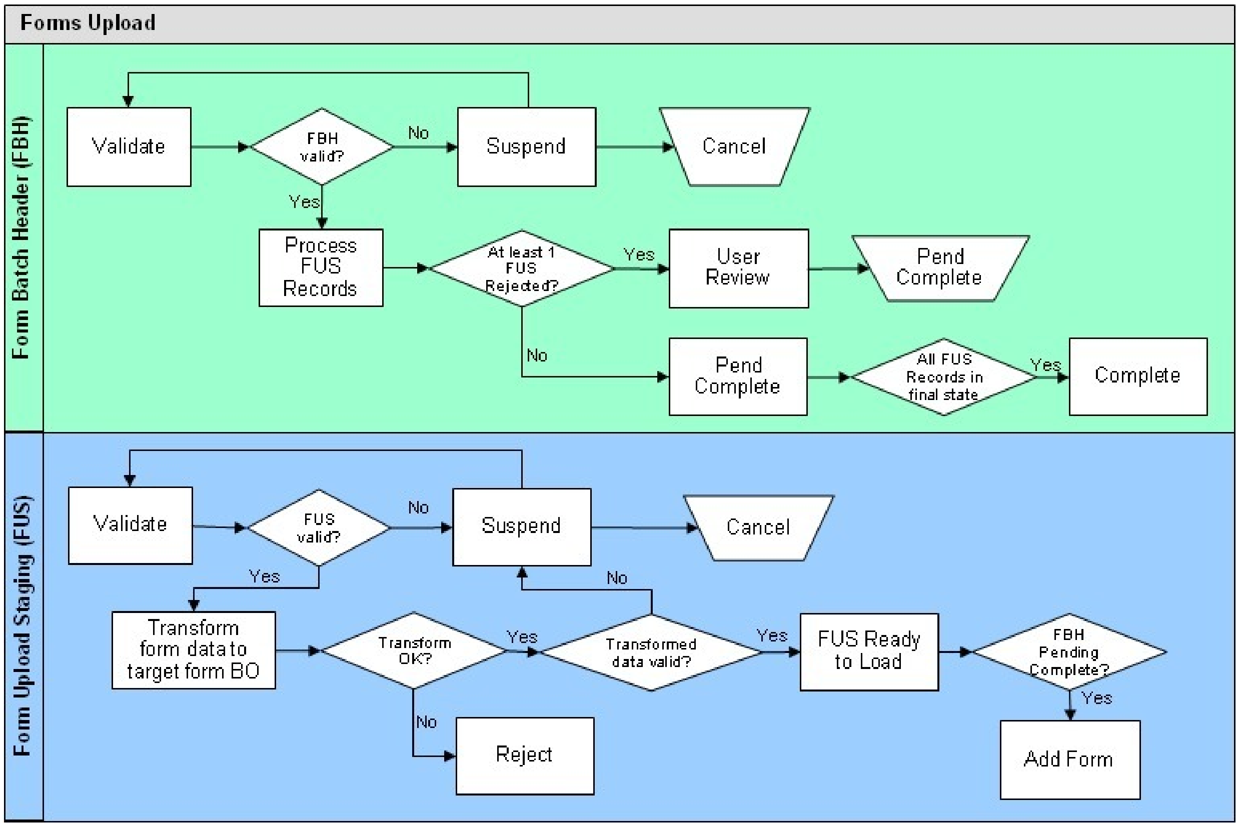
Refer to *Designing Form Upload Staging Types* and *Setting Up Form Upload Staging Types* for details on how to configure your form upload staging types.

Forms Upload Process Flow

Process Flow for Form Batch Headers and Form Upload Staging Records

The following diagram illustrates the general process flows for form batch headers and form upload staging records. This describes the lifecycles of the base product business objects supplied for form batch header (**C1-**

StandardformBatchHeader) and form upload staging (**C1-FormUploadStaging**). The expectation is that implementations should be able to use these base business objects as supplied.



Form upload staging records are not processed until the form batch headers they belong to have passed validation.

Any errors during the validation of a form batch header cause it to suspend for user review. The user is responsible for resolving the issues and re-validating the form batch header. At this point, the form upload staging records are still in their initial states. In some cases, the user may decide that the batch needs to be canceled, in which case the related form upload staging records are also canceled.

When the batch header passes validation, its form upload staging records can start processing. The batch sits in the state **Form Upload In Progress** until either all staging records are in some final state - i.e. form added successfully, rejected or canceled.

Each form upload staging record goes through a number of processing steps before the corresponding tax form or registration form can be added. Any issues from these processing steps prevent the corresponding form from getting added.

- The form upload staging is validated. This validation is not related to the specific form data attached, but rather more general issues with the staging record itself. If any errors are found, the form upload staging transitions to **Suspended** requiring a user to review. The user is responsible for resolving the issues and re-validating the form upload record. In some cases, the user may decide that the form upload staging needs to be canceled. Note that at this point, the corresponding form is not created yet.
- When a form upload staging suspends, its batch header status does not change. The idea is that a user may be able to fix the issue with the staging record or decide to cancel it. Either action on the current staging record should not prevent other staging records in the batch from being processed.
- When the form upload staging passes validation, it goes through the important step of mapping the fields from the raw XML into the target Form business object schema.
    - An error may be received in the process of attempting to map the data. This is typically a result of a problem with the submission and not something a user is able to fix. In this case, the record is rejected.

  - The mapping algorithm may perform some basic form validation, for example to detect missing data such as a receive date. In this case the record is suspended and a user must fix the problems before continuing.
  - If the transformation is successful and no validation errors are found, the form upload staging goes into a state where the tax form or registration form can be added / loaded.

If any of the form upload staging records are rejected, the batch header transitions to **Review Needed** so that a user can review the batch and decide what to do next. A user may take either of these actions:

- If a large number of records were rejected, the user may decide that there is something corrupt with the transmission and choose to Cancel the batch. Canceling the batch also cancels its form upload staging records. Note that at this point, there are no forms existing yet because the batch is in a cancelable state.
- If a small number or records are rejected, the user may decide to let the batch complete and determine the specific issues related to the rejected records.

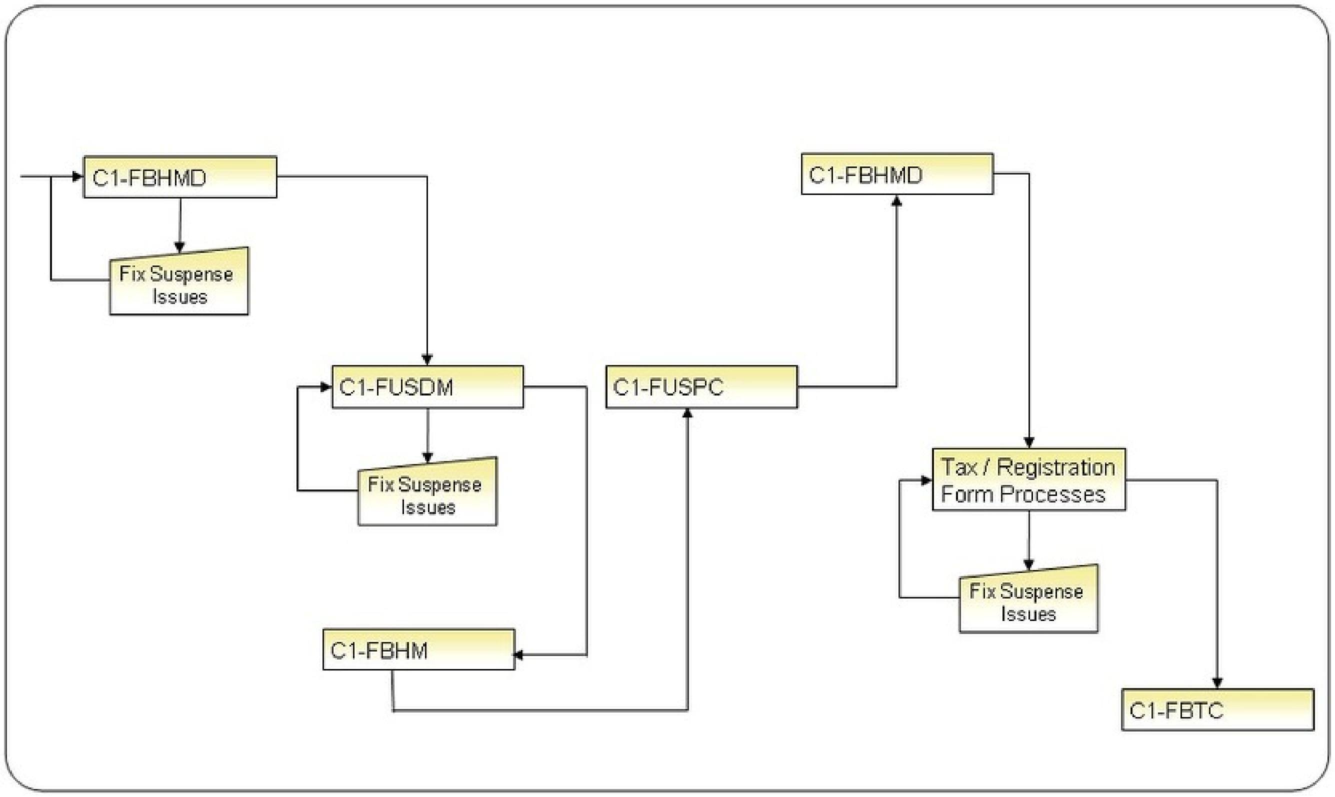
When all form upload staging records are either in **Ready for Load** state or are **Canceled**, the batch transitions to the **Ready to Complete** state. This interim state is one where batch cancellation is no longer possible. This ensures two things:

- That forms for that batch are added at the same time
- That once the forms are added, the batch cannot be canceled. Allowing batch cancellation when some forms are already added would require cleaning up the forms - i.e. canceling any pending / suspended / waiting forms get canceled and reversing any posted tax forms.

When the batch is in a **Ready to Complete** state, the forms in that batch can be added. When the forms are successfully added, their form upload staging records go to their final state, **Form Added**. Once all form upload staging records are in a final state, the batch is completed.

Forms Upload Batch Process Flow

The following diagram shows the forms upload process batch flow when using the base business objects for form batch header **C1-StandardFormBatchHeader** and form upload staging record **C1-FormUploadStaging**, and the base monitor background processes.



The processing of a form, from initial upload to posting is performed by the background processes as follows:

- **C1-FBHMD (Form Batch Header Deferred Monitor)**
  - Form batch header records transitioned from **Pending** to **Form Upload in Progress**.
  - Validates the form batch header records.
- **C1-FUSDM (Form Upload Staging Deferred)**
  - Form upload staging records transitioned from **Pending** to **Ready to Load**.
  - Validates the form upload staging records whose form batch header is in **Upload in Progress**.
- **C1-FBHM (Form Batch Header Monitor)**
  - Form batch header records transitioned from **Form Upload in Progress** to **Pending Complete**.
  - Updates the record counts on the form batch headers.
- **C1-FUSPC (Form Upload Staging Deferred Monitor)**
  - Form upload staging records transitioned from **Ready to Load** to **Form Added**.
  - Processes form upload staging records whose form batch header is **Pending Completion**.
- **C1-FBHMD (Form Batch Header Deferred Monitor)**
  - Form batch header records transitioned from **Pending Complete** to **Complete**.
  - Completes form batch header records.

  At this point the batch monitor processes related to processing tax forms or registration forms need to run. As part of this processing, the individual forms are validated and processed, including creating any payment, where applicable. Refer to *The Nightly Processes* for the list of all form related batch jobs.
- **C1-FBTC (Balance Form Batch Tender Controls)**

- Balances any tender control created for batch headers containing payments.

NOTE: The sections above describe the batch processing and lifecycle transition of the form batch header and form upload staging records using the base business objects. Since this functionality is BO driven, it can be customized based on your specific implementation requirements.

Validating Transformed Form Data

If a form upload staging record is unsuccessful in adding a form, a rollback is performed, the state is changed back to **Ready to Load** and an error is written to the batch run tree. The lifecycle does not support navigating to **Suspended** at this point. It is not common for there to be a problem in adding the form because tax forms and registration forms perform minimal validations when creating. It is possible that one of the few required fields is missing (such as Received Date). However, this is more likely pointing out a technical problem with the integration and not something that would occur in production.

In order to try to catch possible form errors earlier in the process where a user can review and update the form or where an implementer can identify an issue in the implementation, the base product Map to Form algorithm (**C1-FUS-MPFRM**) includes logic to optionally perform a step to validate the form. It uses the business service **F1-BOValidate**, which receives the target form's business object name and the transformed form data. The service executes schema validation as well as executing the BO validation algorithms. If any validation errors are detected, an issue is logged and the form upload staging record transitions to **Suspended**. At this point the user is able to fix the form before attempting to validate it again.

NOTE: Implementations may want to only run this validation during testing or with low volumes as the additional validation per form may have an impact on performance. A Feature Configuration Option is used to indicate if the form validation should be performed or not. Refer to *Configuring Transformed Form Validation* for more information.

Including Payments With Forms

Payments are uploaded with tax forms by specifying payments amounts on the corresponding form upload staging records. The actual sum of payment amounts from the form upload staging records within a batch are verified against the total payment amount specified on the form batch header.

The payment amount on the form upload staging record is simply copied over when the tax form is created. The actual payment may be created at different points in the tax form's lifecycle - depending on what's happening with the form.

- When the form posts, the obligation is already identified. Thus, the payment can be created for that obligation.
- When the form suspends, your implementation may want to create the financial transaction upfront and not wait for the suspense issue to be fixed. In this case, the payment can be created for a suspense obligation that is designated on the batch's tender source. When the form subsequently posts, the payment can be transferred to the correct obligation.
- When the form cancels, your implementation may want to create the payment anyway so that the financial transaction is still recorded in the system. In this case, the payment can be created for a suspense obligation that is designated on the batch's tender source. Manual follow-up is assumed in this case.

Refer to *Payments With a Tax Form* for more details on processing form payments.

Preparing To Upload Forms

The following topics describe the various considerations for preparing the system to upload forms.

Designing Form Sources

A form source is defined for each external source that interfaces forms into the system.

The following points describe a recommended design process:

- Define the categories for your form sources by updating the C1_FORM_CHANNEL_FLG lookup flag. Examples of these broad categories are: Lockbox, Third Party Data Capture, In-House Data Capture, etc.
- Determine the information about your external sources that the system needs to capture.
- Configure you tender sources. A unique tender source is usually associated to each form source.

The base product provides a form source business object **C1-FormSource** that maps all of the fields on the form source maintenance object.

If this BO's structure is sufficient for your implementation, you can use the BO as is. Your implementation can extend the BO's processing logic by adding / overriding algorithms and BO options.

If your implementation needs to capture additional information, you can: inherit from the base BO, copy the base BO or create your own BO.

Refer to existing help on configuration tools for more details on configuring business objects.

A Generic Form Upload Staging Business Object

The base product provides the form upload staging business object **C1-FormUploadStaging** that was designed to be generic such that your implementation can use it out of the box.

This BO has a raw XML (CLOB) field that is used to hold the actual form details. Staging the form details as raw XML provides the following benefits:

- A uniform way of interfacing form details:
  - For different types of forms
  - From various external sources that may upload data for a particular form type using different record structures/ layouts.
- External sources and Process X (i.e. the process that reads the input files and stores the form data into the form upload tables) do not need to know about the structure of the tax form and registration form business objects in the system.
- Form details can be uploaded into the system with minimal validation.
- As forms change in structure from year to year or as new form types are added, there is no need to create new form upload staging business objects. Your implementation just needs to configure new form upload staging types (that reference the generic form upload staging BO), and define the mapping logic for the new forms.

The mapping logic configured on the form upload staging types takes care of transforming the data. This BO also has a transformed XML (CLOB) field that holds the transformed data. The value of this field is used when the tax / registration form is added.

Use **C1-FormUploadStaging** if the structure and lifecycle suits your implementation's form batch headers. As with any base BO, your implementation can extend the BO's processing logic by adding/overriding algorithms and BO options.

If your implementation needs to capture additional information, you can do any of the following:

- Inherit from the base BO - if the base BO lifecycle and processing logic works for you.
- Copy the base BO or create an entirely new BO - if your form upload staging records have a different lifecycle

Refer to existing help on configuration tools for more details on configuring business objects.

Designing The Mapping of Upload Data From The Different Sources

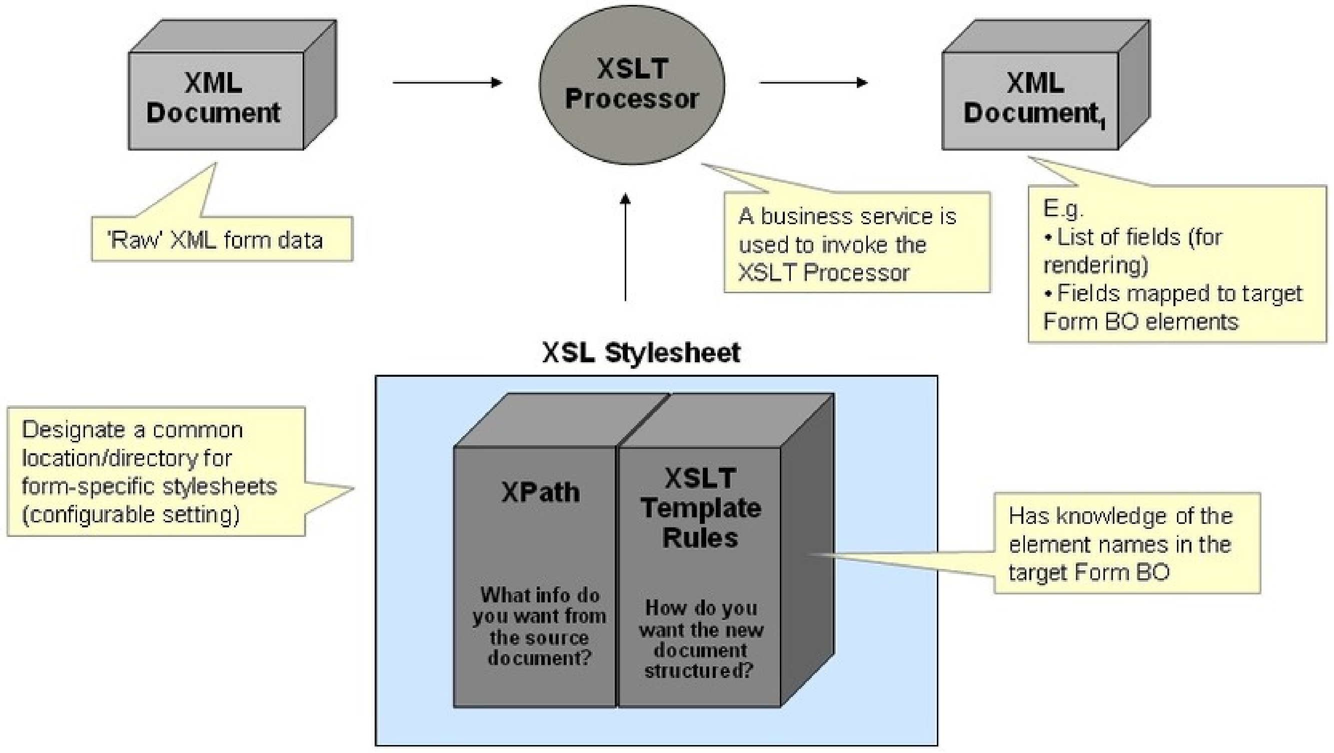
Given that the system expects the form data to come in as raw XML, your implementation needs to decide on the approach for mapping the raw XML data into the target form BO schemas. This decision usually entails looking at existing tools that are already available to your implementation.

Using XSL Transformation

The base product uses XSL transformation (XSLT) as an example for how to map the raw form upload data into the target form objects.

Since the input raw form upload data and the target form BO schemas are both in XML format, it makes sense to use XSLT. XSLT is a standard technique for transforming one XML document to another XML document. The key components in this approach are the XSLT Processor and the XSL style sheets.

The base product algorithm allows an implementation to configure whether the source XML to use for the XSL is the raw form data uploaded from the external source or if it is the full form upload staging business object. Refer to *C1-FUST-XSLT* for more information


XML Document
XSLT Processor
XML Document
'Raw' XML form data
A business service is used to invoke the XSLT Processor
E.g. • List of fields (for rendering) • Fields mapped to target Form BO elements
XSL Stylesheet
Designate a common location/directory for form-specific stylesheets (configurable setting)
XPath
What info do you want from the source document?
XSLT Template Rules
How do you want the new document structured?
Has knowledge of the element names in the target Form BO

XSL Transformation

The framework is already using an XSLT processor for transforming XMLs not only into other XMLs but also into HTMLs. A base business service calls an existing Java method for invoking that XSLT processor.

The base product provides a sample XSL file for mapping to the **C1-SalesAndUseTaxForm** business object - *C1-SalesAndUseTaxForm.xsl*. This XSL is configured to do a full business object transformation.

NOTE:

The sample Sales & Use XSL logic assumes a very specific 'raw' XML data form that was not based on any single real-life sales & use form. This sample XSL is just meant to demonstrate this particular mapping approach.

Both base product and CM XSL files need to be stored in specific locations in the file system designated at installation time.

- Base files are stored in **@SPLEBASE@/splapp/resources/xsl/**.
- CM files are stored in **@SPLEBASE@/splapp/resources/xsl/cm/**.

XSL Transformation Is Not Required

Although base logic uses XSLT for transforming form upload data, your implementation is not required to use this approach. The base logic that invokes XSLT is in a 'Map Form Upload Data' plug-in on the Form Upload Staging Type BO. You can override the base logic by inactivating the base logic and adding your own. This is done as part of configuring your form upload staging types, which is described in the next section.

Designing Form Upload Staging Types

The following topics describe the various considerations for configuring your form upload staging types.

Form Upload Staging Type (Admin) Business Object

The base product provides the **C1-FormUploadStagingType** business object. This object maps all of the fields on the maintenance object plus additional fields (in the CLOB) for the suspense To Do Type and To Do Role. Since the form upload staging type does not have a soft status, **C1-FormUploadStagingType** does not have a lifecycle.

Use the base BO if the structure suits your form upload staging types. Otherwise, you can either subclass the BO or create an entirely different BO. Refer to existing help on configuration tools for more details on configuring business objects.

When opting to use the **C1-FormUploadStagingType** BO and your chosen mapping technique is not XSLT, you need to do the following:

- Inactivate the base algorithm **C1-FUST-XSLT (FUS Type - XSL Transformation)** by adding a BO option to **C1-FormUploadStagingType** with:
  - BO option type = **Inactivate Algorithm**
  - BO option value = **C1-FUST-XSLT**.
- Plug in your CM algorithm on the **Map Form Upload Data to Target Form** plug-in spot on the **C1-FormUploadStagingType** BO.

NOTE: If your chosen approach is to not have mapping logic at all - i.e. Process X will populate the Transformed XML field in the staging record - you still need to plug in the CM algorithm, but the algorithm will do nothing but terminate.

C1-FormUploadStagingType also has BO options for automatic rendering of raw XML data on the UI.

- **Display HTML Generator XSL Stylesheet = @SPLEBASE@/splapp/resources/xsl/c1FormUploadStagingUIMapForDisplay.xsl**
- **Schema Extractor XSL Stylesheet = @SPLEBASE@/splapp/resources/xsl/c1FormUploadStagingSchemaForUIMap.xsl**
- **Maintenance HTML Generator XSL Stylesheet = @SPLEBASE@/splapp/resources/xsl/c1FormUploadStagingUIMapForInput.xsl**

NOTE: The generated maintenance HTML (third option) is just for editing raw XML data in existing form upload staging records. It is not used for adding form upload staging records online. A mass data entry solution is not included in this release.

If your implementation needs to render the raw XML data in a different way, you simply add option values for these three option types, using a higher sequence number. (For 'single' BO option types, the framework knows to get the highest sequence number.)

Defining Form Upload Staging Type Data

The following describes the steps in defining your form upload staging types.

- Determine the number of form upload staging types needed based on the number of your form types (that can be uploaded) and the sources that can interface those form types.
- Create a form upload staging type for each unique combination of form type and form source.
  - Select the form upload staging type (admin) business object. See *Form Upload Staging Type (Admin) Business Object* for details configuring this BO.
  - Specify the form upload staging type code (primary key)
  - Put in a description of the form upload staging type
  - Specify an **Active** status if you want to record to be effective immediately. Otherwise, specify an **Inactive** status and update the status later when the form upload staging type is ready.
  - Specify the related transaction business object - the form upload staging BO. The base product provides a *generic form upload staging BO* for this purpose.
  - Specify the form source.
  - Specify the form type.
  - If you are using XSLT to map raw XML form data to the target form BOs and you are using the base algorithm **C1-FUST-XSLT**, specify the XSL Transformation File Name. The file name must include the full path - e.g. **com/splwg/cm/domain/forms/formUploadStaging/xsl/salesAndUseForm.xsl**.
  - If you selected the **C1-FormUploadStagingType** administrative base BO, specify the suspense To Do type and To Do role.

Configuring Transformed Form Validation

As described in *Validating Transformed Form Data*, if your implementation wishes to enable validation of the transformed form schema during the Map to Form step, you must set up a *Feature Configuration* option. Find the Feature Configuration record for the **Forms Processing** feature type. (It may need to be defined if it does not exist). Add the **Option Type** of **Validate Upload Data**. Define the value **C1VA** if validation of the transformed schema should occur and **C1DV** if it should not.

NOTE: The default behavior is not to validate so there is no need to set up the feature configuration option if the validation should not occur.

Designing Form Batch Headers

The following points describe a recommended design process:

- Define the information that your form batch headers need to capture, e.g.:

    - The valid form types that can be included in the batch - i.e. single vs. multiple
    - Record counts
    - Total amounts
- Examine the lifecycle of your form batch headers. Part of this effort is figuring out the points in the form batch header's lifecycle in which there is 'handshaking' with the form upload staging records' lifecycles.
- Design your form batch header processing algorithms.
    - Identify issues that cause form batch processing to stop until a user can resolve the issues. Determine the appropriate notification action for such issues - e.g. To Do entries
    - Identify conditions that require a review of the form batch header. Determine the appropriate action for triggering such review - e.g. To Do entries
    - Identify conditions that cause form batch processing to complete. Determine the actions that need to take place at completion - e.g. final update of counters.
    - Etc.

The base product provides the **C1-StandardFormBatchHeader** business object, which is designed to work for both batch uploads and mass data entry. It allows for multiple valid form types within the same batch. It also includes processing counters that capture the results of processing the included form upload staging records. This BO's lifecycle supports cancellation, validation, suspense, review and completion and is designed to work with the lifecycle of the form upload staging BO that is also provided in base - i.e. **C1-FormUploadStaging**.

Use **C1-StandardFormBatchHeader** if the structure and lifecycle suits your implementation's form batch headers. As with any base BO, your implementation can extend the BO's processing logic by adding/overriding algorithms and BO options.

If your implementation needs to capture additional information, you can do any of the following:

- Inherit from the base BO - if the base BO lifecycle and processing logic works for you.
- Copy the base BO or create an entirely new BO - if your form batch headers have a different lifecycle

Refer to existing help on configuration tools for more details on configuring business objects.

Setting Up Form Processing Options

The topics in this section discuss the various options for setting up the processing of forms.

Setting Up The System To Enable Forms Processing

This table describes all objects that must be defined as part of the forms processing setup process. It lists the order in which objects should be defined. It also identifies the other objects that are associated with or referenced by each setup object, thus providing a useful map of the relationships between objects in the system.

Seq	*Object*	*Description*	*Prerequisite*	*Associated With / Referenced By*	*Admin Submenu*
1	Form Type	Type of form	Obligation Type	Form Section, Form Line, Child Form Types, Valid Obligation Types, Tax Form, Registration Form, Form Upload Staging Type	Form Type
2	Form Section	Section of a form type	Form Type	Form Type, Form Line	Form Type

Seq	*Object*	*Description*	*Prerequisite*	*Associated With / Referenced By*	*Admin Submenu*
3	Form Line	A line in a section. Defines the type of information captured / displayed on the line	Form Section	Form Type, Form Section, Field, Lookup Field	Form Type
4	Form Change Reason	Predefined reasons for changes to information on the form		Form Type	Form Change Reason
5	Form Rule Group	A logical grouping of form rules		Form Type, Form Rule	Form Rule Group
6	Form Rule	Defines how information on the form is processed	Form Rule Group	Form Rule Group, Registration Form Exception, Tax Form Exception	Form Rule
7	Form Source	Predefined sources of forms	Tender Source	Tender Source, Form Upload Staging Type, Form Batch Header, Form Upload Staging, Tax Form, Registration Form	Form Source
8	Form Upload Staging Type	Type of form upload staging record	Form Source, Form Type	Form Source, Form Type, Form Batch Header, Form Upload Staging	Form Upload Staging Type

Setting Up Form Types

Form Types contain the rules that control how forms are processed. To set up a Form Type, open **Admin Menu** > **Form Type**.

Form Type Query

Use the *query portal* to search for an existing form type. Once a form type is selected, you are brought to the maintenance portal to view and maintain the selected record.

Form Type - Main

This portal appears when a form type has been selected from the Form Type Query portal.

The topics in this section describe the base-package zones that appear on this portal.

Actions

This zone displays the actions that control the processing of the form type and generation of the form.

FASTPATH: Refer to *Generating Forms* for more information on the form type lifecycle and form generation.

Form Type

The Form Type zone contains display-only information about selected form type.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_FORM_TYPE*.

Section List

This zone lists the sections of the form type in sequential order. The Edit, Duplicate, and Delete actions are available. You can click the Add link in the zone's title bar to add a new section. You can click a form section hyperlink to go to the maintenance portal for that form section.

Line List

This zone accepts a broadcast form section and lists the section's lines in sequential order. The Edit, Duplicate, and Delete actions are available. You can click the Add link in the zone's title bar to add a new line. You can click a line hyperlink to go to the maintenance portal for that form line.

Child Form Types

This zone displays the list of tax form types to which the current form type is a parent and is only visible if child form types exist. The base product does not currently provide functionality that uses this form type relationship.

Form Type - Rules

To view the Form Type - Rules page, open **Admin Menu** > **Form Type** select a form type, and then navigate to the **Rules** tab.

This page displays the form type's rule groups in prime key order. A row is displayed for every rules linked to the form type rule group. The list includes the form rule information with a hyperlink to the rule's maintenance portal. You can add or edit the form type rule group by clicking the link in the title bar.

Form Type - Log

To view the Form Type - Log page, open **Admin Menu** > **Form Type** select a form type, and then navigate to the **Log** tab.

This page contains a standard *log zone*.

Setting Up Sections and Lines

Setting Up Form Sections

This portal appears when a section has been selected from the Section List on the Form Type portal or from the Current Form Type's Sections dashboard zone.

The topics in this section describe the base-package zones that appear on this portal.

Actions

This is a standard actions zone. The **Edit, Delete** and **Duplicate** actions are available.

Form Section

This zone contains display-only information about selected form section.

Please see the zone's help text for information about this zone's fields.

Form Section Line List

This zone accepts a broadcast form section and lists the section's lines in sequential order. The **Edit, Delete** and **Duplicate** actions are available. You can click the **Add** link in the zone's title bar to add a new line. You can click a line hyperlink to display the line details.

Setting Up Form Lines

This portal appears when a line has been selected from the Section Line List on the Form Type portal or the Form Section Portal.

The topics in this section describe the base-package zones that appear on this portal.

Actions

This is a standard actions zone. The **Edit, Delete** and **Duplicate** actions are available.

Form Line

This zone contains display-only information about selected form line.

Please see the zone's help text for information about this zone's fields.

Setting Up Form Change Reasons

Typically the tax authority will want to track the reason/s behind a form change, especially if the changes were made by a user. The user will be required to select from a dropdown of form change reasons whenever making a change that requires a reason. To set up a Form Change Reason, open **Admin Menu** > **Form Change Reason**.

The topics in this section describe the base-package zones that appear on the Form Change Reason portal.

Form Change Reason List

The Form Change Reason *List zone* lists every form change reason. The following functions are available:

- Click the *broadcast* icon to open other zones that contain more information about the adjacent form change reason.
- The standard actions of **Edit, Delete** and **Duplicate** are available for each form change reason.
- Click the **Add** link in the zone's title bar to add a new form change reason.

Actions

This is a standard actions zone. The **Edit, Delete** and **Duplicate** actions are available.

Form Change Reason

The Form Change Reason zone contains display-only information about the form change reason. This zone appears when a form change reason has been broadcast from the Form Change Reason List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Form Types

The Form Types List zone displays a list of form types that currently use the broadcast form change reason. The following functions are available:

- Click the **Add** link in the zone's title bar to link a new form type to the broadcast form change reason.
- Use the **Delete** action to delete a link between the broadcast form change reason and a form type.

Setting Up Form Sources

A form source is created for every external source that interfaces tax / registration forms into the system.

To set up a form source, select **Admin Menu** > **Form Source**.

The topics in this section describe the base-package zones that appear on the Form Source portal.

Form Source List

The Form Source *List zone* lists every form source. The following functions are available.

- Click the *broadcast* icon to open other zones that contain more information about the adjacent form source.
- The standard actions of **Edit, Delete** and **Duplicate** are available for each form source.
- Click the **Add** link in the zone's title bar to add a new form source.

Form Source

The Form Source zone contains display-only information about a form source. This zone appears when a form source has been broadcast from the Forms Source List or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_FORM_SRCE*.

Setting Up Tender Sources

A *tender source* is created for each form source that interfaces payments with tax forms.

These tender sources will be used on the form batch headers' tender controls.

The tender source also specifies the suspense obligation to which payments should be applied if the tax form's obligation is not yet determined at the time of payment creation - e.g. when the form gets suspended or canceled.

To set up a form source, select **Admin Menu** > **Tender Source**. Refer to existing help on setting up tender sources.

Setting Up Form Upload Staging Types

A form upload staging type is created for a unique combination of form type and form source.

Use the Form Upload Staging Type transaction to view and maintain form upload staging types. To set up a form upload staging type, select **Admin Menu** > **Form Upload Staging Type**.

Form Upload Staging Type Query

Use the *query portal* to search for an existing form upload staging type. Once a form upload staging type is selected, you are brought to the maintenance portal to view and maintain the selected record.

You can click the **Add** link on this portal to add a form upload staging type.

Form Upload Staging Type Portal

This portal appears when a form upload staging type has been selected from the Form Upload Staging Type Query portal or if this portal is opened via drill down from another page.

The topics in this section describe the base-package zones that appear on this portal.

Form Upload Staging Type

The Form Upload Staging Type zone contains display-only information about the form upload staging type.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_FORM_UPLD_STG_TYPE*.

The Big Picture of Form Control

The base product's standard set of form processing batch jobs are configured by default to process all tax and registration forms received. Any form that passes validation is processed through to **Posted** if all the batch processes are run. The batch processes support input parameters to process only forms of certain form type or for a certain batch header ID, but this is considered unusual and not part of standard production nightly processing.

There are some implementations that want to control volumes of tax forms that are posted and prefer to select which tax forms should be posted. In addition, certain implementations may choose to hold off on processing a certain set of tax forms. The product provides an object called Form Control that may be used by implementations that prefer to work this way. The topics in this section provide more information about form control records.

NOTE: The form control functionality does not apply to registration forms, only to tax forms.

How Are Form Controls Created?

The product expects that the form control is created via a web service. The expectation is that analysis is performed using an appropriate reporting or analysis tool and that the tax form IDs are selected in that tool. The reporting / analysis tool can then create a file and an appropriate mechanism like Oracle Service Bus transforms the file and interfaces the information via a web service call. The web service is used to create the new form control and link the selected tax forms.

The product supports automatic creation of form controls for batch-uploaded tax forms. A special batch process can be run after the forms upload and tax form validation processes. This batch process will pick up any ‘ready for posting’ tax forms and create a form control for each distinct form batch header. The form batch header’s included tax forms are then linked to the form control.

The product does not provide any logic for manually defining which tax forms to include in the form control.

The form control type includes configuration to indicate whether the list of IDs provided in a new form control represent internal tax form IDs or rather external references (document locator number or DLN).

The form control record is created in the **Pending** status. The base product attaches a deferred monitor to the pending state so that the validation of the IDs occurs in batch allowing for large volumes. Depending on an implementation's business practice, this deferred monitor may be one to run often during the day.

Once the record is validated, it is routed for approval. Refer to *Form Control Review / Approval* for more information.

Tax Form Configuration for Form Control

The form control functionality expects that the standard base batch processes to upload tax forms and validate the tax forms are still run. The form control relates to tax forms that are in **Ready for Posting** status.

Implementations that choose to use form control processing must adjust the configuration of the **Ready for Posting** status in the parent tax form BO lifecycle. Refer to *Tax Form BO Configuration* for more information.

Form Control Review / Approval

Once the validation of the tax forms linked to the form control has completed a user must review the form control and determine the next steps. The form control type defines the To Do type and To Do role used to alert the appropriate users that a form control is ready for review. Refer to *Form Control Review* for more information on the actions the user may take.

Posting Tax Forms

Once a form control is approved, its tax forms may be posted. The approval step stamps the form control with the batch job that is responsible for posting the tax forms. The batch job is taken from the form control type. The current run number for that batch job is stamped as well. The base product provides a batch job (C1-FCPTF - Process Form Control Tax Forms) that is responsible for finding form controls for the current run number and progressing all the tax forms with an **Active** link status to its next state.

NOTE: If the batch job determines that any of the tax forms are no longer in the appropriate state, the link status is updated to **Skipped** as an audit.

Configuring Form Control Options

If your implementation supports form controls, the topics in this section describe configuration requirements.

In addition, your implementation should read the detailed descriptions for the base product business objects provided for Form Control Type (**C1-FormControlType**) and Form Control (**C1-FormControl**).

Setting Up Form Control Types

A Form Control Type defines the configuration information that is common to form controls of a given type. The type of information captured on the form control type is governed by the form control type's business object.

To set up a Form Control Type, select **Admin Menu** > **Form Control Type**.

The topics in this section describe the base-package zones that appear on the Form Control Type portal.

Form Control Type List

The following functions are available:

- Click a *broadcast* button to open other zones that contain more information about the adjacent form control type.
- Click the **Add** link in the zone's title bar to add a new form control type.

Form Control Type

The Form Control Type zone contains display-only information about a Form Control Type. This zone appears when a Form Control Type has been broadcast from the Form Control Type List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *C1_FORM_CTRL_TYPE*.

Tax Form BO Configuration

For implementations that opt to use form control processing must adjust the configuration of the **Ready for Posting** state of the Parent Tax Form BO (**C1-ParentTaxForm**) as follows so that it does not automatically progress to **Posted**.

- Remove the Monitor Process configured for the status.
- Deactivate the base monitor algorithm. To do this add a BO Status option with an option type of **Inactive Algorithm** and an option value of **F1-AT-RQJ**.

In addition, the following configuration enables the tax form user interface to display information about the form control that a tax form is linked to:

- Identify status values that may be linked to a form control. The suggested values are **Ready for Posting**, **Posted**, **Adjusted**, **Reversed** and **Transferred**.
- For each status identified, add an option type of **Status Category** and an option value of **C1FA** (Related Control Applies).

Form Control Batch Jobs

The following batch jobs must be scheduled as per your implementation's business rules:

- **C1–CREFC** (Create Controls For Uploaded Forms). This batch process creates form controls for batch-uploaded forms. To create form controls using this process, the batch job must be run after all the forms upload processes have run. This background process selects **Ready for Posting** tax forms that reference a Form Batch Header ID and that are not yet linked (as **Active**) to an existing form control. The selected tax forms are grouped by form batch header so that a **Pending** form control can be created for each unique form batch header. All forms related to the form batch header are linked (as **Active**) to the form control.
- **C1-FCMD** (Form Control Monitor (Deferred)). This batch job selects form control records that are **Pending** and validates that the tax forms in the list are valid and in an appropriate state. Once this batch job completes, the processed form controls are ready for approval. Implementations should schedule this based on the business practice of the users that are creating the form controls and the users that are expected to approve the form controls. The assumption is that the approval should be done in a timely manner. So one option is to schedule this job on a frequent schedule, such as every 15 minutes. That way the approvals are processed quickly throughout the day.

NOTE: Email routing. If your implementation plans to use email routing for the "form control requires approval" To Do entry, then the background process **F1-TDERR** (To Do External Routing) must be scheduled after the monitor batch job with the same or similar frequency.

- **C1-FCPTF** (Process Form Control Tax forms). This batch job selects form controls that are stamped with this batch code and the current run number and posts the "active" tax forms. This batch job is creating financial transactions among other objects created by posting a tax form and the assumption is that implementations will run this in their nightly batch run.

Refer to *Batch Process Dependencies* for more information.

Implementing Web Service Options

The product provides the following configuration to support interfacing a list of tax forms to link to a form control via a web service.

The XAI Inbound Service **C1-FormControlUpload** provides the API for the service script **C1-FrmCtrlUp**. This service script includes logic creating a form control and linking a group of tax forms. Note that the script has been designed to be called iteratively for cases where a large file is received and the interface tool that processes the file can process it in chunks.

FASTPATH: Refer to the description of the XAI inbound service and the above service script for more information.

Chapter 8

Defining Forms Rules

Forms processing is a core function for a tax authority. When forms are processed, the system needs to verify that the information is correct and update or create the appropriate taxpayer details and transactions.

Form rules provide business users with the ability to configure processing rules for a form type.

Understanding Form Rule Administration

This section describes concepts and common tasks related form rule administration.

About Form Rules

A form rule is combination of configuration data and processing logic that applies a business rule. The configuration that controls the form rule logic is defined on the form rule business object. The form rule BO has an associated **Apply Rule** algorithm to execute the logic of the rule.

When forms are defined, each form section and line must reference a business name that is unique for that form type. As form changes are introduced, these business names remain constant.

Many form types within a given set of forms also have common information such as taxpayer identification, income and credit lines and so on. The validation and processing rules are typically the same for these common components.

Form rules may be designed to reference section and line business names rather than specific form lines. If a section / line exists in the form from one year to the next, its rules can be reused. Form lines that are common across form types can utilize the same rules provided the business names are the same.

FASTPATH:
Refer to *Designing Your Form Rules and Groups* for information about designing your rules.

About Form Rule Groups

A form rule group is a uniquely named group of rules. You can predefine a set of rules common to a particular form section or set of form lines that can be selected and associated with a form type. For example, predefined rule groups can be used to verify that all taxpayer identification details have been filled in on a form or complete the actions required when a form is posted.

Form rule groups have an associated reference form type. Reference form types are used to determine the section and line business names that may be used by rules in this group to reference the form data. The sections and line names defined in the reference form type must be common to all form types that use this rule group.

Form rule groups are also used to define the form rules for a form type. A form type can define one or more form rule groups to be executed during certain steps in the form lifecycle (defined by a rule event). Sequence numbers are used so that multiple form rule groups executed for the same rule event are done in an appropriate order. Note that the system looks to the specific form type for the form being processed to determine which rules to execute, not the reference form type on the rule group. This means that the rules to be executed can be tailored to the different form types within a broader category of forms.

FASTPATH:
Refer to *The Big Picture of Rule Processing* for more information about processing different rule events.

About Exception Categories

An exception category is a broad categorization of exceptions that can be reported by form validation rules. An exception category must be specified on a form exception.

Understanding Rule Processing

As a form transitions through the states in its lifecycle, various rules need to be executed. The following sections describe the way in which the base system processes rules.

Apply Rules Overview

The following section highlights the logic in the base algorithm type Apply Form Rules (*C1-FRM-APPRL*). It has been designed to call the algorithms plugged into the Apply Rule plug-in spot for each rule linked to the form type for a given rule event. This algorithm type should be plugged into the **Business Object Status - Enter** plug-in spot for each state that requires rules to be executed.

At a high level, the base algorithm follows these steps

- Read the form data and store it in memory
- Find the form rule groups linked to the input form type and rule event. For each group in sequence order, retrieve its **Active** rules in sequence order.
- For each rule,
  - Execute its Apply Rule algorithm
  - Check the returned value of Rule Action. If the action is **Terminate**, terminate overall processing; otherwise continue on to the next rule.

- When processing is complete,
  - Update the form using the data updated in memory by the rule algorithms
  - If exception processing applies (for validation rules) and the exception list is populated, compare the current list of exceptions to the existing exceptions for the form, if any. Any exceptions that are no longer in the current list are closed. Add any newly-reported exceptions to the form exception table.

NOTE:
No updates. The individual apply rule algorithms should not do any updates to the form data in the database for performance reasons. These algorithms simply update the internal form data area and exception information in memory with new values. The Apply Form Rule BO enter plug-in is responsible for updating the form and creating or closing form exceptions.

Validation Rules

The following sections highlight unique functionality for validation type form rules.

Verifying Form Data Using Form Rules

Validation rules determine if the information supplied on the form can be processed. This usually includes checking that required information is supplied and validating that the data is in the correct formats. Validation rules are also used to identify whether the taxpayer related to the form already exists in the system. Validation rules are responsible for reporting any issues on a form that may require correction.

Most forms have a lifecycle that includes a validation step. The base package provides a sample tax form **C1-ParentTaxForm** that has a **Validate** status. This status has an enter-plug in that uses the base *C1-FRM-APPRL* algorithm type to execute rules for the **Validation** rule event and process any exceptions reported.

Reporting Exceptions

Algorithms that perform validation rules are responsible for reporting any issues as a form exception. Form exceptions details must include the form rule group and rule that detected the issue, an exception category and an exception class. The base package includes exception class values of **Notification**, **Waiting for Information** and **Suspense**. Your implementation can add further classes as appropriate for your business rules.

Stopping Form Processing When There Are Issues

A form can suspend if there are errors that a tax authority user needs to review or fix.

If the form is missing key information, the form can go into a 'waiting' state until the tax authority receives the information from the taxpayer.

The exception class attribute can be used to determine what state a form should move to if there are open issues after validation. The base package algorithm types Tax Form - Transition on Exception (*C1-FRE-TROEX*) and Registration Form - Transition on Exception (*C1-PRF-TROEX*) are designed to transition a form to a waiting state if open exceptions of class **Waiting for Information** exist or to a suspended state if there are open exceptions of the class **Suspense**. The algorithms are plugged into the respective base form business objects as **Business Object Status - Enter** plug-ins to the Validate states after the**Apply Form Rules** algorithm.

Executing Validation Rules 'On Demand'

There are cases where a user may be entering data or correcting data in a form and may want to check the form's validity. The product provides a service that can execute validation rules and return the list of possible exceptions for display to a user. This functionality has been implemented in the base Tax Form and Registration Form parent business objects. When the status is one that allows editing of the form, when the user is editing the form a **Check Form** button is available.

FASTPATH: Refer to *Validating Forms* for more information on how the form business objects perform validation. Refer to *Web Self Service Rules* for information on executing on demand validation from web self service applications.

Processing Rules

Any rules executed after the Validate state are considered "processing" rules, meaning the algorithms are responsible for causing something to occur. The following sections highlight unique functionality for processing type form rules.

Processing Form Data Using Form Rules

Once a form's data has been verified, it may be posted into the system. A posted form may subsequently be adjusted to correct information, transferred to another obligation or account or reversed altogether. At each of these steps in the lifecycle, other actions need to take place. The system may need to create taxpayers, accounts and obligations, create financial details, create overpayment processes or possibly reverse the effect of previous updates.

The base package **Apply Form Rules** algorithm type (*C1-FRM-APPRL*) is designed to be used on any state that requires rules processing. To use this algorithm type to execute processing rules, you need to define a rule event applicable to that state and plug in an algorithm configured to execute rules for that event. The base package includes the following pre-configured rule events and associated algorithms: **Suspend, Waiting for Information, Ready for Posting, Post, Cancel, Adjust, Transfer** and **Reverse**. The base package business objects **C1-ParentTaxForm** and **C1-ParentRegistrationForm** include appropriate algorithms for all the listed rule events as Enter plug-ins for the corresponding states.

Processing Rules Create Other Objects

Algorithms that perform processing rules are responsible for any data updates that affect other objects in the system such as accounts, obligations, adjustments and others. As such, they are also responsible for creating form log entries that link those objects to the form being processed. Refer to the base algorithm **Create Assessment From Form Line** (*C1-FR-CRASMT*) for an example of a rule algorithm that creates an assessment adjustment and its associated log entry.

Processing Rules Do Not Report Exceptions

In a typical form lifecycle, the actions that take place when a form is posted, adjusted, transferred or reversed are interrelated. If one of these actions is not successful, an error should be reported and the form should remain in the same state as before the updates took place. For this reason, the system does not support exception handling after processing rules are executed. Any issues that are encountered should be reported using standard error handling.

Web Self Service Rules

Many tax authorities provide web self service applications that allow a taxpayer to file forms online. These applications may require form rules that differ from the rules applied when forms are uploaded and processed in the system. The following sections highlight unique functionality for form rules invoked from web self service.

Form Validation

Implementations may choose to offer the ability for the taxpayer to "Check the Form" as it is being filled in. There are some types of validation that a tax authority may not want to execute when a taxpayer requests on demand validation. The product provides the following support for web self service based validation:

- A rule event flag of **WSS Validate** is provided when configuring form type rules. This allows an implementation to configure the validation rules that are appropriate for checking a form filled in online by a taxpayer.
- A special business service called **Apply WSS Validation Rules** (C1-ApplyWSSValidation) is provided. An integration with a WSS system can call this service passing the form data. The service executes the rules linked to the form type via

the **WSS Validate** rule event and returns any exceptions found. It also returns an indication if any of the form data has been changed as a result of "auto-correct" functionality in any of the rules.

Copy Details From Previous Form

Implementations may choose to offer the ability for the taxpayer to pre-populate a new form based on details from a previous filing of the same form type. The type of information that should be copied may differ for different form types. The product provides the following support for this function:

- A rule event flag of **Copy Previous Form** is provided when configuring form type rules. This allows an implementation to configure the rules that govern the information to be transferred from a previous form of this type.
- A special business service called **Copy From Previous Form** (C1-CopyFromPreviousForm) is provided. An integration with a WSS system can call this service passing the WSS form data. The service executes the rules linked to the form type via the **Copy From Previous Form** rule event and returns an indication if any of the form data has been changed as a result of logic in any of the rules.

Pre-populate Form Details

Implementations may choose to automatically pre-populate a new form based on known information about the taxpayer. A common example is demographic information such as names and addresses. The type of information that should be retrieved may differ for different form types. The product provides the following support for this function:

- A rule event flag of **Pre-Populate Form** is provided when configuring form type rules. This allows an implementation to configure the rules that govern the taxpayer information to be retrieved and populated in a form of this type.
- A special business service called **Pre-populate Taxpayer Information** (C1-GetTaxpayerInfoForm) is provided. An integration with a WSS system can call this service passing the WSS form data. The service executes the rules linked to the form type via the **Pre-Populate Form** rule event and returns any exceptions found. It also returns an indication if any of the form data has been changed as a result of logic in any of the rules.

Pre-Processing Rules

Each tax form defines different information which may be used to identify a taxpayer, account and tax role. The form rule infrastructure allows an implementation to configure the appropriate rules to determine and capture these key IDs on the form. These types of rules are typically executed as part of form validation.

For reasons of efficiency and scalability, it is preferable for some of this key information to be identified before the form is created in the system. The product provides the following support to allow the same rules to be used in both situations:

- A rule event flag of **Pre-processing** is provided when configuring tax form type rules. This allows an implementation to configure the same rules used to identify taxpayer information during form validation for the pre-processing step.
- The base package **Tax Form — Apply Pre-processing Rules** algorithm type (*C1-TF-APRERL*) is provided to apply the rules plugged in on the pre-processing event. The base package business object **C1-ParentTaxForm** is configured with this algorithm. Refer to the algorithm type description for more information on how these rules are applied during pre-processing.

Setting Up Exception Categories

When a form's data is validated the system may identify errors which it stores as exceptions. You can use exception categories to group these exceptions. The exceptions for the forms can then be viewed on the Exceptions tab of the form's maintenance portal. To maintain an existing Exception Category, select **Admin Menu** > **Exception Category**.

The topics in this section describe the base-package zones that appear on the Exception Category portal.

Exception Category List

The Exception Category *List zone* lists every Exception Category. The following functions are available:

Click the *broadcast* icon to open other zones that contain more information about the adjacent Exception Category.

The standard actions of **Edit** and **Delete**are available for each Exception Category.

Click the **Add** link in the zone's title bar to add a new Exception Category.

Exception Category

The Exception Category zone contains display-only information about an Exception Category. This zone appears when an Exception Category has been broadcast from the Exception Category List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Setting Up Form Rule Groups

Form rules are logically grouped together in a form rule group. The rules are executed in the following order: Each rule group linked to the form type is executed in sequential order. The rules in each group are executed in sequential order.

To set up a form rule group, open **Admin Menu** > **Form Rule Group**.

Form Rule Group Query

Use the query portal to search for an existing form rule group. Once a form rule group is selected, you are brought to the maintenance portal to view and maintain the selected record.

You can also click the Add link in the query zone's title bar to add a new form rule group.

Form Rule Group Portal

This portal appears when a form rule has been selected from the Form Rule Group Search results.

The topics in this section describe the base-package zones that appear on this portal.

Actions

This is a standard actions zone. The **Edit, Delete** and **Duplicate** actions are available.

Form Rule Group

The Form Rule Group zone contains display-only information about a form rule group. This zone appears when a form rule group has been selected from the Form Rule Group Search results, or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Follow this link to open the data dictionary where you can view the tables that reference *CI_FORM_RULE_GRP*.

Form Rules List

The Form Rule *List zone* provides a summarized view of all form rules for a form rule group. A row is displayed for every form rule linked to the form rule group. The rules are shown in sequential order. The list includes the form rule information and a hyperlink to the rule's maintenance portal.

The standard actions of **Edit, Delete** and **Duplicate** are available for each form rule.

Click the **Add** link in the zone's title bar to add a new form rule.

Form Types

This zone displays all form types that reference the form rule group. This zone only appears if a form rule group exists in the portal context. The list includes the name of the form type and a link to the form type's maintenance portal.

Setting Up Form Rules

A form rule is designed to implement one specific rule used in processing a form. Examples of form rules include taxpayer existence rules, form line validation, calculation and auto-correction rules, tolerance checking, and posting logic.

To set up a form rule, open **Admin Menu** > **Form Rule**.

Form Rule Query

Use the query portal to search for an existing form rule. Once a form rule is selected, you are brought to the maintenance portal to view and maintain the selected record.

You can also click the Add link in the query zone's title bar to add a new form rule.

Form Rule Portal

This portal appears when a form rule has been selected from the Form Rule Search results.

NOTE: When the Form Rule portal is used the context-sensitive Rules on Group zone appears in the dashboard. This zone lists the other form rules in the same group as the rule in context. From this dashboard zone you can add a new form rule to the group, or go to the maintenance portal for the other rules in the group

The topics in this section describe the base-package zones that appear on the Form Rule portal.

Form Rule

The Form Rule zone contains display-only information about a form rule. This zone appears when a form rule has been selected from the Form Rule Search results, or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Follow this link to open the data dictionary where you can view the tables that reference *CI_FORM_RULE*.

Designing Your Form Rules and Groups

The base product provides a generic BO for Form Rule Group, **C1-FormRuleGroup**. Your implementation can add additional business rules to this BO as required.

The base product provides a parent BO for Form Rule, **C1-FormRule**. This BO defines the common elements for all form rules. It is not intended to be used stand-alone. Your implementation can include this BO in each rule BO and add the additional components as required.

The base product supplies a number of BOs that apply particular Form Rules. Refer to the business objects that reference the Form Rule maintenance object **C1-FORMRULE** for a complete list.

Your implementation can define additional form rule BOs as required. The following points highlight the important configuration for this business object:

- Create a new business object and include the **C1-FormRule** business object.
- Develop the appropriate rule processing algorithm to perform the rule logic.
- For any configuration required by the rule processing algorithm, consider whether it should be defined when configuring the form rule by a business user. If so, include the appropriate elements in the form rule business object's schema.
- For a rule that performs validation, consider whether the rule exception information, which includes the Exception Category should be defined when configuring the form rule by a business user. If so, include the appropriate elements in the form rule business object's schema. The base product provides the data area (**C1-ExceptionInformation**) that can simply be included in the business object schema.
- For the user interface, the business object **C1-FormRule** and the data area **C1-ExceptionInformation** both support UI Hints. Many base form rule business objects also support UI hints. However, some still use full UI maps. When creating a new custom form rule, it is recommended to use UI Hints for the new business object. For an example of a base form rule business object that uses UI Hints and includes functionality for referencing a form section and line business name, refer to the Create Customer Contact business object (**C1-CreateCustomerContact**).

  NOTE: See below for information about base UI map fragments that are available for creating a full UI Map for display or maintenance of a form rule.

- Consider what logical grouping this rule belongs to. Note that a group of rules is always executed in the context of a single rule event. Rules should be grouped according to common actions that make take place for a particular event or set of events.

The following UI map fragments are available for use in creating full UI maps for form rule display and maintenance. However, with the use of UI Hints, it is not expected that they are needed.

- Map fragment to display the common elements for all form rules along with the actions: **C1-FormRuleDisplay**
- Map fragment to display the common exception details (for validation rules only): **C1-FormRuleExceptionDisplay**
- Common display service script to plug in on the rule BO to retrieve the form rule details: **C1-FRGenDisp**
- Map fragment to maintain the common elements for all form rules: **C1-FormRuleMaint**
- Map fragment to maintain the common exception details (for validation rules only): **C1-FormRuleExceptionMaint**

The topics in this section describe some of the base rules that may be used by your implementation. Click *here* for a complete list of the apply rule algorithms provided by the system. A corresponding business object exists for each one.

Using Conditional Validation Rules

The base product provides a form rule that performs validation based on form line configuration. The rule Form Line Configuration Validation (**C1-ChkReqSectionsLines**) checks for form sections, groups and lines that are marked as required. It also checks for valid values in form lines configured as foreign key references, lookups or extendable lookups. However, there are many business rules with validation that is conditional based on the values of other form lines in the form. The base product provides a Form Rule BO that is designed handle a large range of conditional validation, **C1-CondElementValidation**.

The **Conditional Element Validation** BO allows a business user to define a series of conditions that apply to sections and lines of a form. The conditions are defined as basic mathematical expressions that have a result of true or false. Your implementation should consider using this rule BO when defining a rule that validates a form line value in relation to other supporting lines on the form.

The following topics highlight some of the available features for conditional expressions. Other mathematical operators and functions are supported but these are not normally used for calculations that are common to forms validation.

Variables

Variable identifiers must be a character between a-z except x. Variable x is always used to refer to the form line that this rule applies to. Variable identifiers are case sensitive.

Operators

The following types of expression operators are supported.

Mathematical Operators	Operators of *, /, %, +, - are allowed. Examples of expression using these operators are: • **x** = **a** + **b** + **c** where the value of **x** should equal the sum of the values **a**, **b** and **c** • **x** = **a** * **r** where **r** is a rate factor variable
Comparison Operators	Operators of =, != (not equal), <, >, <= and >= are allowed. Examples of expression using these operators are: • **x** > **a** where the value of **x** should be greater than the value of **a** • **x** != 0 where **x** must have a non-zero value
Grouping Operators	Parentheses may be used to formulate expressions with sub-computations. An example of an expression using grouping is: • **x** = (**a** + **b**) / **c** where the value of **x** should be equal to the result of adding values **a** and **b** then dividing by the value of **c**

Functions

The following examples show the use of the most common supported functions:

Ceiling	**x** = ceiling (**a*** **b**) where **x** should equal the result of multiplying the value of **a** by the value of **b** and rounding up to the nearest whole number
Floor	**x** = floor (**a*** **b**) where **x** should equal the result of multiplying the value of **a** by the value of **b** and rounding down to the nearest whole number

Round	**x** = round (**a*** **b**) where **x** should equal the result of multiplying the value of **a** by the value of **b** and rounding the nearest whole number. A result of 1.5 would round to 2 and a result of 1.4 would round to 1. To round to a decimal place, use **x** = round (**a*** **b**, **n**) where **n** is the number of decimal places. The default rounding behavior is to round .5 up (ROUND_HALF_UP). The syntax supports overriding this behavior, for example: **x** = round (**a*** **b**, **2**, **'ROUND_HALF_DOWN'**). The rounding modes supported are based on those supported for the BigDecimal java class.
Any	any [value in **a** | value !="] where **a** is a value from a repeating line and the condition is true if any of the values of **a** are not blank
All	all [value in **a** | value !="] where **a** is a value from a repeating line and the condition is true if none of the values of **a** are blank

Conditional Sections and Groups

The base product provides another form rule that performs conditional validation for Sections and Lines. The rule Check Conditionally Required Sections and Groups (**C1-CheckCondReqdSectionGroup**) checks whether or not a form section or group are required or must be empty based on a defined set of conditions. This rule can be used in conjunction with the **Conditional Element Validation** rule described above.

FASTPATH: Refer to the Apply Rule algorithm (*C1-CRQ-APRL*) description for more information.

Using the OPA Validation Form Rule

The base product provides a Form Rule BO that is designed to handle a large range of conditional validation, **C1–OpaValidation**.

The BO allows a business user to specify the OPA rulebase that implements validations and calculations that apply to the form data. The form rule also defines how to interpret the outcome of the OPA rulebase invocation. You should consider using this rule BO when significant part of form data validations is suitable to be implemented using OPA. In this scenario, your implementation should generate an OPA rulebase data model for the Form Type and develop the actual OPA rulebase prior to configuring the form rule.

FASTPATH: Refer to *Generating an OPA Rulebase Data Model for a Form Type* for specific details about data model generation. Refer to *Oracle Policy Automation Integration* for more details about OPA.

Rule Details

The search for the OPA Rulebase name derives the results from the OPA instance that participates in the integration. Make sure your rulebase is deployed on the appropriate determinations server.

Rule Outcome Details

Your implementation can define fine-grained rule outcome **interpretation**. The rulebase output includes two numeric indicators: error severity and rule outcome. You may associate the values of these indicators with appropriate **Exception Classes** and **Rule Actions**. The configuration effort should be tightly coordinated with the rulebase author.

NOTE: We recommend establishing a set of values for **Error Severity** and **Rule Outcome** indicators and using them for the OPA-based form validation. Then use OPA Integration Configuration to define a default rule outcome mapping. These default configurations are copied to the newly created Form Rule and can be edited afterwards..

FASTPATH: Refer to *Configuring the Shared Data Model Integration* for the OPA integration configuration details.

Exception Information

The outcome of the rule can vary because the rulebase performs complicated business logic and multiple validations. The rule may also report multiple exceptions. Nevertheless your implementation can define a single default **Exception Class** and **Rule Action**.

Additional Notes on OPA Rulebase

Implement complicated tax policies and laws. OPA provides a platform for turning the laws and instructions into executable modules. Business analyst may use an original tax documents as a source for the policy and implement it. A majority of validations and calculations that can be performed on the actual form data, without additional database interactions, could be implemented with OPA.

Validate the entire form with OPA. The entire form is passed to the OPA Rule and it is expected to perform multiple validations and calculations and return the modified form and the set of exceptions. OPA webservice call is relatively expensive performance-wise, therefore it make sense to consolidate as much business logic as possible into one OPA rulebase.

Writing OPA Rulebase with minimal knowledge of the base product. The base product provides the integration mechanism that supports the data exchange with OPA. Thus, the OPA Rulebase author works exclusively on the OPA module.

Multiple OPA Rulebases can be written with same data model. The data model is generated once and handed over to the OPA Rule author(s). It can be used in various OPA Rules linked to the same form type.

Share OPA Rulebases across the enterprise. The integration with PSRM Self Service supports import of the form type definition and it uses OPA as a form validation engine. The rulebase data model can be generated in either product for the same form type, and the OPA rulebase based on this model can be used by both products.

Base Rules that Determine Master Data for Tax Forms

The base product provides several rules that attempt to determine master data that may already exist in the system for a tax form being filed. This section describes the functionality of these rules at a high level and highlights any potential dependency between rules. In every case the business object is provided along with a link to the Apply Rule algorithm where the detailed description provides more details. Note that some rules may also be used for registration forms. Refer to the separate registration form section for details. In all cases, the rules are expected to be plugged in as a **validation** rule event.

- Determine Tax Form Filing Period (**C1-DetermineTaxFormFilingPer**). This rule determines the filing period for the tax form using valid filing calendars on the tax type or valid filing periods on the form type along with form dates to identify the appropriate filing period. The filing period is a key piece of information for further validations of the tax form and the taxpayer. This rule or a similar one that works based on the business rules of the particular tax form type should be one of the early validation rules.

  FASTPATH: Refer to the Apply Rule algorithm (*C1-FR-DETFPE*) description for more information.

- Check Taxpayer Existence (**C1-CheckTaxpayerExistence**). This rule uses the person ID and optionally the taxpayer name to identify a taxpayer (person). If the form type supports a secondary taxpayer, the rule supports secondary taxpayer identification as well. If no taxpayer is identified, it issues an error if the tax type indicates that pre-registration is required. Otherwise, the assumption is that the taxpayer will be created at posting time. Note that if a form uses a tax role external ID and does not supply any other taxpayer identification, then this rule is not needed. The Check Pre-registered Account and Tax Role rule can also populate the taxpayer's Person ID based on the tax role.

FASTPATH: Refer to the Apply Rule algorithm (*C1-FR-CHTXEX*) description for more information.

- Account and Tax Role Determination rules. The product provides more than one rule that attempts to determine the account and tax role based on different types of data that may be present on the form.
    - Check Pre-registered Account and Tax Role (**C1-CheckPreregTaxRoleAccount**). This rule is used for tax types that require pre-registration before filing a tax form. It also caters for forms that use a Tax Role External ID as a key identifier. It supports situations where the form does not include other fields for identifying the taxpayer. It also caters for situations where the filing period has not been identified by a previous rule. This rule can populate the filing period based on the determined tax role.

      FASTPATH: Refer to the Apply Rule algorithm (*C1-FR-PRTRAC*) description for more information.

    - Check Account and Tax Role Existence (**C1-AccountTaxRoleCheck**). This rule caters for tax types that may not require pre-registration. It assumes that the filing period for the form has already been identified.
        - If the taxpayer id is not populated, it does not process further. The assumption in this case is that the form allows "auto-registration" and that the person, account and tax role will be added later. Note that it does not check for pre-registration in this scenario because it assumes that a rule that has attempted to identify a taxpayer has already issued an exception for pre-registration tax types.
        - If the taxpayer is populated, it attempts to find the account and tax role for the form. In each case, if more than one is found based on the rules, an exception is created. If no record is found, then it only issues an error for tax types where pre-registration is required.

      FASTPATH: Refer to the Apply Rule algorithm (*C1-ATR-EXIST*) description for more information.

- Check Filing Period Obligation (**C1-CheckFilingObligationExist**). This rule tries to find an existing non-canceled obligation for the account, filing period (filing calendar and end date) and tax role identified for the form. If an appropriate record is found, the ID is populated into the record. If no records are found, no exception is created. The assumption is that even for pre-registered tax types, it is valid for the obligation to have not yet been created when the tax form is received.

Base Rules that Determine Master Data for Registration Forms

The base product provides rules that attempt to determine master data that may already exist in the system for a registration form being filed. This section describes the functionality of these rules at a high level. The business object is provided along with a link to the Apply Rule algorithm where the detailed description provides more details. In all cases, the rules are expected to be plugged in as a **validation** rule event.

- Check Taxpayer Existence (**C1-CheckTaxpayerExistence**). This rule uses the person ID and optionally the taxpayer name to determine if the taxpayer (person) is already known in the system (for updates to existing registrations). No error is issued if no taxpayer is found. The assumption is that the taxpayer will be created at posting time.

  FASTPATH: Refer to the Apply Rule algorithm (*C1-FR-CHTXEX*) description for more information.

Base Rules that Create Master Data for Tax Forms

The base product provides several rules that create key master data such as person, account and tax role. These are all **Posting** rules that should be executed when transitioning the form to posted after it has been fully validated. This section describes the functionality of these rules at a high level and highlights any potential dependency between rules. In every case the business object is provided along with a link to the Apply Rule algorithm where the detailed description provides more details. Note that some rules may also be used for registration forms. Refer to the separate registration form section for details.

- Create Taxpayer (**C1-CreateTaxpayerFromForm**). This rule checks whether the form already indicates the Person ID for the primary taxpayer. If not, it creates a person using appropriate information from the form. If the form includes secondary taxpayer information and does not have a Person ID for the secondary taxpayer, it also creates a person for the secondary taxpayer.

  FASTPATH: Refer to the Apply Rule algorithm (*C1-TXP-CRE*) description for more information.

- Create Account (**C1-CreateAccount**). This rule checks whether the form already indicates the Account ID. If not, it creates an account using the primary taxpayer as the main taxpayer for the account. If the form also identifies a secondary taxpayer, that person is linked as an additional financially responsible person for the account.

  FASTPATH: Refer to the Apply Rule algorithm (*C1-CREACCTFF*) description for more information.

- Create Tax Role (**C1-CreateTaxRoleFromTaxForm**). This rule is provided to create a tax role for the current tax form being filed using the tax form's account and filing calendar, the tax form type's tax type. The rule caters for a tax role start date to be defined on the form. The rule creates a tax role with limited information and expects to determine the tax type based on the tax form type. It is not available for creating tax roles from registration forms.

  FASTPATH: Refer to the Apply Rule algorithm (*C1-FR-CTRFTF*) description for more information.

- Create Filing Period Obligation (**C1-CreateFPObligationFormRule**) This rule creates an obligation for the tax form if one has not already been identified. It assumes that the Account and Tax Role for the form are already identified or have been created. It also assumes that the form's Filing Period has been identified.

  FASTPATH: Refer to the Apply Rule algorithm (*C1-CRE-FPOBL*) description for more information.

Base Rules that Create Master Data for Registration Forms

The base product provides several rules that create key master data such as person and account for a registration form. These are all **Posting** rules that should be executed when transitioning the form to posted after it has been fully validated. This section describes the functionality of these rules at a high level and highlights any potential dependency between rules. In every case the business object is provided along with a link to the Apply Rule algorithm where the detailed description provides more details.

- Create Taxpayer (**C1-CreateTaxpayerFromForm**). This rule checks whether the form already indicates the Person ID for the primary taxpayer. If not, it creates a person using appropriate information from the form.

  FASTPATH: Refer to the Apply Rule algorithm (*C1-TXP-CRE*) description for more information.

- Create Account (**C1-CreateAccount**). This rule checks whether the form already indicates the Account ID. If not, it creates an account using the primary taxpayer as the main taxpayer for the account.

  FASTPATH: Refer to the Apply Rule algorithm (*C1-CREACCTFF*) description for more information.

Base Rules that Pre-populate Form Data

The base product provides a number of rules that attempt to pre-populate a new form with data that already exists in the system for the taxpayer.

These rules are designed to support forms created by online interaction with external applications such as web self service. They are invoked as a stand alone action and return updated form data. They are expected to be plugged in on specific rule events that match those actions.

NOTE: The product provides business services to support invoking these rules as a stand alone process. Refer to *Web Self Service Rules* for more information.

This section describes the functionality of these rules at a high level. In every case the business object is provided along with a link to the Apply Rule algorithm where the detailed description provides more information.

- Copy Previous Tax Form (**C1-CopyPreviousTaxForm**). This rule attempts to find a previously posted tax form linked to a given taxpayer ID and copies specific lines from that form to the current form. The rule searches for forms for which the taxpayer was either the primary or secondary filer. If more than one form is found, the most recently posted form is used. This rule should be configured for the **copy previous form** rule event.

  FASTPATH: Refer to the Apply Rule algorithm (*C1-CPYTXFORM*) description for more information.

- Copy Previous Registration Form (**C1-CopyPreviousRegForm**). This rule attempts to find a previously posted registration form linked to a given taxpayer ID and copies specific lines from that form to the current form. If more than one form is found, the most recently posted form is used. This rule should be configured for the **copy previous form** rule event.

  FASTPATH: Refer to the Apply Rule algorithm (*C1-CPYRGFORM*) description for more information.

- Pre-populate Demographic Data (**C1-PrepopulateDemographicInfo**). This rule retrieves name and address details for a taxpayer and transfers them to the current form being processed. It should be configured for the **pre-populate form** rule event.

  FASTPATH: Refer to the Apply Rule algorithm (*C1-PRPFORM*) description for more information.

Payment Related Rules

The base product provides several rules to manage payments related to a tax form. This section describes the functionality provided for processing payments with forms.

Payments and Forms Overview

The rules provided by the product cater for different ways of processing payments.

- Payment may be received before the form is processed. This may occur when an annual filer is required to pay quarterly estimated tax payments during the filing year prior to filing. This may also occur when a payment is received with the form but is separated for processing and is posted prior to processing and posting the form. The system provides payment transfer rules that attempt to identify existing payments for the taxpayer and link them to the form.
- Many implementations that receive payments with the tax form do not separate the payment for processing but rather create the payment as part of the form processing. The base product provides the posting rule Create Payment to perform this logic. Creating the payment at posting time satisfies most scenarios. However, what happens when the form suspends or must wait for information or gets canceled prior to posting? Payments received with forms must be immediately captured in the system, regardless of what happens to the form. If a form suspends or waits for information or gets canceled, the associated payment should be created in the system anyway. If the obligation is already identified when the form suspends or waits, the payment will be applied to the obligation. Otherwise, the payment will be created for either an excess credit obligation (if the system is configured to do so and an account is identified on the form) or the

suspense obligation (from the related tender controls' tender source). When the form subsequently posts, the suspended payment will be transferred to the obligation identified on the form.

Payment Rule Details

The following points describe the base payment rules at a high level.

- Create Payment (**C1-CreatePayment**). This rule creates a frozen payment for a tax form when the form indicates a payment amount. It should be plugged in for the **posting**, **suspense**, **waiting for information** and **canceled** rule events. As described in the overview, the rule's algorithm posts the payment to an excess credit obligation or a suspense obligation if the form's obligation or account has not been determined.

  NOTE: Because a payment may be created before the form is posted, the rule checks whether a payment exists for a form before creating one. If a payment has been created via prior form processing but is linked to an excess credit obligation or to a suspense obligation, the related rule Transfer Payment to Form's Obligation is responsible for moving the payment appropriately. If a payment has been created outside of form processing that should be redirected to the form, the related rule Transfer Payment Using Form References is responsible for identifying and moving the payment. The transfer rules should be configured in sequence ahead of the Create Payment rule on the **posting** rule event.

  FASTPATH: Refer to the Apply Rule algorithm (*C1-CPAPRL*) description for more information.

- Transfer Payments to Form’s Obligation (**C1-TransferPmtToFormObligation**). This rule is provided to transfer an existing payment currently posted to either an excess credit obligation or a suspense obligation to the obligation that is identified on the tax form. This rule should be plugged into the **posting** rule event. This rule is designed to redirect payments that fall into the following categories:
  - Payments that were created when the form suspended or waited for information.
  - Payments from a previous ‘transfer to’ form that were transferred to a suspense obligation.
  - Payments that were processed before the form was processed using a set of distribution rules referencing document locator number, taxpayer ID, tax type and revenue period.

  NOTE: The logic to transfer payments processed before the form is optional and is only executed if the related distribution rule characteristic types are populated on the rule. Since the logic is tightly linked to a specific distribution rule configuration, it is recommended that those rule details be left blank and the ‘Transfer Payments Using Form References’ rule be used to identify and move payments processed outside form processing.

  FASTPATH: Refer to the Apply Rule algorithm (*C1-TPFO-APRL*) description for more information.

- Transfer Payments Using Form References (**C1-TransferPmtUsingFormRef**). This rule is provided to transfer an existing payment that was created prior to form processing and is currently posted to either an excess credit obligation or a suspense obligation to the obligation that is identified on the tax form. This rule should be plugged into the **posting** rule event. The rule’s algorithm is designed to identify payments that should be redirected to the form by matching characteristic values on the payments to payment reference details on the form. The rule supports the following sources of payment reference information on the form:
  - Document Locator Number
  - Taxpayer ID Number
  - Tax Type
  - Revenue Period
  - Form Line

NOTE: This rule is recommended for transferring payments created outside form processing. It provides a more flexible means of identifying related payments than the Transfer Payments to Form's Obligation rule and is designed to work with payments that have been created via the payment event upload process with the related payment references attached. Refer to *Interfacing Payments Using Distribution Rules* for more information on payment event upload.

FASTPATH: Refer to the Apply Rule algorithm (*C1-TPFR-APRL*) description for more information.

- Move Payment to Suspense (**C1-TransferPmtToSuspense**). This rule is provided to transfer any existing payments for tax forms to a suspense obligation. This rule should be plugged in as **transfer** and **canceled** rule events. Note that the rule is needed in the canceled rule event to cater for forms that may have gone into suspense or waiting for information where the account or obligation had already been known.

FASTPATH: Refer to the Apply Rule algorithm (*C1-TPS-APRL*) description for more information.

Invoking a System Service

The base product provides a form rule that allows you to invoke a system service. The rule C1-InvokeSystemService (**C1-InvokeSystemService**) can invoke either a Service Script, Business Service or Business Object.

The input for the system service can come from the form lines (including collections of form lines) or constant values mapped to the various input elements of the service. The output from the service can be mapped back to form lines.

FASTPATH: Refer to the Apply Rule algorithm (*C1-ISS-APRL*) description for more information.

Chapter 9

Defining Calculation Engine Options

Revenue authorities need to perform a variety of calculations as part of core business processing. A primary example is calculating billable amounts based on the assessed value of assets.

The calculation engine provide business users with the ability to define a wide variety of calculations using a combination of calculation rules and eligibility criteria.

Understanding Calculation Rule Administration

This section describes concepts and common tasks related to calculation rule administration.

About The Calculation Engine

The calculation engine uses a defined calculation processing interface designed to support calculations for a wide variety of purposes. While it is not confined to calculations using in billing, it is a primary tool for calculating bills and the interface closely mirrors the bill data structure.

The key elements in the calculation processor API are:

- A collection of value details that can be pre-populated and added to by calculation rules. These value details form the basis of many computations performed by the rules.
- A collection of calculation lines which define calculated amounts and other attributes that control how these amounts contribute to the calculation. Calculation lines are the primary results of the computations performed by the rules

FASTPATH:
Refer to *Designing Your Calculation Rules and Groups* for information about the calculation processor interface and further details for designing your rules.

About Calculation Rules

A calculation rule is combination of configuration data and processing logic that applies a rule to compute a value. The configuration that controls the calculation rule logic is defined on the calculation rule business object. The calculation rule BO has an associated **Apply Calculation Rule** algorithm to execute the logic of the rule.

. The most common form of calculation rule will:

- Read from the calculation processor data accumulated by prior rules in order to perform a specific computation, with additional input from external values or factors
- Take the results of the computation and add new calculation lines or value details to the calculation processor data according to instructions in the rule's configuration

A calculation rule may have associated eligibility criteria that control the conditions under which the rule is applied.

FASTPATH:
Refer to *Designing Your Calculation Rules and Groups* for information about designing your rules.

Refer to *About Eligibility Criteria* for information about configuring eligibility criteria.

About Eligibility Criteria

An eligibility criteria rule is a combination of configuration data and processing logic that evaluates conditions to determine if a calculation rule should be applied or not. The configuration that controls the eligibility logic is defined on the eligibility criteria business object. The eligibility criteria BO has an associated **Apply Calculation Rule Eligibility** algorithm to execute the logic of the evaluation.

The eligibility criteria rule algorithm is designed to indicate whether the criteria evaluate to **True**, **False** or **Insufficient Data**. A result of **Insufficient Data** indicates that some or all of information required to evaluate the criteria was not available.

The eligibility criteria BO includes configuration to indicate what action to take for each of the possible evaluation results. The available actions are **Apply Rule**, **Do Not Apply Rule**, **Check Next Condition** or **Terminate Calculation**. A result of **Insufficient Data** has an additional action option of **Error** which will cause the calculation processor to terminate calculation and report an exception.

FASTPATH:
Refer to *Designing Your Calculation Rules and Groups* for information about designing eligibility criteria for your rules.

About Calculation Line Category Types and Values

Calculation line category types are used to establish connections between calculation rules. Calculation rules can specify a member category and value to which they belong. When this same category and value are referenced as a 'Target Category' by other rules, the rule logic will reference the calculation lines created by rules that belong to the category and value.

For example, calculation rules that compute amounts based on asset value that are distributed to the applicable city for the asset may have a value of 'Assessable Value', 'City Revenue'. A subsequent calculation rule that summarizes the overall revenue to be disbursed to the city will reference category line type and value combination as a target category.

About Calculation Groups

A calculation group is a uniquely named group of rules. You can define a calculation group with a set of rules that represent a specific version of a calculation. That calculation group can then be linked to a calculation control together with the date on which that version of the calculation comes into effect.

FASTPATH:
Refer to *The Big Picture of Calculation Rule Processing* for more information about processing calculation rule groups.

About Calculation Controls

A calculation control defines a calculation for a specific purpose. They are used to link system objects to applicable calculation rules. For example, asset based tax types may define valid billing calculation controls which are in turn assigned to the associated tax roles for that tax type. The billing process can then reference the tax role’s calculation control to apply the correct set of calculation rules when generating bills.

A calculation control maintains an effective-dated link to a calculation group that defines a calculation control version. New versions are created when something about the rules changes and prior versions need to be kept for historical recalculations.

The Big Picture of Calculation Rule Processing

When a process in the system needs to perform a calculation, it invokes the Calculation Control processor, passing the appropriate version of the calculation control. This process invokes the Calculation Group processor which is responsible for executing the calculation rules.

At a high level, the Calculation Group processor follows these steps for each calculation rule in sequence for the calculation group

- Fetch each of the rule’s Eligibility Criteria and execute its Apply Criteria algorithm
- Check the returned action value,
  - If the action is **Terminate** or **Error**, terminate overall processing
  - If the action is **Check Next Condition**, continue on to the next set of criteria
  - If the action is **Do Not Apply Rule**, continue on to the next calculation rule
  - If the action is **Apply Rule**, execute the Apply Calculation Rule algorithm for the rule
- After applying a calculation rule, check the rule action flag. If the action is **Terminate** , terminate overall processing

When errors occur in processing, it is usually necessary to retain the interim calculation details in order to diagnose the problem . The calculation group processor handles errors in two ways:

- Calculation rules and eligibility criteria have the option of recording an exception in the calculation processor data error instead of issuing standard errors.
- Errors issued by lower level logic are trapped by the Calculation Control processor and converted to an exception message in the output data.

In both circumstances, higher level logic is responsible for detecting and reporting calculation exceptions as appropriate.

Setting Up Calculation Groups

Calculation rules are logically grouped together in a calculation group. The rules in each group are executed in sequential order.

To set up a calculation group, open **Admin Menu** > **Calculation Group**.

Calculation Group Query

Use the query portal to search for an existing calculation group. Once a calculation group is selected, you are brought to the maintenance portal to view and maintain the selected record.

You can also click the Add link in the query zone's title bar to add a new calculation group.

Calculation Group Portal

This portal appears when a calculation group has been selected from the Calculation Group Search results.

The topics in this section describe the base-package zones that appear on this portal.

Calculation Group

The Calculation Group zone contains display-only information about a calculation group. This zone appears when a calculation group has been selected from the Calculation Group Search results, or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *C1_CALC_GRP*.

Calculation Rules List

The Calculation Rule *List zone* provides a summarized view of all calculation rules for a calculation group. A row is displayed for every calculation rule linked to the calculation group. The rules are shown in sequential order. The list includes the calculation rule information and a hyperlink to the rule's maintenance portal.

Click the **Add** link in the zone's title bar to add a new calculation rule.

Click the **Resequence Rules** link in the zone's title bar to change the execution sequence of the calculation rules within the group.

Calculation Control List

The Calculation Control *List zone* provides a summarized view of all calculation controls that reference this calculation group. A row is displayed for every calculation control linked to the calculation group as a calculation control version. The control versions are shown in effective date / calculation control order. The list includes the calculation control information and a hyperlink to the control’s maintenance portal.

Setting Up Calculation Rules

A calculation rule is designed to implement one specific rule that creates one or more calculation lines or details that comprise part of an overall calculation. Examples of calculation rules include rules that apply a rate to an asset value to calculate a specific charge, rules that create billable charges from externally supplied values, and rules that summarize other calculation lines.

To set up a calculation rule, open **Admin Menu** > **Calculation Rule**.

Calculation Rule Query

Use the query portal to search for an existing calculation rule. Once a calculation rule is selected, you are brought to the maintenance portal to view and maintain the selected record.

You can also click the Add link in the query zone's title bar to add a new calculation rule.

Calculation Rule Portal

This portal appears when a calculation rule has been selected from the Calculation Rule Search results.

The topics in this section describe the base-package zones that appear on this portal.

Calculation Rule

The Calculation Rule zone contains display-only information about a calculation rule. This zone appears when a calculation rule has been selected from the Calculation Rule Search results, or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *C1_CALC_RULE* .

Eligibility Criteria List

The Eligibility Criteria *List zone* provides a summarized view of all eligibility criteria for a calculation rule. Calculation rule eligibility criteria are user-definable conditions that could cause a given calculation rule to be applied or skipped. A row is displayed for every set of eligibility criteria linked to the calculation rule. The criteria are shown in sequential order. The list includes the eligibility criteria information and a hyperlink to the Eligibility Criteria maintenance portal.

Click the **Add** link in the zone's title bar to add a new set of eligibility criteria.

Calculation Rule Eligibility Criteria Portal

This portal appears when a set of calculation rule eligibility criteria has been selected from the Calculation Rule Eligibility Criteria list zone.

The topics in this section describe the base-package zones that appear on this portal.

Calculation Rule Eligibility Criteria

The Calculation Rule Eligibility Criteria zone contains display-only information about a set of eligibility criteria. This zone appears via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *C1_CALC_RULE_CRT* .

Setting Up Calculation Line Categories

Some calculation rules may reference the results of previous rules in their calculation logic. You can define calculation line category types and values to categorize calculation rules. Rules can then be configured as members of a specific category so their resulting calculation lines can be easily identified by subsequent rules. To maintain an existing Calculation Line Category Type, select **Admin Menu** > **Calculation Line Category Type**.

The topics in this section describe the base-package zones that appear on the Calculation Line Category Type portal.

Calculation Line Category Type List

The Calculation Line Category Type *List zone* lists every Calculation Line Category Type. The following functions are available:

Click the *broadcast* icon to open other zones that contain more information about the adjacent Calculation Line Category Type.

The standard actions of **Edit**, **Duplicate** and **Delete** are available for each Calculation Line Category Type.

Click the **Add** link in the zone's title bar to add a new Calculation Line Category Type and Category Values.

Calculation Line Category Type

The Calculation Line Category Type zone contains display-only information about a Calculation Line Category Type and its Category Values. This zone appears when an Calculation Line Category Type has been broadcast from the Calculation Line Category Type List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *C1_CL_CAT_TYPE*.

Setting Up Calculation Controls

A calculation control defines a group of calculation rules for a specific purpose, such as creating a bill. Calculation controls support the creation of different versions of a calculation over time by maintaining an effective dated list of the calculation groups that define the rules.

To set up a calculation control, open **Admin Menu** > **Calculation Control**.

Calculation Control Query

Use the query portal to search for an existing calculation control. Once a calculation control is selected, you are brought to the maintenance portal to view and maintain the selected record.

You can also click the Add link in the query zone's title bar to add a new calculation control.

Calculation Control Portal

This portal appears when a calculation control has been selected from the Calculation Control Search results.

The topics in this section describe the base-package zones that appear on this portal.

Calculation Control

The Calculation Control zone contains display-only information about a calculation control. This zone appears when a calculation control has been selected from the Calculation Control Search results, or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *C1_CALC_CTRL*.

An Overview Of Factors

The primary purpose of the calculation engine is to support the calculation of an obligation’s charges. One of the most common rules for calculating a charge is to apply a rate to a fixed or assessed value. While it is possible to specify the rate directly on the rule, it is more flexible to have the rule reference a factor whose values can be maintained independently.

A factor defines a centrally stored set of values for use in calculations and other processes. A factor can have different values depending upon some definable attribute of a system object, such as a district associated with an address.

You can use factors to define any of the following types of values:

- A monetary amount. This would typically be used to define fixed charges such as fees.
- A number. This would typically be used to define charges per unit, such as five cents per square foot.
- A percentage. This would typically be used for tax rates and exemption percentages.

As a general guideline, you would use a factor when any of the following situations exist:

- The amount being charged is dependent on some characteristic of a related attribute. For example, if charges levied for distribution to schools differ from one district to another, you would use a factor to define variable charges for each district.
- The same charge exists in many calculations. For example, if there are several calculation controls that all include a fixed handling charge, it would make sense to use a factor to levy this charge rather than specify the same value on multiple calculation rules.
- The amount being charged is dictated by some external organization and therefore can change independently from the calculation rule.

The following sections provide a description of how factors are structured and details on how to configure and maintain factors.

The Structure Of A Factor

All factors have the capability of having different values depending on some characteristic of an associated object such as an asset, or the asset billing address. For example, a factor used to levy a waste disposal fee would have a different rate depending on the waste management district associated with the asset's address. Factors share a common structure, made up of the following tables:

- A **factor** contains descriptive information and attributes that control how the factor may be used in the system.
- A **factor characteristic value** defines each instance of a characteristic for which there is a unique value for a charge. In the waste management district example, a factor characteristic value would exist for each district that levied a waste management fee.
- Each factor characteristic may have many **factor values** over time. To continue the example, waste management fees may vary from one revenue year to another.

NOTE:
Some factors don't need a characteristic. There are factors whose value does not differ based on a characteristic. For example, the values of a factor that levies a state wide asset related charge will not vary because of a specific asset attribute. However, because of the relational design of the system, every factor value must reference both a factor and a characteristic value in the database. The base product provides two factor business objects to support the different situations. The Simple Factor BO flattens the characteristic type and value to base values of 'Not Applicable', so that only the factor details and effective values need to be entered. The Characteristic Based Factor BO supports the maintenance of the complete factor, factor characteristic values and factor values relationship.

FASTPATH:
For more information about setting up characteristics, see *Setting Up Characteristic Types & Their Values*.

Deriving Factor Characteristic Values

The Structure Of A Factor section described how factor values can vary according to an attribute of another object. This section describes how the system determines which characteristic value is applicable when fetching a factor value for calculations.

When you set up a factor to use a characteristic, you also need to configure the algorithm that can derive its appropriate characteristic value from a given source. The Determine Characteristic plug-in spot provides the ability to pass in a list of maintenance objects and their prime keys in addition to the required characteristic type. This allows the algorithm to apply business rules based on knowledge of the internal MO relationships, to find the correct source of the value.

The base product supplies a number of algorithms that may be used by your implementation to derive factor values from the main master data objects, including asset, asset billing address, tax role and account. These algorithms recognize the relationships between those objects and the key bill objects of tax role and obligation. Click *here* for a complete list of the determine characteristic value algorithms provided by the system.

NOTE:
In some circumstances, the invoking logic may already have access to the correct characteristic value. The factor value retrieval process allows for a specific characteristic value to be passed in, in which case, the determine characteristic value algorithm is not used.

Factors And Distribution Codes

Another aspect to consider when defining factors is the associated GL distribution codes. Factors allow you to associate a specific distribution code to the charge that is defined by that factor. For factors whose values vary by a characteristic value, the system allows you to define a unique distribution code to each characteristic value / factor value combination.

For example, if you need to calculate charges that are distributed to different school districts and each school district contribution is calculated at a different rate, you can define a separate distribution code for each district and factor value. Calculation rules can then override the rule level distribution code with the one specific to the factor value used in computing the calculation value.

Refer to *Using The Apply Rate To Base Value Rule* for an example of how calculation rule logic can apply factor value specific distribution codes.

Setting Up Factors

To set up a factor, open **Admin Menu** > **Factor**.

Factor Query

Use the query portal to search for an existing factor. Once a factor is selected, you are brought to the maintenance portal to view and maintain the selected record.

You can also click the Add link in the query zone's title bar to add a new factor.

Factor Portal

This portal appears when a factor has been selected from the Factor Search results.

The topics in this section describe the base-package zones that appear on this portal.

Factor

The Factor zone contains display-only information about a factor. This zone appears when a factor has been selected from the Factor Search results, or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *C1_FACTOR*.

Factor Characteristic Values

The Factor Characteristic Values *List zone* appears only if the factor displayed has values that vary by the values of a attribute represented by a characteristic type. A row is displayed for every characteristic value for the factor’s characteristic type that governs a factor value. The list includes the characteristic description, the distribution code applicable to amounts based on this factor and characteristic value combination and details of the value currently in effect.

Click the *broadcast* icon to open the factor values zones to view and maintain the values for the adjacent Factor / Characteristic Value combination.

Factor Values

The Factor Values *List zone* provides a history of all the effective dated values for this factor and the broadcast characteristic value, if applicable. The values are shown in descending order by effective date. The list includes a hyperlink on the effective date field that navigates to the factor value’s maintenance portal.

The standard actions of **Edit** and **Delete** are available for each Factor Value.

Click the **Add** link in the zone's title bar to add a new Factor Value.

Factor Value Portal

This portal appears when an effective dated value has been selected from the Factor Values list zone.

The topics in this section describe the base-package zones that appear on this portal.

Factor Value

The Factor Value zone contains display-only information about a specific effective dated factor value. This zone appears via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *C1_FACTOR_VALUE* .

Designing Your Calculation Rules and Groups

This section provides an overview of how to create and maintain calculation rules used with the calculation engine. Initial sections describe base objects and reusable components that may be used in many rules. Later sections of this document provide information on some of the key calculation rules provided with the base product.

Calculation Data

The calculation rules are intended to operate on the internal calculation data area, both to reference previously populated details and to add new calculation details. Refer to the base data area **C1-CalcEngineProcessorData** for information on the details passed to the processor.

In addition to the information defined in this data area, the calculation engine expects the following inputs:

- **Calculation Control Version** identifies the set of calculation rules to be executed for this calculation
- **Calculation Object** is the system object which is the subject of the calculation, such as a bill.

NOTE:
The calculation object is provided as a read-only reference for calculation rule logic. The calculation engine does not support updates to the object as a result of calculation rules.

For an example of how to invoke the calculation processor, refer to the base algorithm that generates bill details: *C1-TB-GN-CLC*.

Base Calculation Objects

There are a number of base calculation engine objects that can be used by your implementation as the basis for new calculation rules.

The base product provides a generic BO for Calculation Group, **C1-CalcGroup**. Your implementation can add additional business rules to this BO as required.

The base product provides a parent BO for Calculation Rule, **C1-GenericCalcRule**. This BO defines the common elements for all calculation rules. It is not intended to be used stand-alone. Your implementation can include this BO in each rule BO and add the additional components as required.

The base product supplies a number of BOs that apply particular Calculation Rules, some of which are described in later sections. Refer to the business objects that reference the Calculation Rule maintenance object **C1-CALC-RULE** for a complete list.

The base product provides a parent BO for Calculation Rule Eligibility Criteria, **C1-GenericCalcRuleElCrt**. This BO defines the common elements for all eligibility criteria objects. It is not intended to be used stand-alone. Your implementation can include this BO in each eligibility criteria BO and add the additional components as required.

The base product supplies a BO that applies eligibility criteria using a flexible expression evaluator, which is described in a later section. Refer to the business objects that reference the Calculation Rule Eligibility Criteria maintenance object **C1-CALC-R-EL** for a complete list.

Your implementation can define additional calculation rule and eligibility criteria BOs as required. The following points highlight the important configuration for new business objects for these maintenance objects:

- Create a new business object and include the appropriate parent business object.
- Develop the appropriate processing algorithm to perform the logic for the BO.
- For any configuration required by the processing algorithm, consider whether it should be defined by a business user when configuring the rule or criteria. If so, include the appropriate elements in the business object's schema.

In addition to the BOs listed above, the base product provides a number of reusable components that map common rule configuration elements that may be used when developing new rules. The following topics highlight some of the base components that may be used by your implementation.

Common Variables

The two main calculation rule and eligibility criteria BOs provided in base use conditional expressions in their rule logic. These expressions are supported in turn by a common set of variable elements that are defined in a data area to allow reuse across rules.

The following topics highlight some of the available features for using conditional variables.

Variable Definition

The Calculation Engine Variable List DA incorporates the following elements:

- **Variable** is the unique identifier for the variable. Variable identifiers must be single letter from a to z.
- **Variable Type** defines the type of variable to be used. Valid types are **Calculation Line**, **Characteristic**, **Constant**, **Factor** and **Value Detail**
- **Value Class** defines the type of value this variable represents. Valid types are **Alphanumeric**, **Money**, **Number** and **Percentage**
- **Variable Details** are additional details need to derive the variable value. These are described in the section below.

- **Value Required** indicates whether a value should exist for the variable. If a value is required and the value cannot be determined, the calculation processor will report an exception; if not, the value will be treated as zero or blank in the expression, as applicable.

Additional Variable Details

Most variable types require additional configuration details, as follows:

Calculation Line	Enter the category types and values that identify the calculation line. Multiple categories may be entered; the variable value will then be the sum of the values of all targeted lines. • **Calculation Line Category Type** is the category type of the targeted calculation line or lines • **Calculation Line Category Value** is the category value of the targeted calculation line or lines.
Characteristic	Enter the following details that identify the characteristic value: • **Calculation Characteristic Entity** is the source entity for the characteristic value. Valid entities are: **Account**, **Address**, **Asset**, **Characteristic Collection** and **Tax Role** • **Characteristic Type** is the type of characteristic from which the variable value is derived. • **Effective Date Source** is the source of the effective date to use when fetching the characteristic value. Valid values are: **Calculation Period Start Date** , **Calculation Period End Date**, **Revenue Period Start Date** and **Revenue Period End Date**
Constant	Specify the constant value to be used in the expression
Factor	Enter the following details that identify the factor value: • **Factor** is the factor from which the variable value is derived. • **Effective Date Source** is the source of the effective date to use when fetching the factor value. Valid values are: **Calculation Period Start Date**, **Calculation Period End Date**, **Revenue Period Start Date** and **Revenue Period End Date**
Value Detail	Specify the value detail type whose value is to be used in the expression

Deriving Variable Values

With the exception of factors and characteristics, the calculation processor derives variable values from details within the calculation data area maintained in memory. The following highlights important details about the logic used to derive the values for certain variable types.

Calculation Line	The calculation processor uses common logic to identify all calculation lines that are members of the target categories and return the summarized value. The same logic is used by base rules for summarizing calculation lines.
Characteristic	The following rules apply when determining the characteristic value: • If the source is **Characteristic Collection**, the processor will look only within the internal calculation data characteristic collection for the value

	• If the source is one of the valid entities, the processor will derive the entity ID from the obligation passed to the internal calculation data area. If the source is **Tax Role**, the processor assumes that the obligation is linked to a tax role. If the source is **Asset**, the processor assumes that the obligation's tax role is linked to an asset. Note that an entity of **Address** is assumed to be an asset billing address.
Factor	If the variable type is factor and the factor values vary by a characteristic type, the calculation processor assumes that the characteristic values supplied to the internal data area take precedence over the factor determine characteristic logic. If a value for the factor's characteristic type exists in the internal collection, that value will be used by the factor value processor to derive the value.

Using Conditional Expressions

The base product makes use of common logic that evaluates basic mathematical expressions. This logic can be used to compute values that form part of the calculation or to evaluate conditions to determine whether to apply calculation rules.

The following topics highlight some of the available features for expressions. Other mathematical operators and functions are supported but these are not normally used for standard revenue calculations or simple conditions.

Variables

Variable identifiers must be a character between a-z. Variable identifiers are case sensitive.

Operators

The following types of expression operators are supported.

Mathematical Operators	Operators of *, /, +, - are allowed. Examples of expressions using these operators are: • **x** = **a** + **b** + **c** where the value of **x** should equal the sum of the values **a**, **b** and **c** • **x** = **a** * **r** where **r** is a rate factor variable
Comparison Operators	Operators of =, != (not equal), <, >, <= and >= are allowed. Examples of expressions using these operators are: • **x** > **a** where the value of **x** should be greater than the value of **a** • **x** != 0 where **x** must have a non-zero value
Grouping Operators	Parentheses may be used to formulate expressions with sub-computations. An example of an expression using grouping is: • **x** = (**a** + **b**) / **c** where the value of **x** should be equal to the result of adding values **a** and **b** then dividing by the value of **c**

Functions

The following examples show the use of the most common supported functions:

Ceiling	**x** = ceiling (**a*** **b**) where **x** should equal the result of multiplying the value of **a** by the value of **b** and rounding up to the nearest whole number
Floor	**x** = floor (**a*** **b**) where **x** should equal the result of multiplying the value of **a** by the value of **b** and rounding down to the nearest whole number
Round	**x** = round (**a*** **b**) where **x** should equal the result of multiplying the value of **a** by the value of **b** and rounding the nearest whole number. A result of 1.5 would round to 2 and a result of 1.4 would round to 1.
Any	any [value in **a** | value !="] where **a** is a value from a repeating line and the condition is true if any of the values of **a** are not blank
All	all [value in **a** | value !="] where **a** is a value from a repeating line and the condition is true if none of the values of **a** are blank

The system provides a base calculation rule BO and a base calculation rule eligibility criteria BO that make use of conditional expressions to support a wide variety of computations and eligibility rules. These BOs make use of reusable variable definitions that are mapped to standard elements in the calculation processor data area.

FASTPATH:
Refer to *Common Variables* for information about using common variable definitions.

Common Elements

Several of the base product calculation rules use similar elements to govern how a calculation line affects a specific calculation. These elements are defined in data areas to allow reuse across rules.

The Calculation Rule Print Details DA incorporates the following elements:

- **Print Option** indicates whether to always print this calculation line, print only if the value is not zero or never to print.
- **Description On Bill** is the text to print on a bill for this calculation line.

The Calculation Rule Rounding and Precision DA incorporates the following elements:

- **Rounding Method** indicates whether to use a rounding method of 'up', 'down' or 'nearest' when rounding the resulting calculation value.
- **Precision** is the decimal precision to which the resulting calculation value is rounded.

The Calculation Rule Member Category List DA maps the member calculation line category type and value collection of a calculation rule. Most rules will incorporate this configuration as a means of categorizing the resulting lines.

The Calculation Rule Target Category List DA maps the target calculation line category type and value collection of a calculation rule. Rules whose processing logic is dependent on the results of other calculation rules will incorporate this configuration as a means of identifying the required lines.

There are a number of other common elements that are common to rules configuration that are not defined in data areas, as follows:

- **Create Bill Line** indicates whether the resulting calculation line needs to be captured on the associated bill.
- **Include In Total** indicates whether the resulting calculation value contributes to the overall bill total. Calculation lines of this type are regarded as having a financial impact and must have an associated distribution code.
- **Distribution Code** is the distribution code that is associated with the resulting calculation line. The logic that processes the result of the calculation will use this distribution code to record the financial effect of this calculation line.

NOTE: The distribution code associated with the calculation line may be overridden by the distribution code configured on the factor used to compute the line value.

Refer to *Factors And Distribution Codes* for more information on assigning distribution codes to specific factor values.

The base product includes common processing logic to validate these configuration elements and to utilize them in rule processing. Refer to the base product calculation rules for examples.

Using The Apply Rate To Base Value Rule

The base product provides a Calculation Rule BO that is designed handle a large range of common calculations, **C1-ApplyRateToBaseValue**.

The **Apply Rate To Base Value** BO allows a business user to define a an expression that computes a base value and a specific rate to be applied to that value. The rule produces a calculation line whose result is computed by multiplying the rate by the base value and which captures details about the rate and base value that can be used in later rules.

The following topics highlight some of the available features for this rule.

Variables

The variables to be used in the base value computation expression are defined using a common data area. Refer to *Common Variables* for more details.

Base Value Formula

The formula for computing the base value to be used in calculating this line is defined as a mathematical expression. Refer to *Using Conditional Expressions* for more details.

Rate Details

The details of the source of the rate for the computation are defined in a base data area, **C1–CalcRuleRateData** . The following section provides details of the available rate types:

Constant	The rate to be applied may be a constant, in which case the following details are required: • **Value Class** is the type of value represented by the rate. • **Constant Value** is the rate value to be used.
Factor	The rate to be applied may be a factor, in which case the following details are required: • **Value Class** is the type of value represented by the rate. • **Value Required** indicates whether it is an error if the factor value cannot be determined. • **Factor** is the factor from which the rate value is derived. • **Effective Date Source** is the source of the effective date to use when fetching the factor value. Valid values are: **Calculation Period Start Date**, **Calculation Period End Date**, **Revenue Period Start Date** and **Revenue Period End Date**
Value Detail	The rate to be applied may be a value detail, in which case the following details are required: • **Value Class** is the type of value represented by the rate. • **Value Required** indicates whether it is an error if the rate value cannot be determined.

- **Value Detail Type** is the detail type within the calculation data from which the rate value is derived.

Additional Rule Configuration

The rule includes other common configuration options governing how the calculation line is treated. Refer to *Common Elements* for more details.

Defining The Rate Source

This section highlights additional considerations when configuring the rate source. For a full description of the rule logic, refer to *C1-CR-APRTBS* .

- If the rate source is a factor, the distribution code for this calculation line will be derived from the distribution code on the factor value, if applicable. You may wish to consider using a factor to define rates for the types of charges that are disbursed to different entities depending on some system attribute such as an address. For example, asset based revenue collected to fund schools may be distributed to different school districts depending on the related billing address. Rather than have a separate rule for each possible distribution code, the factor for those amounts can be configured to have a different rate value and distribution code based on district
- In addition, if the rate source is a factor, the rule retains the characteristic type and value governing the factor value on the calculation line, if applicable. Choose a factor for the rate if you want to capture that information associated bill lines or later GL distributions.
- Some calculation rules may not apply depending on whether a system attribute applies. For example, revenue levied to fund special programs such as parking control may only apply to certain locations. Rather than control this by applying eligibility criteria to the rule, you may wish to configure the rate as a factor and indicate that it is not an error if no value is found. If the attribute governing the rate does not apply, the factor value will be zero and the calculation line result will also be zero. This behavior can be used in conjunction with the ability to configure zero value not to be printed to have later processing ignore these lines. Similarly reasoning may apply to calculation lines based on value details that may or may not be present in all cases.

Chapter 10

Defining Penalty and Interest Options

Penalty and Interest (P&I) is the commonly used term to describe a wide range of penalty, interest and fee liabilities. These charges are usually imposed by legislation.

- A penalty is a liability imposed for noncompliance with tax law. Revenue authorities can file penalties against taxpayers for "failure to file," "failure to pay," "substantial understatement," and so on
- Interest compensates the government for cost of money over time
- Fees are imposed to reimburse revenue authorities for their cost of doing business. For example, revenue authorities may incur fees for sending certified mail or for handling defective checks and will pass those fees onto the taxpayers

P&I may be triggered in any of the following ways:

- Event driven, for example, reversing a payment due to non-sufficient funds will impose a fee. These types of transactions are simply one-off adjustments that are created by an appropriate algorithm when the event occurs
- User triggered, for example, auditors may impose an inadequate record-keeping penalty their discretion. The amount may be predefined or determined by the auditor. These types of transactions are created manually or may be created as part of a business process defined by your implementation.
- Ongoing, system-generated charges based on predefined configuration rules. The remainder of this chapter focuses on these types of charges.

The Big Picture of Credit Allocation

Before explaining penalty and interest logic, another important concept called credit allocation must be introduced. The following sections provide more information.

Credit Allocation

Credit allocation is the process of taking credits within an obligation and applying business rules to allocate them against tax, penalty, interest and fees.

Credit allocation sequences specify an order to apply the credit against liability types. This is usually very specific. For example, you could set up the credit allocation sequence to allocate credits in the following order:

1. Defective check fee
2. Failure to pay penalty
3. Interest
4. Failure to pay penalty
5. Tax
6. Outsource collection agency fee

They may also differ based on the type of credit. For example,

- Withholding credit may be allocated differently than a payment
- Payment received before the due date may be allocated differently than a late payment
- Individual payments may have unique priorities dictated by a court order or overridden by a user

The following sections provide more information about credit allocation.

Credit Allocation Is Dynamic

The term "dynamic" is used to describe credit allocation because the system does not store the matching of credits to debits in the database. Rather, the system applies the credit allocation rules real-time when a request for the detailed balance of an obligation is performed.

Debt Categories and their Priorities

In order to be able to apply credits to debits at the granular level shown above, each financial transaction whose amount is a debit (amount is greater than or equal to zero) must be assigned a debt category

- For adjustments that are debits, the debt category is defined on the *adjustment type*
- For bill segments, the debt category is defined on the *obligation type*
- For payments that are debits, for example "negative" payments used to refund money via direct deposit, the base product provides an FT freeze algorithm called *C1-CFTZ-RPDC* to plug into the Account Type that populates the debt category.

Debt categories may also be linked to credit financial transactions. The base algorithm that allocates credits to debits uses the debt category on credit financial transactions as a prioritization. See *Determine Balance Details Algorithm* for more information.

The debt category priority defines the order in which credits may be applied to debits.

- The default debt category priority is defined on the *obligation type*
- If a certain type of credit adjustment, for example a withholding credit dictates a different priority than the default, define the override debt category priority on the *adjustment type*.
- A *financial transaction* could reference a debt category priority directly for unusual situations where a payment may require a special debt category priority

Determine Balance Details Algorithm

The dynamic credit allocation is performed by an algorithm plugged into the **Determine Balance Details** plug-in spot on the *obligation type*. The determine balance details algorithm is used throughout the base P&I calculation algorithm but may also be called by other processes to get a balance broken out by assessment / debt category.

The following information highlights the logic in the base algorithm *C1-PS-DB-STD*.

The algorithm may receive specific FTs in which case it will only use those FTs. Otherwise, it retrieves the FTs for the obligation.

A special plug-in spot **Retrieve FT List** on *obligation type* is called to retrieve the FTs for an obligation.

NOTE:
Base plug-in. Click *here* to see the algorithm types available for this system event and for more information about the behavior of this plug-in spot.

NOTE:
Debt Category / Assessment on Credits. This algorithm assumes that if a credit financial transaction references an assessment and/or a debt category that this information is used to prioritize the application of the credit, not restrict its application. This ensures that the results do not include one assessment (or debt category) with an overall debit balance and another assessment (or debt category) with an overall credit balance.

The algorithm processes credits in order from oldest to newest. For each credit found,

- If it references a debt category or assessment (group FT ID) or both, the credit is allocated first to debits with a matching debt category and / or assessment whose effective date is *on or before* the credit's effective date.
- If credit is still remaining, the credit is allocated to debits whose effective date is *on or before* the credit's effective date using the credit allocation rules for the appropriate *debt category priority* for this credit.
- If credit is still remaining, the credit is allocated to debits with a matching debt category and/or assessment whose effective date is *after* the credit's effective date.
- If credit is still remaining, the credit is allocated to debits whose effective date is *after* the credit's effective date using the credit allocation rules for the appropriate *debt category priority* for this credit.

The algorithm returns a detailed list of FTs including which credits were applied to which debits.

NOTE:
FT Details. The details of the FTs for the obligation and how the credits are allocated to the debits is not hard-coded but rather is defined in a data area supplied to the Determine Detailed Balance plug-in spot. The base algorithm uses the data area **C1-PI-MainFtInfo**. However, if your implementation requires different information to be supplied for each FT, you may introduce your own data area and an appropriate determine detailed balance algorithm to populate the details accordingly.

Credit Allocation Zones

The base product provides zones on 360 degree view (on the Financial tab) and on control central (on the Account Information and Taxpayer Information tabs) to show the results of credit allocation for the account or taxpayer's current balance.

FASTPATH:

Refer to *Credit Allocation Zone*for details.

The Big Picture of Penalty and Interest

The following sections describe topics related to penalty and interest calculations.

P&I is Calculated for an Obligation's Assessments

The system creates, stores, and maintains tax liability *assessments*. These assessments may incur P&I over time. When more than one assessment exists for an obligation, the type of assessment may control P&I rates and rules. For example, if the taxpayer is being audited, harsher penalty rates and rules are used.

As described in *Credit Allocation* the base product determine detailed balance algorithm allocates credits for an obligation across assessments such that an obligation with multiple assessments would not incur P&I on one assessment while another assessment is in credit. As a result, the P&I calculation algorithm is invoked for an obligation and it always calculates penalty and interest for all assessments for the obligation.

P&I Rules for a P&I Control Define the Calculation

For each obligation type where ongoing, system generated penalty and interest rules apply, you must create a P&I control with its collection of P&I Rules that define the distinct penalty, interest charge or fee.

Each P&I rule controls the actual calculation and allows a user to configure information such as the basis of the charge, the rate used to apply the charge and other controls such as whether there is a maximum amount that the overall calculations may not exceed.

P&I rules to apply to an obligation or configuration related to existing rules may change over time, based on legislation changes or business rule changes. When this occurs, a new P&I control with its new set of rules must be created. The obligation type includes an effective dated link to the P&I controls that govern its P&I charges.

FASTPATH:
Refer to *Apply P&I Rules for Each Time Period* for information about how P&I rule calculations are applied during the Calculate P&I process.

FASTPATH:
Refer to *Designing Your P&I Control and P&I Rules* for information about designing your rules.

P&I Calculation Algorithm is the Engine

An algorithm plugged into the **P&I Calculation** plug-in spot on the *obligation type* is responsible for calculating / forecasting penalty and interest for an obligation.

NOTE:
Configuration Requirements. The base product P&I calculation algorithm relies on many other algorithms plugged into obligation type for it to work properly. In addition, it expects effective P&I controls with at least one P&I rule to exist for the obligation type. Refer to *Setting Up Penalty and Interest* for more information.

Calculation Overview

The following sections highlight the logic in the base algorithm *C1-PI-CALC*. It has been designed to call many other algorithm plug-in spots to perform key steps in the calculation. The goal is for your implementation to use the base product P&I calculation algorithm and implement your organizations specific rules by plugging in appropriate algorithms in the various plug-in spots called by the base algorithm. However if the logic in the base algorithm does not satisfy your business requirements, you may introduce your own, using this algorithm as a sample.

At a high level, the base algorithm follows these steps

- Determine whether or not this obligation is *eligible* for P&I calculations.
- For eligible obligations
  - Determine *discrete time periods* and perform any other pre-processing needed for the calculations.
  - For each time period, apply the P&I rules. Calculations are performed in memory.
  - Post processing, including canceling and creating the P&I adjustments if the algorithm is called with a P&I Calculate / Update action.
- Update the obligation's calculate to date

The algorithm may be called with an action of **P&I Calculate/Update, P&I Standard Forecast** or **P&I Detailed Forecast**. When calling the algorithm with a "forecast" action, the calling program may provide specific FTs in which case it will only use those FTs. Otherwise, it retrieves the FTs for the obligation.

A special plug-in spot **Retrieve FT List** on *obligation type* is called to retrieve the FTs for an obligation.

NOTE:
Base plug-in. Click *here* to see the algorithm types available for this system event and for more information about the behavior of this plug-in spot.

Financial Transaction Details

The list of the FTs for the obligation includes detail for each FT that is needed by the penalty and interest calculation. This list of details is not hard-coded but rather is defined in a data area supplied to the P&I Calculation plug-in spot and to the Retrieve FT List plug-in spot. The base algorithm uses the data area **C1-PI-MainFtInfo**.

Besides the details of the FTs passed back and forth to services that invoke this plug-in spot, the base P&I calculation algorithm uses data internally that must be passed to the various plug-in spots it invokes. The base algorithms use the data area **C1-PI-InternalCalculationInfo**. The data area has the following information:

- A calculation periods collection. This includes start and end dates along with the P&I rules in effect for those dates.
- Existing FT collection. This includes all the FTs that exist for the obligation and is used to compare to the P&I charges being calculated in the current execution to determine if any changes are needed to historical periods.
- Running charges collection. This is the list of charges that are populated by the P&I rule algorithms that calculate the charges. The amounts in this collection are captured with detailed precision to minimize ongoing rounding discrepancies.
- Working financial transaction collection. This collection represents the new list of FTs that are a result of the current P&I calculation. When the service is invoked with an "update" action, this is the list that will be used when creating the actual FTs at the end of the process. If the service is invoked with a "forecast" action, this is the list of FTs that is used to pass out to the calling program. The definition of this list matches the definition of the list of FTs in the P&I Main FT Info data area. This list also includes a collection of detailed P&I calculation info that may be populated by P&I rule algorithms when invoked with a "forecast" action. Refer to *P&I Calculation Details* for more information.
- Waiver information collection. This list contains detailed information for any active waiver that exists for the obligation.

NOTE:
If your implementation requires additional information to be passed around to the various P&I calculation plug-in spots that support the internal P&I information, you may extend the base data area to define the additional elements needed. Alternatively, you may introduce your own data area. The appropriate decision for your implementation will depend on the type of additional information needed. For example, if your implementation requires some addition information about the taxpayer or the obligation to help determine rule eligibility, for example, then extending the base data area is the correct solution. However, if you require different or additional data within one of the existing groups in the base data area, it may be more correct to introduce a new custom data area.

Eligibility

There are some types of taxpayers that may be exempt from penalty and interest. Examples of such taxpayers include other government agencies and non-profit organizations.

The base product provides an *obligation type* plug-in spot **P&I Eligibility** where an implementation may plug in algorithms that check an obligation's eligibility for penalty and interest. The P&I calculation algorithm provided with the base product calls the P&I eligibility plug-ins prior to performing any penalty or interest calculations. If the algorithms indicate that the obligation is ineligible, no calculations are performed.

NOTE:
Base plug-in. Click *here* to see the algorithm types available for this system event and for more information about the behavior of this plug-in spot.

Recalculate Every Time

Various events may cause the system to recalculate historic penalty and interest:

- Back-dated payment
- Waive existing penalty or interest
- Change to the obligation's override filing due date
- Canceling an assessment

For normal requests from the system to "bring P&I up to date", where historical calculations are not affected, the system still recalculates the P&I from the beginning every time.

One reason for this logic is to ensure that P&I is always accurate. It could be that something has changed in the system that does affect historical calculations and a request to recalculate historical P&I was not performed. The next time P&I runs, recalculating from the beginning ensures that the latest calculation reflects the correct charge.

Another important reason to recalculate P&I from the beginning every time P&I is brought up to date is to avoid rounding issues that may occur over time.

Consider the following example.

- A monthly penalty amount for a delinquent taxpayer is 34.3444
- If the system calculates, rounds and posts, each month ignoring the running total, the penalty after 2 months is
  - Month 1 calculation: 34.3444, rounded to 34.34
  - Month 2 calculation: 34.3444, rounded to 34.34
  - Total: 68.68
- If instead the system keeps track of the running total and creates the adjustment based on the latest running total, the calculation is as follows:

- Month 1 calculation: 34.3444, rounded to 34.34
- Month 2 calculation: 34.3444 + 34.3444 = 68.6888 less amount already posted (34.34): 34.3488, rounded to 34.35
- Total: 68.69

Over time these rounding differences add up. To avoid that the system calculates from the beginning every time P&I is calculated and the running total is kept throughout the calculations.

Calculate For Discrete Time Periods

There are many discrete time periods for which P&I must be calculated:

- Every date for which a credit financial transaction exists (because the calculations are based on an outstanding balance, which is affected by each credit).
- Any effective dated change in the P&I Control linked to the obligation
- Any accrual day of the month. This is necessary if you have a monthly charge that should accrue on a given day of the month and the charge includes other charges in its calculation basis. For example, if penalty accrues on the first day of the month and includes interest in its calculation basis, you must stop and calculate interest and then the penalty on the accrual day of the month so that your calculation basis is accurate.
- Effective dates of any active waivers. If any assessments for the obligation have a waiver that is effective dated, the system should stop and calculated up to that date and then after that date for the waived amount to be accurate.
- More… Your implementation may identify other events in the system that affect P&I calculations. For example, if a bankruptcy is logged in the system as a case, P&I should be calculated up to the bankruptcy start date and then skipped until the bankruptcy end date

The base product P&I calculation algorithm relies on *P&I pre-processing* algorithms to build this list of dates. Once the list is built, the P&I calculation algorithm *applies P&I rules for each time period*.

Pre-Processing

When preparing to calculate penalty and interest for an obligation, the P&I calculation plug-in must gather some information that is needed throughout the processing. For example:

- The P&I controls and P&I rules in effect for the full calculation period.
- Information about waivers that are in effect for the obligation's assessments.
- Identifying the dates during the full P&I calculation period that affect the calculation such that the process should stop and calculate P&I up to that date.
- Determining whether some P&I rules are not applicable for an obligation or one of its assessments. For example, a failure to file penalty is only applicable for obligations where the taxpayer did not file the return on time; and perhaps only applies to the original return and not any amendments.
- Determining whether some P&I rules are only applicable during certain date ranges. For example, if a taxpayer has extended their filing date, perhaps the failure to file penalty is only applicable after the extended filing date but the failure to pay penalty is applicable starting from the original payment due date.

The system provides two plug-in spots to use for building up all this information.

- An *obligation type* plug-in spot **P&I pre-processing** is used to define dates and retrieve and configure information that is related to all P&I rules for the obligation type. For example, dates are built for all credit financial transactions for the obligation.

NOTE:

Base plug-in. Click *here* to see the algorithm types available for this system event.

- A P&I Rule plug-in spot **Rule Pre-processing** is used to define dates and retrieve and configure information that is specific to a given P&I rule. For example, a rule that is for the Failure to File penalty has a pre-processing algorithm to determine if the taxpayer failed to file on time.

NOTE:
Base plug-in. Click *here* to see the algorithm types available for this system event.

These algorithms populate information in the P&I Internal Calculation Info data area (**C1-PI-InternalCalculationInfo**) for use by subsequent algorithms in the process.

The following sections highlight additional details related to the responsibility of pre-processing algorithms.

Build P&I Controls and Call Rule Pre-processing Plug-ins

The base product provides an algorithm that should be the first pre-processing algorithm plugged in on your obligation types. It builds the initial date ranges for the P&I calculation, builds the list of P&I Controls and P&I Rules that are in effect for the full date range and, for each P&I rule, it calls the Rule pre-processing algorithms, if any exist.

For more information, refer to the algorithm type *C1-PI-PR-PIC*.

Marking Rules as Not Applicable Based on Periods

Although the initial pre-processing algorithm builds a list of all the P&I rules that are in effect for the full P&I calculation period for this obligation, there may be P&I rules that don't apply for the obligation or rules that should only apply for certain time periods. For example:

- The failure to file penalty should only apply if the taxpayer did not file the tax return by the due date.
- A collection fee that is based on the outstanding balance of the tax should only apply if the taxpayer was selected for overdue processing and should only apply to the time period on or after the creation date of the overdue process
- For a taxpayer that filed for bankruptcy, P&I Rules should not be applied during specific bankruptcy dates.

To mark P&I rules as not applicable for certain time periods (or for all the time periods built for the calculations) P&I pre-processing algorithms on the obligation type or rule pre-processing algorithms on the P&I rule should find the P&I rule in the calculation periods collection in the P&I - Internal Calculation Info data area and mark it as not applicable. Refer to the base product rule pre-processing algorithm Determine Failure to File Applicability *C1-PI-RA-DFF* for an example of this type of logic.

Marking Rules as Not Applicable for One or More Assessments

There may be P&I rules that only apply to a subset of the assessments for an obligation for one or more time periods. For example:

- The failure to file penalty may apply only to the original assessment and not to amendments
- Different failure to pay penalty rules may exist for the original assessment and for amendments
- For a taxpayer that filed for bankruptcy, P&I Rules should not be applied during specific bankruptcy dates.

To mark P&I rules as only applicable for certain assessments in a given time period, P&I pre-processing algorithms on the obligation type or rule pre-processing algorithms on the P&I rule should find the P&I rule in the calculation periods collection in the P&I - Internal Calculation Info data area and indicate the assessments that are not applicable. Refer to the base product rule pre-processing algorithm Determine Failure to File Applicability *C1-PI-RA-DFF* for an example of this type of logic.

Determine if Active Waivers Exist

The base product provides a plug-in *C1-PI-PR-WVR* to retrieve details about waivers that exist for any of the obligation's assessments. This information is used by any subsequent algorithm that may need information about the waivers.

Apply P&I Rules for Each Time Period

Once the discrete time periods for P&I calculation are determined, the base P&I calculation algorithm needs to determine the balance by debt category for the time period in question and call the P&I rule algorithms.

Balance By Debt Category During P&I Calculation

To calculate the balance by debt category, *Determine Balance Details* algorithm is used. However, because the P&I calculation algorithm is recalculating P&I from the beginning every time and needs the balance recalculated for each period, it needs to be explicit about which financial transactions are used in the balance calculation. For example:

- When calculating the balance as of a given date, credits effective on that date should be ignored because the system is trying to find the balance as of that date prior to applying the credit. However all debits on or before the given date should be considered.
- Existing P&I transactions should not be factored into the calculation, only the ones created by this P&I run.
- Existing *waiver* transactions should not be factored into the calculation. However, waiver transactions corresponding to the P&I transactions calculated in the current run must be considered.

To ensure that the appropriate financial transactions are used to calculate the balance for each time period, a special plug-in spot on the *obligation type* is used: **P&I Prepare Periodic Balance**. This plug-in is responsible for calling the **Determine Detailed Balance** plug-in passing the appropriate financial transactions. Note that this plug-in spot receives an indication as to whether it's the **Initial**, **Interim** or **Final** call to get the balance details. This information allows the plug-in to do extra steps at the beginning or at the end.

NOTE:
Base plug-in. Click *here* to see the detailed description for the base algorithm type available for this system event and for more information about the behavior of this plug-in spot.

NOTE:
Adjustment Category. The base plug-in for the Prepare Periodic Balance relies on the adjustment type category values of **Penalty and Interest** and **Waiver** to identify adjustments that should not be factored in during the **Initial** call to the algorithm. Care should be taken to only configure *adjustment types* with the Penalty and Interest category if they are adjustment types created through the P&I rules driven P&I calculation.

Once the balance is available for each time period, for each assessment linked to the obligation, the P&I calculation algorithm invokes the rule processing algorithm for each P&I rule in effect for the obligation type's P&I control during this time period.

P&I Rules Calculate the Charge

The actual calculation of each P&I rule's charge is done in the **Rule Processing** algorithm plugged into the business object for the P&I rule. Configuration that the rule processing algorithm needs to successfully calculate the charge for each period is often captured on the P&I rule when defining it.

All logic related to calculating the charge for the P&I rule, including determining if and how much of the charge is waived, must be done by a Rule Processing algorithm.

The following sections describe the responsibilities of algorithms of this type.

NOTE:
Base plug-in. Click *here* to see the algorithm types available for this system event.

Calculation Basis

In many cases the P&I charges are calculated as a percentage of outstanding debt, referred to as the calculation basis. The calculation basis may simply be the amount of unpaid tax or it may be the amount of unpaid tax plus other unpaid charges, such as unpaid penalty or unpaid interest.

The calculation basis can change over time based on changes to the legislation and recalculation of historic info should use the calculation basis in effect at the time of the charge. A change in calculation basis requires a new P&I control and a corresponding new set of rules to be created and linked to the obligation type for the correct effective date.

The base product P&I rule business objects allow the user setting up the P&I rule to configure one or more debt categories that are used as the calculation basis. The balance by debt category is calculated prior to the call to the P&I rules and is passed in using the working FT collection.

Adjustment / Debt Category Configuration

When the penalty or interest charge is posted to the system to affect the taxpayer's balance, an adjustment is used. The adjustment type is defined on the P&I rule.

Each penalty and interest charge is identified by a debt category and this debt category is important for P&I calculation. Because adjustment type references a debt category, the P&I rules supplied with the base product do not capture the debt category explicitly. They derive the penalty or interest rule's debt category from its adjustment type.

NOTE:
No Duplication of Debt Categories. The base algorithms for calculating P&I expect that no two P&I rules for a P&I control refer to the same debt category (via its adjustment type).

Calculating New Charges and Canceling Incorrect Charges

The Rule Processing algorithm is responsible for calculating the appropriate charge for the current time period. However, because P&I is recalculated from the beginning every time P&I is called for an obligation, it's possible for the calculated charge for this time period already exists in the database.

Once the new charge is calculated and added to the running FT collection, the algorithm should perform logic to align existing and running charges and update the working FT collection appropriately. The base product logic does the following:

- When the current calculation matches the existing P&I charge in the database, the existing charge is added to the working FT collection.
- When the current calculation is more than the existing P&I charge in the database, the existing charge is included in the working FT collection and a new entry for the incremental increase is added to the collection.
- When the current calculation is less than the existing P&I charge in the database, the existing charge is marked to cancel in the working FT collection and a new charge for the new amount is added to the collection. In other words, the base algorithms do not create negative P&I charges.

There are situations where a P&I charge is calculated and after that charge was created, something occurs to cause this P&I charge to no longer be applicable for the taxpayer or for the time period or for the assessment. For example, perhaps a backdated payment is entered and the calculation basis for this time period is now zero. In this case although no new calculation occurs, existing charges must still be marked to cancel. To do this, the algorithm should perform the logic to align existing and running charges whenever it detects that no charge is required. In this case, the running charges will have no entries for the time period / assessment so any existing charges get marked to cancel in the working FT collection.

Use Rates To Define the Penalty or Interest Rate

There are two ways that the actual percentage may be applied to the calculation basis: using a rate schedule or using a rate factor.

- Use a rate factor if the calculation is a simple percentage applied to the calculation basis. The rate factor can contain an effective dated list of percentage rates. Using this method, the appropriate rate factor is defined when configuring the P&I rule and the rule processing algorithm is responsible for retrieving the rate factor value and performing the calculation.
- Use a rate schedule if the calculation is more complicated and you would like to take advantage of the calculation logic built into the various types of rate components for a rate. For example, if your charge has a maximum amount that can be charged (for example, a penalty that is 2.5% of the outstanding tax up to a maximum 25% of the outstanding tax), this can be configured easily using the **Maximum** rate component type. Using this method requires the rule processing algorithm to set up all the amounts that the rate schedule requires to successfully apply the rate. For the current example, it must supply the outstanding tax amount that is the basis of calculation and the maximum amount allowed. Then it should call the rate application service. The P&I rule defines the rate schedule, rate quantity identifier (RQI) for passing in the calculation basis and any other information needed to supply to the rate application service. In this example, the Not to Exceed percentage and an RQI for passing to rates the calculated Not to Exceed amount.

NOTE:
Rate Schedule vs. Rate Factor. The simple percentage calculation may also be performed using rates. However, it requires an administrative user to configure a rate schedule, rate version and a rate component in addition to the rate factor. Designing the algorithm to apply the rate factor directly saves the need for extraneous rates configuration.

Waiver Processing

As charges are being calculated for each period, rule processing algorithms must also determine whether or not these charges should be waived and enter appropriate entries in the working FT collection to represent the waived amounts.

The base product provides a separate algorithm for each *supported waiver type* in the system that should be plugged into any P&I rule whose charge may be waived. Each algorithm does the following:

- Determine if there is a waiver in effect for the current debt category / assessment. This information is in the Waiver information collection in the P&I - Internal Calculation Info data area assuming that the pre-processing algorithm to *Determine if Active Waivers Exist* is plugged in.
- The algorithm then does the appropriate logic to determine whether the amount should be waived based on the logic for the type of waiver.
- If the amount should be waived, an entry is added to the working FT collection. This ensures that subsequent balance calculations for this debt category do not include the amount that is waived.

No adjustments are created or canceled at this time. Post processing algorithms are responsible for that logic.

Sample P&I Rule Processing Logic

The following points highlight typical processing for a P&I rule processing algorithm that calculates a penalty or interest charge.

- The algorithm should first check whether the charge is applicable for the current period and for the current assessment. This information is populated in the base internalPenaltyAndInterestDataArea by a P&I pre-processing algorithm that may determine that this charge is only applicable for certain time periods and / or for certain assessments. If it's not applicable, the algorithm checks to see if existing charges need to be cleaned up using the 'align existing and running charges' logic (see below).

- If the charge is applicable, then the algorithm should apply the charge as per the business rules.
- If the P&I calculation is called with the P&I Detailed Forecast action, the algorithm should populate appropriate *calculation details* in the *calculation info* collection.
- The algorithm should include a step to *Align Existing and Running Charges*, which should compare the existing charges in the database (from the previous P&I update) to the current running P&I collection.
  - If the total of the current running charges is more than the existing charges a new charge should be added to the working FT collection for the difference.
  - If the total of the current running charges is less than the existing charges, the base recommendation is for the algorithm to mark the appropriate existing charges as "canceled" in the working FT collection by populating the Cancel Reason from the P&I Rule and to add a new entry for the new smaller amount.
  - If the P&I calculation is called with the P&I Detailed Forecast action, the calculation details from the running charges should be added to the working FT collection.

NOTE:
No updates. The rules processing algorithms provided in the base product do not do any updates to the database for new charges or cancellation of existing charges. These algorithms simply update the internal P&I information with new charges calculated and indicating any charge that should be canceled. Post processing algorithms are responsible for creating / cancelling financial transactions if Calculate P&I is called with the Calculate / Update action.

NOTE:
The base algorithms rely on data provided in the base P&I Main FT Info data area (**C1-PI-MainFtInfo**) and the base P&I internal calculation info data area (**C1-PI-InternalCalculationInfo**).

Post Processing

The system provides an *obligation type* plug-in spot **P&I post-processing** to perform any steps that need to occur at the end of all the P&I calculations.

In the base P&I calculations, post processing algorithms are used to do the actual creation and cancellations of penalty and interest adjustment and waiver adjustments in the system as determined by the rule processing algorithms. This is only applicable when the P&I algorithm is called with a **P&I Calculate / Update** action.

NOTE:
Base plug-in. Click *here* to see the algorithm types available for this system event and for more information about the behavior of this plug-in spot.

Final Updates

When penalty and interest is updated for an obligation, it is updated with a Calculation Through Date. This information is captured so that users reviewing the detailed balance for an obligation knows how recently the P&I calculation has occurred.

The balance details are calculated one more time to produce the final results for output.

Forecasting Penalty and Interest

Your implementation may include business processes that require forecasting of an obligation's balance to a current or future date without posting the P&I transactions:

- Sending a bill may include amount to pay if late, for example:
  - If you pay by January 1, please pay $2000
  - If you pay by January 15, please pay $2020
  - Etc,
- Sending a collection notice may include "please pay X amount by Y date" where the X amount is the forecasted balance on that date, which P&I included.
- If a payment is received at an account or taxpayer level where the taxpayer has several unpaid obligations, the payment algorithm that determines how to distribute the amount across obligations. When determining how much to direct to each obligation, the algorithm should forecast P&I to the payment event's effective date so that an accurate picture of the balance of each obligation is known.
- *Proposing scheduled payments* for a pay plan should include logic to forecast P&I to the future so that the scheduled payments cover P&I charges that will continue to accrue

The P&I calculation algorithm may be called to forecast. When forecasting, the algorithm perform all the same calculations, but does not store or cancel any adjustments to affect the obligation's balance.

FASTPATH:
The credit allocation zones on 360 degree view (on the Financial tab) and on control central (on the Account Information and Taxpayer Information tabs) allow a user to forecast P&I for a current or future date. Refer to *Credit Allocation Zone* for more details.

A service that calls P&I calculation with a "forecast" action may optionally provide financial transactions to use. If financial transactions are not provided, the algorithm retrieves the FTs currently linked to the obligation and forecasts P&I to the input date. If FTs are provided, the algorithm uses them in the P&I calculations. Supplying FTs allows a calling program to forecast the P&I with a "what if" scenario, for example "what if the taxpayer makes a payment on date X?". This is useful when proposing scheduled payments for a payment plan.

P&I Calculation Details

The product provides a page where a user can view the details of the penalty and interest calculation for a given obligation. However, because the calculation of each penalty or interest transaction is governed by the algorithm linked the P&I rule, the algorithm is responsible for supplying the details shown on this page. this section provides more information on how to configure the system to provide P&I details.

FASTPATH:
Refer to *Detailed P&I View* for more information about the page and its behavior.

Details Captured for Each P&I Transaction

The working FT section of the internal P&I data area used during the penalty and interest calculation include a collection of one or more *calculation info* strings that can be populated with the details related to the calculation of the charge. This information is passed out to the calling program using the main P&I data area.

The P&I Rules Define the Details

Because each P&I charge is calculated by an algorithm, it's the algorithm's responsibility to provide the details of the calculation.

The base product algorithms populate the detail based on their specific calculations. Click *here* to see the algorithm types provided by the base product and view their description details for more information.

If your implementation defines custom P&I rule algorithm to support your specific calculations and your users may wish to view the details of the P&I calculation using the Detailed P&I View page, you should populate one or more *calculation info* strings in the P&I data areas appropriately.

Details Provided During Forecast Only

The base product does not store the calculation details with the adjustments created for penalty and interest. The details are only provided by the base product algorithms when the Calculate P&I service is called with the **P&I Detailed Forecast** action.

P&I and Cash Accounting

It is common for tax authority to practice cash accounting for certain situations. For example, perhaps your implementation is a state revenue agency collects sales tax revenue on behalf of counties in your state. You may only distribute the revenue to the counties after the payment is made and not at the time of posting the assessment.

FASTPATH:
Refer to *Payables Cash Accounting* for general information about cash accounting functionality.

When dynamic credit allocation is practiced, where amounts are directed specifically to tax, penalty, interest and fees in a specific order, the allocation of credits to the various debt categories must be performed before any updates to the general ledger can be made as a result of cash accounting.

The base product provides an *obligation type* plug-in spot **Cash Accounting True Up** where an implementation may plug-in the algorithm that makes all necessary adjustments to the general ledger to transfer amounts from "holding" general ledger accounts to true payable accounts.

This algorithm is executed by the Calculate P&I service provided with the product when invoked with an "update" action. The service determines if the obligation type supports the P&I plug-in and if so, executes it. If the obligation type has a Cash Accounting True Up algorithm, it is then called, regardless of whether the obligation type supports P&I.

It means that any update to the system that impacts a taxpayer's financials where either P&I or Cash Accounting may be affected, the Calculate P&I service should be called. The business service is **C1-CalculatePenaltyAndInterest**.

NOTE:
Base plug-ins. Click *here* to see the algorithm types available for this system event and for more information about the behavior of this plug-in spot.

When Is P&I Updated?

There is no hard coded logic in the system to invoke the penalty and interest calculation. All requests to bring penalty and interest up to date are executed using plug-in logic. The base product provides logic to forecast P&I or bring P&I up to date when certain events occur. Your implementation may introduce new events that should cause P&I to be recalculated.

The following events in the system cause P&I to be brought up to date

- When a *tax form* posts or is reversed, transferred or adjusted, using an appropriate Form Rule. Basically any state transition of a form that causes adjustments to be created should include a rule to bring P&I up to date.
- When a payment is frozen or canceled, assuming that an appropriate plug-in is entered on the *account type*
- When certain types of adjustments are frozen or canceled, assuming that an appropriate plug-in is entered on the *adjustment type*. Refer to *Adjustments and Updating P&I* for guidelines related to configuring adjustments to update penalty and interest.
- When the *effective date* of a frozen payment is changed

- If your *obligation* is governed by the **C1-FilingPeriodObligation** business object, a post processing plug-in recalculates P&I if the override due date changes.
- When a *waiver* is activated or canceled
- An *overdue event* algorithm to calculate P&I is provided

A user may also request that P&I is brought up to date using the Credit Allocation zones on the *360 Degree View - Financial Information* portal and on Control Central on both the *Account Information* portal and the *Taxpayer Information* portal.

C1-CALPI - Bring P&I Up to Date

Besides the events in the system that may cause a recalculation of P&I, the system also provides a background process *C1-CALPI* to periodically bring P&I up to date for all obligations eligible for P&I processing. This is used as a "fall back" step to cater for obligations that may not be getting actively reviewed by another process.

The background process looks for all non-canceled non-closed obligations and invokes the calculate P&I service, which then invokes the appropriate P&I calculation algorithm for the obligation, if one is plugged in on the obligation type. Because it is executed for all the obligations, it should be scheduled infrequently, for example once a month.

Adjustments and Updating P&I

The base product provides adjustment type plug-ins to bring P&I up to date when an adjustment is frozen or canceled. Implementers should carefully consider which adjustment types require these algorithms.

- Certain adjustments are created from the P&I calculation, for example the penalty and interest charges and waiver charges. For these types of adjustments, the algorithms to recalculate penalty and interest when freezing or cancelling should not be plugged in on the adjustment type.
- Certain adjustments where P&I should be brought up to date are part of a business scenario where several adjustments are created for various charges. For example, when a tax form is processed, adjustments may be created for the tax assessment due and for the withholding credit and for refundable tax credits. The system should wait until all the adjustments are created before bringing P&I up to date, otherwise a lot of unnecessary calculations will occur. These types of adjustments should not reference the algorithms to recalculate penalty and interest when the adjustment freezes or is canceled.

The algorithms should be used for adjustments like manual penalties where penalty and interest is affected and the adjustment is not created as part of a greater process. The assumption is that an adjustment type would either have both algorithms or neither algorithm. In other words, if the creation of a given kind of adjustment affects penalty and interest, then cancelling that type of an adjustment should also affect penalty and interest.

Also note that when an assessment adjustment is canceled, all related penalty and interest adjustments should be canceled as well. The P&I calculation does not cater for cleaning up penalty and interest for canceled assessments.

NOTE:
Special service. When cancelling an assessment adjustment, the business service **C1-CancelAssessmentAdj** should be used. This will clean up penalty and interest adjustments for the canceled assessment.

Waivers

Waivers provide the ability to forgive or waive certain penalties, interest or fees. Sometimes waivers are referred to as abatements. You can waive a full amount or a partial amount (flat rates or percentages) or waive charges during a certain time period.

What is Waived?

A waiver is created for a specific assessment and a specific debt category for that assessment. For example, interest for the obligation's original assessment may be waived or the failure to pay penalty for the amendment may be waived.

Supported Types of Waivers

The base product provides business objects and appropriate algorithms out of the box that support the following types of waivers:

- One-time waivers. Waivers of this type are used to waive charges for a given assessment / debt category up to a specific amount.
- Effective dated waivers. Waivers of this type are used to waive charges for a given assessment / debt category that occur on or after a given start date. Waiver of this type may optionally specify an end date to waive charges for a specific period.
- Ongoing waivers. Waivers of this type are used to waive all charges for a given assessment / debt category.

Waivers vs. Exemption

A waiver is treated differently from an exemption in the base product algorithms. The assumption is that when an obligation is exempt from penalty and interest, calculations should not even be performed. Refer to *Eligibility* for more information.

For a waiver, the base product calculates the penalty or interest charge as normal and then creates appropriate waiver adjustments to offset the waived amounts. This allows an implementation to keep track of the amount of penalty and interest that has been waived.

Waivers and P&I Calculation

In order for waivers to affect P&I calculation, appropriate algorithms must be plugged in. The following points highlight the algorithms that are required to support waiver functionality:

- Information about existing waivers should be retrieved by a P&I pre-processing algorithm for use by the P&I rule processing algorithms. Refer to *Determine if Active Waivers Exist* for more information.
- If your implementation supports effective date waivers, the P&I pre-processing algorithm **C1-PI-PR-WDT** Adjust Calculation Periods for Effective Waivers should be plugged into any obligation type that supports P&I processing with waivers.
- Every P&I rule that calculates a charge that may be waived should include appropriate *rule processing waiver* algorithms plugged into the P&I Rule BO to adjust calculated charges in the internal P&I information data area based on existing waivers. Algorithms are provided to support one-time waivers, ongoing waivers and effective dated waivers. The base product P&I rule BOs are configured with the waiver algorithms already plugged in.
- *P&I post processing* algorithms **C1-PI-PS-WV1** Process One-time Waivers, **C1-PI-PS-WV2** Process Ongoing Waivers and **C1-PI-PS-WV3** Process Effective Dated Waivers are provide to create or cancel waiver financial transactions when calling P&I with an "update" action. These algorithms should be plugged into any obligation type that supports P&I processing for waivers.

NOTE:

No excess waiver. The base product algorithms assume that when a waived penalty or interest charge is later modified to cause the original amount to be less than the waiver, the waiver amount must also be adjusted accordingly. A waiver credit should never be in excess of the debt it's waiving.

Designing Your Waiver Options

The base product provides admin and transaction business object pairs for one-time waivers (**C1-WaiverTypeOneTime** and **C1-OneTimeWaiver**), ongoing waivers (**C1-WaiverTypeOngoing** and **C1-OngoingWaiver**) and effective dated waivers (**C1-WaiverTypeEffectiveDated** and **C1-EffectiveDatedWaiver**). Your implementation can add additional business rules to these BOs as required. If your implementation has waiver rules that aren't satisfied by one of the above business objects, you may create your own using the above as samples.

Designing Your P&I Control and P&I Rules

The base product provides a BO for P&I Control **C1-StandardPIControl**. Your implementation can add additional business rules to this BO as required. If your implementation has radically different requirements for your P&I controls, you can create a different business objects with their own business rules.

The base product supplies two BOs for P&I Rules.

- **C1-MonthlyChargePIRule** - this BO includes an algorithm that calculates the monthly charge with a Not to Exceed limit, using a rate schedule.
- **C1-SimpleCalculationPIRule** - this BO includes an algorithm that calculates a simple charge applying a rate factor to the outstanding calculation basis amount.

Your implementation can define add additional business rules to these BOs as required. If your implementation's P&I rules are not satisfied by one of the above business objects, you may create your own using the above as samples. The following points highlight the important configuration for this business object:

- Develop the appropriate rule processing algorithm to calculate the charge correctly.
- If *waivers* are applicable for the P&I rule's debt category, be sure to plug in appropriate rule processing algorithms to handle waiving the P&I rule's charge
- For any configuration required by the rule processing algorithm, consider whether it should be defined when configuring the P&I rule by a business user. If so, include the appropriate elements in the P&I rule business object's schema.
- Determine whether any P&I Rule *Pre-processing* algorithms are applicable for this type of rule.

Setting Up Penalty and Interest Options

The topics in this section describe how to set up the system to enable penalty and interest calculations.

Setting Up Account Types

In order for penalty and interest to be brought up to date when a payment is frozen or canceled, configure the account types with the following algorithms:

- System Event: **Payment Freeze**, Algorithm **C1-PI-PAYFRZ**
- System Event: **Payment Cancellation**, Algorithm **C1-PI-PAYCAN**

Setting Up Debt Categories

Define *debt categories* for the following:

- One for each separate penalty or interest charge that is calculated by a system P&I rule.
- One for any other type of debt that is not calculated by P&I calculations, but is used in the calculation basis for one of the P&I rules.
- One for any other category of debts. For example, overpayments. When an obligation is overpaid, at some point the *overpayment* amount is reduced by one or more methods, such as minimum amount write down, carry forward to a future obligation, refund, etc. The adjustments created to reduce an overpayment are debits and therefore require a debt category. The suggestion is to create a special "Overpayment" debt category for these types of debts.

Setting Up Debt Category Priorities

Define appropriate as *debt category priorities* required by your implementation's business rules. Refer to *Debt Categories and their Priorities* for more information.

Setting Up Adjustment Types

Adjustment types are needed for posting P&I charges.

- A separate adjustment type must be created for each debt category that is part of the P&I calculation.
- Each adjustment type created should use the **Penalty & Interest** adjustment type category.

Adjustment types are needed for posting waivers.

- A separate adjustment types must be created for each debt category whose P&I charges may get waived.
- Each adjustment type should refer to the **Waiver** adjustment type category.
- Each adjustment type should define the **C1-WAIVR** characteristic type as a valid template characteristic. This is used to reference the waiver that caused the adjustment to be created.

An adjustment type is needed for transferring cash accounting amounts from a "holding" general ledger account to the true "payable" account.

For any adjustment that is a manual penalty and should cause P&I to be recalculated, configure an adjustment freeze algorithm and corresponding adjustment cancellation algorithms to bring P&I up to date. The product provides the adjustment freeze algorithm *C1-ADF-CALPI* and the adjustment cancellation algorithm *C1-ADC-CALPI* to use.

Setting Up Adjustment Cancel Reasons

Define appropriate *adjustment cancel reasons* to use when cancelling P&I adjustments and based on recalculation and cancel reasons to use when cancelling waiver adjustments.

Setting Up Rate Factors

Create appropriate *rate factors* that define the percentage to use when calculating your various P&I charges.

- If you are defining P&I rules using the base BO **C1-SimpleCalculationPIRule**, the P&I rule requires a rate factor to use when applying the charge.

- If you are defining P&I rules using the base BO **C1-MonthlyChargePIRule**, the P&I rule expects to use a Rate Schedule. You may choose to define rate factors for configuring the charges used in the specific rate components defined in this rate schedule.

Setting Up Rate Quantity Indicators

Create appropriate *RQIs* required for your P&I rules. If you are defining P&I rules using the base BO **C1-MonthlyChargePIRule**, you need an RQI for the following information to pass into rate application:

- The calculation basis amount
- The not to exceed amount

Setting Up Rate Schedules

Create appropriate *rate schedules* and their components to calculate the charges appropriately.

If you are defining P&I rules using the base BO **C1-MonthlyChargePIRule**, you need a rate schedule with rate components that calculate the charge by applying an appropriate percentage to the calculation basis passed in using the RQI defined above. If a Not to Exceed amount is applicable, there should also be a maximum rate component that adjusts the calculated charge based on the Not to Exceed amount.

The following are some additional notes related to the setup of this rate:

- The charges are not expected to prorate for short or long periods. To accomplish this, an appropriate frequency must be configured for the rate schedule. Because the P&I rule is expecting to apply monthly charges, the frequency may be set to have 1 period annually and 365 days for the minimum and maximum offset. Refer to the demonstration data for an example.
- The individual rate components should all be set as follows:
    - FCPO is checked
    - The Result Type is **Charge**
    - Rounding Precision is set to the maximum allowed to ensure that the maximum calculation precision is returned.
    - Create Bill Line is checked
    - FCPO Retention Rule is set to **Retain amount on bill line**
    - To support *calculation details*, appropriate description on bill values should be populated and the Print flag checked.

Setting Up P&I Control

A **P&I Control** is created to hold all the rules that work together to calculate automated / ongoing penalty and interest. The P&I control includes a list of **P&I Rules** that make up the individual calculations.

To set up P&I Control, select **Admin Menu** > **P&I Control**.

The topics in this section describe the base-package zones that appear on the P&I Control portal.

P&I Control List

The P&I Control List zone lists every P&I Control. The following functions are available:

- Click a *broadcast* button to open other zones that contain more information about the adjacent P&I Control.
- Click the **Add** link in the zone's title bar to add a new P&I Control.

Actions

This is a standard actions zone. The **Edit, Delete** and **Duplicate** actions are available.

P&I Control

The P&I Control zone contains display-only information about a P&I Control. This zone appears when a P&I Control has been broadcast from the P&I Control List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_PI_CTRL*.

P&I Control Obligation Types

This zone lists obligation types currently linked to the broadcast P&I control.

P&I Rules

This zone lists the P&I rules associated with the broadcast P&I control. Click on a P&I rule in the list to view its details in the *P&I rule portal*.

If the P&I control is in **pending** status, Edit and Delete actions are available for each P&I rule. In addition you may add another P&I rule using the **Add** link in the zone's title bar.

P&I Control Log

This is a standard *log zone*.

Setting Up P&I Rules

This portal is provided to view and maintain details of a given P&I rule. This portal is not available from a menu. You navigate to this portal by drilling into a specific P&I rule from the *P&I control* portal.

The topics in this section describe the base-package zones that appear on the P&I Rule portal.

Actions

This is a standard actions zone. The **Edit** and **Delete** actions are available.

P&I Rule

The P&I Rule zone contains display-only information about a P&I rule.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_PI_RULE*.

Setting Up Obligation Types

Each obligation type where penalty and interest charges are applicable must be configured appropriately:

- Define the default *debt category priority*.
- It should include appropriate P&I controls that define the P&I rules that are in effect
- It should include appropriate P&I calculation algorithms. The following provides a list of the plug-in spots that must are required for P&I calculations. Unless noted, base algorithms are provided. If the functionality provided in the base algorithms does not satisfy your implementation's business rules, you may define your own:
  - P&I Calculation
  - Determine Balance Details
  - Retrieve FT Details
  - P&I Prepare Periodic Balance
  - P&I Pre-processing
  - P&I Post Processing - algorithms are provided for all algorithm types except for the *cash accounting* algorithm type. That one needs to be configured with the appropriate adjustment type.
- In addition to the required algorithms above, if your implementation's business rules include P&I Eligibility requirements, provide appropriate algorithms.

Setting Up Waiver Types

To open the Waiver Type zone, select **Admin Menu** > **Waiver Type**.

Waiver Type List

The Waiver Type List zone lists every waiver type. The following functions are available:

- Click a *broadcast* button to open other zones that contain more information about the adjacent waiver type.
- Click the **Add** link in the zone's title bar to add a new waiver type.

Waiver Type

The Waiver Type zone contains display-only information about a Waiver Type. This zone appears when a Waiver Type has been broadcast from the Waiver Type List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_WAIVER_TYPE*.

Waiver Type Log

This is a standard *log zone*.

Chapter 11

Defining Asset Options

Assets and valuations are central objects for information that’s used in billing taxes..

This chapter describes options related to assets, asset ownership and valuations. The following sections describe how to set up the tables that control this functionality.

The Big Picture of Assets

Asset records store data related to assets/property that may be subject to billing. Besides information about the asset itself, the following information are also captured:

- Billable address associated with the asset. For real property assets, this is the location of the asset. For other types of assets such as vehicles and boats, the address may be related to the address of the main taxpayer for the asset.
- One or more external identifiers for the asset.
- Current taxpayer who is responsible for paying taxes that billed for the asset.
- Persons related to the property besides the taxpayer.
- Parent assets. This applies to cooperatives, where there is a main/parent asset that gets billed on behalf of the child assets/ units that belong to the cooperative.

The following sections provide more information.

The Asset Model

Asset records exist for the primary purpose of billing taxes.

The base product has the following assumptions for configuring the Asset and the related entities:

- For the asset to be billable, it needs to be linked to a tax role. The tax role is created and stays with the asset for its billable life. The base product supplies asset-based tax roles, including a specific one for real property tax roles. Refer to the **C1–TaxRoleRealProperty** and **C1–TaxRoleAsset** base business objects for more information.

- The tax role’s related account and person are created for the asset/property and are not real persons or businesses. This person/account is different from the owner(s) of the property, which do get captured as persons/businesses, but are linked to the asset through ownership history records. The special account and persons (for billing purposes) can be set up using the **C1–BillAssetAccount** and **C1–BillAssetPerson** business objects that the base product provides. Refer to these business objects for more details.
- The tax role’s obligations are maintained automatically as the tax role is created and when the tax role’s start or end date changes. Refer to*Tax Roles Can Maintain Obligations* for more information.
  The base product provides a business object for obligations linked to tax roles that are billed. Refer to the **C1–BilledObligation** business object for more details.

Asset External Identifiers

An asset may have one or more external identifiers. Some types of identifiers have simple values. Other types could specify a value with multiple components — e.g. property identifiers that specify “square, suffix, lot”.

An asset external identifier type references a type which controls how the asset identifier value is validated and processed.

The asset’s type defines the valid external identifier types that can be specified for that type of asset.

Asset Addresses

Many assets are billed based on attributes related to their location.

- For real property assets, the billing address is usually the property’s address, which is static.
- For vehicles, the billing address changes as the vehicle’s location changes

Some assets may not require a billing address.

Asset records in the system can include a collection of billing addresses. This is an effective-dated collection, to allow for changes over time.

The asset’s tax role can also capture a collection of addresses. This can be used for any miscellaneous addresses, if needed.

Owners of the asset are captured as persons linked to assets. If these person records specify addresses, those can also be used for sending information to specific owners.

Asset Ownership

An asset can be associated with one or more owners. Owners are set up as persons linked to the asset through an effective-dated Asset Ownership history record. One of these owners is designated as the primary and each owner can be designated a percentage of ownership.

When an asset/property is sold, changes in the ownership are reported by an external system, which results in corresponding changes to the ownership history information.

Billing an Asset

The product supports two models for tracking debt and associating a taxpayer to an asset:

- Asset-based model where the tax role is created and stays with the asset for its billable life. The tax role's account and person are created for the property and do not represent real persons or businesses. The owners of the property are captured as person records but are linked directly to the asset using an effective-dated ownership history record.

- Standard "account" model where the taxpayer is linked to an account and the account is linked to a tax role (that links to an asset) on down to the financial transactions. When ownership changes, the tax role is end dated and the account is responsible for the debt incurred while it was the owner of the asset. I.e., for the debt incurred during the tax roles' effective dates. The new owner gets a new tax role.

NOTE:
The second option described above is not a common scenario for real property taxes. While the data model supports it, out-of-the-box algorithms to set up the relationships for ownership changes (in this scenario) are not supplied. Your implementation will need to provide them.

Refer to *The Big Picture of Bills* for more information on billing.

Related Persons

One or more persons aside from the owner(s) can be linked to the asset. For instance, a real property asset may be associated with a facilities manager.

Refer to the inline help on *Asset — Main* for details on how to these related persons are linked to the asset.

Cooperatives

A cooperative operates for the benefit of its members on a not-for-profit basis in order to provide the goods and services members need at the lowest practical cost. Members/shareholders own the cooperative and participate equally in the governance of the cooperative.

For assets that are part of a cooperative, a parent asset is linked to child assets. The parent asset receives the bill, which is calculated based on the number of units. Valuation records containing information needed for billing are linked to the parent asset. The billing system may or may not store specific valuation information for the child assets units.

If a cooperative is dissolved or converted (e.g. to a condominium), the parent asset is end-dated and the individual units/ assets become billable.

The Asset object in the system allows for parent/child asset relationships that cooperative and cooperative unit assets need. The supplied **C1–RealPropertyAsset** base business object supports the ability to define an asset as either a cooperative or a unit in the cooperative using the Special Role lookup.

The Big Picture of Valuation

Valuation is an appraisal or estimation of an asset’s value as of a given period of time.

For real property, in particular, valuations provide the assessment value for a specific revenue year. The property’s value is assessed based on a number of key factors like the property’s use, age, square footage, amenities, current market conditions, etc.

Valuations are initiated for any of the following reasons:

- New property
- Changes to existing property — e.g. construction/improvement, splits/merges, demolition, etc.
- Ongoing/periodic valuation
- Property is sold and/or changes ownership

Valuation information is typically maintained at an external system and any information that's needed for billing is interfaced to the billing system. Within a specific revenue period, one or more valuations may exist for the asset, depending on changes that occur for that asset.

The following sections provide more information.

Valuation

A valuation record captures the details of an asset's assessed value for a specific revenue period. Changes to the asset itself or to the asset's ownership within a specific revenue period typically results in the creation of additional valuations for that period. Such valuations, in turn, feed into billing — e.g. for generating supplemental bills. The system supports multiple valuations for the asset and for the same revenue period.

The valuation record can specify the reason for creating that valuation, as well as the external source that sent the information.

A valuation's lifecycle is simple. Once the valuation is linked to a valid asset in the system, it persists in the Active state and stays there unless there is a reason to manually cancel the valuation. The base product supplies the **C1–Valuation** business object, which has this lifecycle.

Valuation Details

The bulk of the information in valuations are in the valuation details. A valuation detail could specify a simple number, a percentage or a monetary amount. For real property valuations, for instance, a valuation could contain a large volume of details about the property's physical attributes (square footage, number of units etc.) and miscellaneous charges for the property.

Each valuation detail references a value detail type, which determines whether or not the supplied value is valid. It also determines whether or not a specific type of detail can be specified multiple times within the same valuation.

The valuation type controls the value detail types that are valid for a type of valuation. When adding a valuation record, the value detail types are filtered to only those that are valid for the selected valuation type.

The base package provides the **C1–ValueDetailType** and **C1–ValuationType** business objects that support the above functionality. Refer to these business objects for more details.

Billing Valuation Details

A valuation detail's type controls whether or not the detail is billable. When a value detail type is configured to create bill charges, the description that appears for the resulting bill charge and the distribution code to use for that charge can also be specified.

When a valuation goes to the Active state, it is considered billable. Billing calculation rules could pull in the billable details of the asset's active valuations for the specific revenue period being billed.

Refer to *The Big Picture of Bills* for more information on billing.

Uploading Valuations

Valuation data is commonly maintained in an external system. For instance, real property valuations can come from mass appraisal systems that gather data for the purposes of determining property value and market analysis.

With any type of asset (e.g. real property, personal property, vehicle, etc.), valuations contain a collection of valuation details, most of which are used in bill calculations.

The following sections describe concepts about uploading valuations into the system.

Direct Upload of Valuations

Valuation uploads tend to be high volume and each uploaded valuation record can contain a large number of details. For instance, real property valuations could have very detailed information about the land, improvements, exemptions, etc.

However, valuation upload processing is simpler than most upload processes in the system. In a lot of the cases, valuation upload is just a two-step process: determine the asset and add the valuation record with its details. Aside from checking that a valuation's related asset exists, there is not much else to validate during the upload process. The assumption is that the main system of record (i.e. the external system that interfaces valuation data) has the responsibility of ensuring that valuation data is valid prior to the upload.

Valuation record structures also tend to be static and hardly change from time to time.

It is for the above-mentioned reasons that the base product does not provide a separate 'staging' object for uploading valuations. The supplied **C1–Valuation** base business object is designed for use in upload processes that load records directly into the Valuation table through a series of BO adds.

NOTE: The upload records should go through BO processing, to make sure that all BO rules, specially the pre-processing step of determining the asset, are executed.

Determining the Asset

A valuation exists for a specific asset. When valuations are uploaded from an external system, it's common for these records to specify the related asset's external identifier. Such records will need to go through the process of finding the related Assets in the system using the supplied external identifiers.

The **C1–Valuation** base business object has pre-processing logic to determine a valuation's related asset. If it is unable to do so, it will put the valuation into the Error state.

Reprocessing Valuations in Error

There are two common reasons for valuations to go into error:

- The related asset exists in the system, but the supplied external identifier is invalid/incorrect. This will usually require a user to investigate and fix the issue. In exceptional cases where there is a general issue with a whole batch of uploaded valuations, the batch might have to be re-uploaded and the existing valuations in error are canceled.
- The related asset does not yet exist in the system. This can happen when asset information is uploaded/entered into the system separately and the valuation information happens to get uploaded first. The **C1–Valuation** base business object has logic to monitor valuations in error, in case information about the related assets become available in the system.

Setting Up Asset Options

This section describes tables that must be set up before you can define assets and valuations.

Setting Up Asset External ID Types

Asset External ID Type controls how a specific type of asset external identifier is structured and processed.

To set up an Asset External ID Type, select **Admin Menu** > **Asset External ID Type**.

The topics in this section describe the base-package zones that appear on the Asset External ID Type portal.

Asset External ID Type List

The Asset External ID Type *List zone* lists every asset external ID type. The following functions are available:

- Click a *broadcast* button to open other zones that contain more information about the adjacent asset external ID type.
- The standard actions of **Edit, Delete** and **Duplicate** are available for each asset external ID type.

Click the **Add** link in the zone's title bar to add a new asset external ID type.

Asset External ID Type

The Asset External ID Type zone contains display-only information about an Asset External ID Type. This zone appears when an Asset External ID Type has been broadcast from the Asset External ID Type List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *C1_ASSET_EXT_ID_TYPE*.

Setting Up Asset Types

Asset Types contain the rules that control how a specific type of asset is processed.

To set up an Asset Type, select **Admin Menu** > **Asset Type**.

The topics in this section describe the base-package zones that appear on the Asset Type portal.

Asset Type List

The Asset Type *List zone* lists every asset type. The following functions are available:

- Click a *broadcast* button to open other zones that contain more information about the adjacent asset type.
- The standard actions of **Edit, Delete** and **Duplicate** are available for each asset type.

Click the **Add** link in the zone's title bar to add a new asset type.

Asset Type

The Asset Type zone contains display-only information about an Asset Type. This zone appears when an Asset Type has been broadcast from the Asset Type List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_ASSET_TYPE*.

Setting Up Value Detail Types

Value Detail Types contain the rules that control how valuation details are processed. To set up a Value Detail Type, open **Admin Menu** > **Value Detail Type**.

Value Detail Type Query

Use the *query portal* to search for an existing value detail type. Once a value detail type is selected, you are brought to the maintenance portal to view and maintain the selected record.

Value Detail Type - Main

This portal appears when a value detail type has been selected from the Value Detail Type Query portal.

The topics in this section describe the base-package zone that appears on this portal.

Value Detail Type

The Value Detail Type zone contains display-only information about selected value detail type.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *C1_VAL_DTL_TYPE*.

Setting Up Valuation Categories

Valuation categories allow for classification of valuation types into broad categories — e.g. fixed assessment, exemption, etc.

If valuation categories apply, identify these categories and configure them as Valuation Category extendable lookup values.

- Open **Admin Menu** > **Extendable Lookup**.
- Search for and select the **Valuation Category** extendable lookup business object.
- The list of existing valuation categories are displayed in a standard *List zone*.
- Choose an existing valuation category to view, edit, delete or duplicate.
- Use the **Add** link in the zone header to create a new valuation category.

Setting Up Valuation Types

Valuation Types contain the rules that control how a specific type of valuation is processed.

To set up a Valuation Type, select **Admin Menu** > **Valuation Type**.

The topics in this section describe the base-package zones that appear on the Valuation Type portal.

Valuation Type List

The Valuation Type *List zone* lists every valuation type. The following functions are available:

- Click a *broadcast* button to open other zones that contain more information about the adjacent valuation type.
- The standard actions of **Edit, Delete** and **Duplicate** are available for each valuation type.

Click the **Add** link in the zone's title bar to add a new valuation type.

Valuation Type

The Valuation Type zone contains display-only information about a Valuation Type. This zone appears when a Valuation Type has been broadcast from the Valuation Type List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *C1_VALTN_TYPE*.

Setting Up Valuation Reasons

Identify the various reasons for creating a valuation and configure those reasons as Valuation Reason extendable lookup values.

- Open **Admin Menu** > **Extendable Lookup**.
- Search for and select the **Valuation Reason** extendable lookup business object.
- The list of existing valuation reasons are displayed in a standard *List zone*.
- Choose an existing valuation reason to view, edit, delete or duplicate.
- Use the **Add** link in the zone header to create a new valuation reason.

Setting Up Valuation External Sources

Identify the various sources for valuations and configure them as Valuation External Source extendable lookup values.

- Open **Admin Menu** > **Extendable Lookup**.
- Search for and select the **Valuation External Source** extendable lookup business object.
- The list of existing valuation external sources are displayed in a standard *List zone*.
- Choose an existing valuation external source to view, edit, delete or duplicate.
- Use the **Add** link in the zone header to create a new valuation external source.

Chapter 12

Defining Overpayment Processing Options

An overpayment occurs when the payment(s) exceeds the liability in an obligation, causing the obligation's balance to become a credit. Not every obligation that is in credit will get processed by an overpayment process. This will typically depend on the tax type and the authority's rules.

The Big Picture of Overpayments

For return-based taxes, such as individual income tax, a credit balance will be considered an overpayment only once a return is filed. Any payments or credits received in advance of a tax return are not considered overpayments until the tax form is processed, and an assessment is created. If a tax return is not filed, the obligation will be in a credit balance until the tax authority reviews it.

For billing-based taxes, different rules may apply to determine if an obligation is overpaid.

The following are common scenarios that would result in an overpayment:

- A majority of taxpayers receive refunds for their individual income tax filing because employer withholding tends to exceed the probable liability.
- Recalculation of tax, penalty, interest or fees that results in a reduced liability.

Not all tax types have overpayments. Assessments for sales tax and withholding tax are seldom overpaid because the taxpayer pays for an activity that has already happened. If an overpayment results, it is usually due to a math error on the form.

The valid overpayment types for a given tax type must be configured. This list may be configured on both the overpayment process type page (list of tax types for which the overpayment process type is valid) and on the tax type page (list of valid overpayment process types). Also note that the system provides a “usage” flag that may be used for any functionality where a default overpayment process type should be used for a given scenario. Refer to *Setting up Tax Types* for more information.

An Overpayment Process Can be Created Automatically or Manually

An overpayment process can be created in a variety of ways:

- As a result of posting a return. A form rule related to the posting system event could be used to create an overpayment process if the form reports a refund. The base product provides a tax form posting form rule to create an overpayment. Refer to the business object **C1-CreateOverpayment** for details.
- An account monitor rule can be designed to create an overpayment process when the obligation's balance is a credit and a return has already been filed. The base-package provides the algorithm **Initiate Overpayment Process** (*C1-CC-INITOP*) that can be plugged in on the Collection Class Overdue Rules. For more information on how account monitor rules can be configured for overpayment, see *Overdue Rules Are Embodied In Algorithms*.
- An overpayment process can be created manually by selecting **Main Menu** > **Accounting** > **Overpayment Process**.
- Implementations may supply algorithms to create overpayments when other events occur.

An Overpayment is Typically Reviewed

When an overpayment is identified, it typically goes through a review process, which may include a combination of automated checks and user review and/or approval.

Examples of review activities include:

- Processing Rules - these can include checking for a valid mailing address, verifying bank account details.
- Compliance Rules - these include specific criteria defined by the tax authority for overpayments that require additional review.
- Minimum Balance - the overpayment amount is compared against a minimum threshold amount to determine whether the overpayment should be processed.
- Risk-based thresholds and overrides - if the overpayment amount is greater than a predefined maximum threshold, it may require special handling and additional approval.
- Suppression - an overpayment may be suspended or put on hold if an account is under investigation for other debt or another reason.
- Separation of Duty - the tax officer approving the refund must be different from the tax officer who changed the account's bank information.

The review rules may differ with each tax authority and tax type. However, the end result is the same: to determine whether the overpayment can be processed further.

Not all Overpayments Result in a Refund

Before an overpayment is refunded to the taxpayer, some or all of the overpaid amount may be offset against other existing debt, carried forward to a future period, and/or contributed to designated charitable funds or organizations.

Minimum Amount Write Off

Implementations may opt to write off (sometimes called write down) credit amounts that are within a certain threshold, defined on the overpayment process type.

Refer to the base business objects for algorithms supplied to write off small credit amounts.

Offset

When an overpayment is offset, the credit is used to extinguish other debts. The order of allocation of overpayment amount to debt depends on the tax authority's rules. Typically, the credit is applied in the following order:

- Within the same assessment
- Other assessments within the obligation
- Other obligations under the same account
- Other accounts for the taxpayer

In some cases, a tax authority may also allow offsetting against debt from other agencies. This type of debt is called external debt.

Where offset is possible, it is done before attempting to contribute, carry forward or refund. Offset is not allowed on future obligations, but a carry forward is done instead.

Refer to the base business objects for algorithms supplied to offset credit to other obligations.

Calculating Interest

Before any other action is taken, it might be required to calculate interest on the overpayment amount. The calculation rules will differ by tax authority and by tax types. In addition, the decision of whether to calculate interest before or after offsetting other debt may differ based on an implementation’s business rules.

Refer to the base business objects for algorithms supplied to calculate interest.

Carry Forward

Taxpayers may opt to pay an amount of an overpayment to a future period. This is known as carry forward. This is common to individual income tax filing, where the taxpayer can indicate on the form a designated amount that will go to the next filing period (next tax year).

A tax authority's rules may designate certain tax types to only allow overpayments to be carried forward. Some common examples include excise or sales and use taxes. In these cases, the entire overpayment amount is carried forward.

Note that the base form rule currently supplied to create an overpayment does not support supplying carry forward information to the overpayment process. In addition, the Standard Overpayment Process business object does not currently support carry forward.

Contributions

Most tax forms provide an option to contribute to a charitable fund/organization. The taxpayer indicates a specific amount to contribute to each selected fund/organization. The contribution is made only if an overpayment exists.

Note that the base form rule currently supplied to create an overpayment does not support supplying contribution election information to the overpayment process .In addition, the Standard Overpayment Process business object does not currently support contributions.

Refunds

Refunds result after attempts to offset, contribute and/or carry forward. The credit amount is given back to the taxpayer in the form of a paper check or a direct deposit. The taxpayer could either authorize a one-time direct deposit by putting bank account information on the tax return or indicate a recurring direct deposit for all refunds. In the case of a recurring direct deposit, the bank account information stored for the taxpayer on the *account* will be used.

Configuration on the overpayment process type indicates the mechanism for issuing a direct deposit. The options are Bank Event (recommended) and "credit" payment that triggers an ACH staging record. The credit payment is legacy functionality and not recommended for new releases.

FASTPATH: Refer to *The Big Picture of Bank Events* for more information.

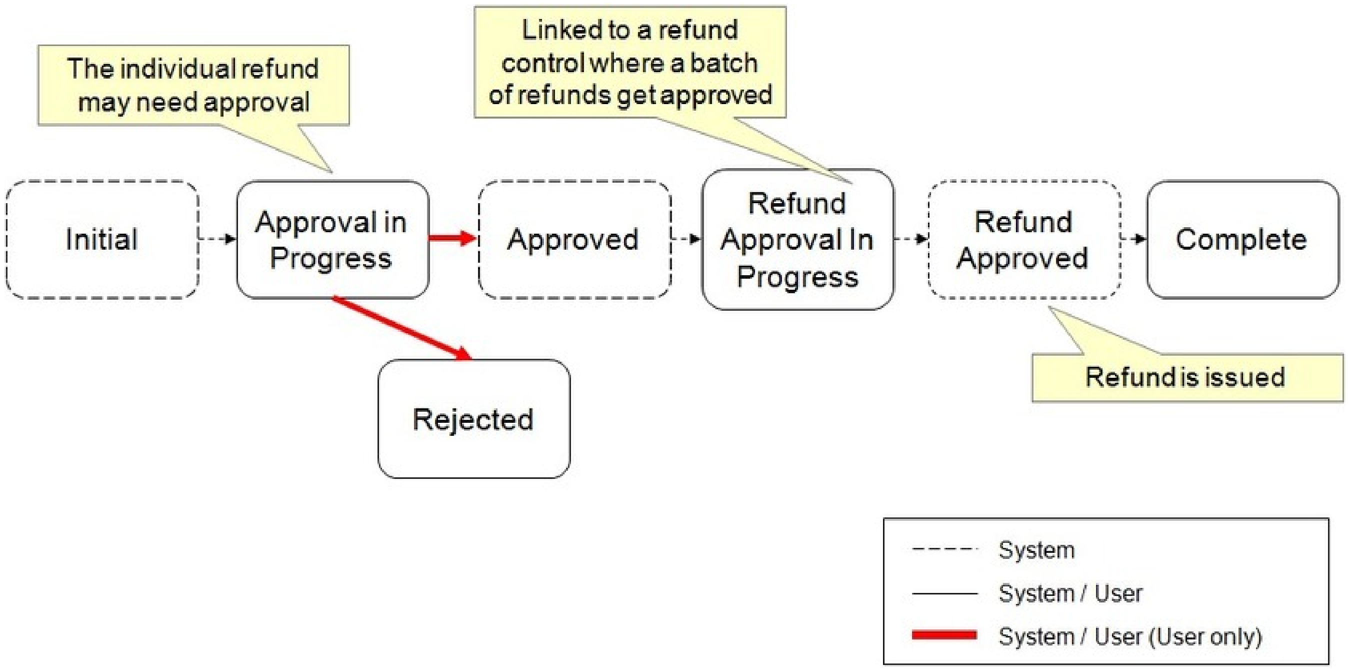
Standard Overpayment Process

The base product includes the base business object **C1-OvrpyProcStandard**, which is designed to cater for most overpayment process. Further details in this section are applicable to this business object.

NOTE: The product also provides a more specific business object that is not recommended for future use but is still supported for upgrade purposes. Some of the logic in this section applies to this legacy business object, but not all. Refer to *Individual Taxpayer Overpayment Process* for more information.

The Overpayment Process Lifecycle

The lifecycle of the overpayment process depends upon the configuration of the associated business object. The Standard Overpayment Process BO (**C1-OvrpyProcStandard**) has a lifecycle that is expected to be used by most implementations. The lifecycle looks very complicated when viewing all the valid state transitions. The following diagram highlights the typical "positive" flow. Refer to the business object — summary tab for this business object for the full lifecycle diagram.


The individual refund may need approval
Linked to a refund control where a batch of refunds get approved
Initial
Approval in Progress
Approved
Refund Approval In Progress
Refund Approved
Complete
Rejected
Refund is issued
System
System / User
System / User (User only)

- After the overpayment is created in the Initial state and found to be valid, it transitions to Approval in Progress to assess if any approvals are needed.
- The threshold amounts and approval roles are defined on the overpayment type. One or more approvers may be required. A user can Approve or Reject the overpayment.

NOTE: Refer to *Overpayment Process Approval* for more information.

- Once all the required approvals have been obtained, certain actions will take place such as offset.
- Assuming there is still an amount to refund, the record transitions to the Refund Approval in Progress state where it waits to be linked to a refund control.

NOTE: Refer to *The Big Picture of Refund Control* for more information.

- Once the refund control is approved, a background process transitions the overpayment to Refund Approved, where the refund is issued and finally the status is Complete.

The following points highlight additional detail about the lifecycle:

- Validation issues:
  - After the overpayment is created, it executes overpayment validation plug-ins defined on the overpayment process type. If any issues are found, they are captured and the status goes to Issues Detected. From there a user may resolve the issue and reprocess or may Reject the record.
  - An addition validation check may be performed after it is clear that there is an amount to refund. The refund approval in progress state is configured to call refund validation algorithms on the overpayment process type. This allows for checking for valid bank information, for example. If any issues are found at this stage, the overpayment transitions to Issues Detected. Otherwise, it waits to be linked to a refund control.
  - The refund approved state once again calls the refund validation plug-ins on the overpayment process type. This is to cater for the possibility that a taxpayer has updated their banking information while the refund was awaiting refund control approval. If any issues are found at this stage, the overpayment transitions to Issues Detected. Otherwise, the refund is issued appropriately.
- Several conditions are checked from any state where the overpayment process may transition from one where it was sitting and waiting for some action to occur. These include On Hold, Issues Detected, Approval in Progress and Refund Approval in Progress.
  - The overpayment itself is running algorithms that may reduce the credit. If any of these algorithms reduce the credit to zero, the status transitions to **Complete**.
    - Write off a small amount
    - Offset the credit to debt on another obligation
    - Issue the Refund
  - The obligation's balance may have changed independent of the overpayment process. This condition is checked from any state where the overpayment process may transition from one where it was sitting and waiting for some action to occur. The action taken depends on the status and the change in the balance:
    - If the balance change is detected during the transition out of On Hold or in Issues Detected, if the balance is no longer credit, the process completes. Otherwise, the new balance is checked again against the minimum write off. If it's greater than that threshold, it continues to the approval step. (Otherwise it's written off).
    - The balance change may be detected during the approval process. Note that the user interface for the base Standard Overpayment Process BO displays a message if the obligation’s current balance differs from the balance captured by the overpayment. if the balance is no longer credit, the process completes. If the balance is below the minimum write off amount, the amount is written off. If the balance is less credit or if it is more credit, the action depends on if the process is in the middle of a multi-level approval process (controlled by configuration on the overpayment process type).
      - If there is a single approver or if the balance change is detected at the final approval of a multi-level approval, the process continues if the balance is less credit or is more credit, but still within the threshold band of the

        current approver. If the credit is more credit and pushes the amount into a higher threshold band, then the process requires additional approval based on the new threshold band.

        - If balance change is detected during a multi-level approval, where future approvers are needed, the process continues if the new balance is in the same threshold band as the previous balance. It means that the same approval hierarchy applies. However, if the new balance is in a different threshold band than the original amount, then the process adjusts the future approver list appropriate for the new balance.

    - If the balance change is detected during the refund approval process (after the refund control is approved), if the balance is no longer credit, the process completes. If the balance is below the minimum write off amount, the amount is written off. If the balance is still credit but is less credit than before (but still above the minimum write off amount), the assumption is that the amount is still valid and that the refund can be issued without further review. If the amount is a greater credit, it should be reviewed again by a user. It is routed back to approval in progress for the appropriate final threshold level.

  Any change detected during the lifecycle to the obligation's balance where it is brought to zero or is a debit amount, the BO transitions to Completed.

  - A suppression may have been logged for the obligation. After each transition that occurs after a waiting period, an algorithm checks if a suppression record exists for the obligation. If so, the process transitions to On Hold. In addition, implementations may introduce other conditions that cause the overpayment to transition to On Hold.

- If a process that initiated the overpayment (such as a tax form) is subsequently canceled, the overpayment may also be Canceled. This can happen even after the overpayment process is complete. Algorithms executed when entering the complete status may attempt to undo any actions that the overpayment did, such as cancel write-offs or offsets or attempt to cancel the refund, if possible.

Overpayment Process Validation

The overpayment process type includes two system events for validation algorithms: overpayment validation and refund validation. Overpayment validation should be used for checking conditions that are applicable for any overpayment process. For example, verifying that the obligation has a credit balance. Refund validation should be used for validation that is only applicable if there is a refund to be issued, for example validating the bank information.

Having validation algorithm on the overpayment process type allows for different types of overpayment process to have different validation. For example, perhaps an obligation for a tax type where returns are filed must have a valid tax form posted prior to any overpayments being issued.

The base business object Standard Overpayment Process **C1-OvrpyProcStandard** calls the validation algorithms on the overpayment process type when checking the validity of the record at various state transitions.

Overpayment Process Approval

An overpayment process typically requires one or more levels of approval before any financial activity can take place.

An Overpayment Process Typically Requires Approval

If the overpayment process type defines an approval hierarchy, users will need to approve the overpayment process in order for the process to continue. When the overpayment process is in the **Approval in Progress** state, two action buttons will appear: **Approve** and **Reject**.

If the user chooses to reject the overpayment process, they will be prompted for a reject reason.

Approval Is Controlled By Its Type

The overpayment process type contains the rules that define if and how an overpayment process is approved. If an overpayment process type does not reference an approval details, the related overpayment processes do not require third-party approval before they proceed. If an overpayment process type references approval detail, the approval hierarchy defines if the overpayment process requires approval and who the authorized approvers are. For example, an overpayment process type can be configured with the following approval hierarchy:

- Overpayment Processes <= $10 do not require approval
- Overpayment Processes > $10 and <= $100 require the approval of a user that belongs to the "level 1 approvers role"
- Overpayment Processes > $100 requires the approval of a user that belongs to the "level 2 approvers role"

The overpayment process type includes a setting to indicate if the overpayment process requires **Single** approval or **Multiple** levels of approval. If there is an overpayment process for $200 and the overpayment process type is configured for **Single** approval, only one approval is needed by the "level 2 approvers role". If instead the overpayment process type for this record indicates **Multiple** levels of approval: first a user that belongs to the "level 1 approvers role" must approve the overpayment process; afterwards, the overpayment process must be approved by a user that belongs to the "level 2 approvers role".

NOTE: Separation of Duties. The base product logic includes validation that prevents a user from approving an overpayment process if that user was the one that created the record or if that user already approved a previous approval "level" (for approvals that define **Multiple** levels).

NOTE: Different Approval Logic. If an implementation has approval logic that is not based on threshold amount but on some other condition, the base Determine Approval algorithm on the Standard Overpayment Process business object would need to be inactivated and a different Determine Approval algorithm is required. If no configuration is required for the custom approval logic, configuring the overpayment process type using the base Standard Overpayment Process Type business object may still be possible. Simply leave the threshold details blank.

To Do Entries Are Created To Notify Approvers

When the overpayment process detects an approval is required, it notifies the first approver by creating a To Do entry. The To Do entry is created using the To Do type and To Do role defined on the overpayment process type. All users who belong to the approving To Do role can see the entry. When a user drills down on an overpayment approval To Do entry, the *overpayment process* portal is opened. This portal contains summary information about the overpayment process. This portal is also where the user approves or rejects the overpayment process.

When the user approves the overpayment process, the To Do entry is **Completed** and the overpayment process's log is updated. If there are no higher levels of approval required, the overpayment process will transition to the **Approved** state. If there are higher levels of approval required, a new To Do entry is created to the next To Do role in the approval hierarchy.

To Do entries can create email. A To Do entry can be configured to create an email message for every user in the To Do role to inform the user(s) of new overpayment processes requiring their attention. Refer to *To Do Entries May Be Routed Out Of The System* for the details.

Monitoring and Escalating Overpayment Approvals

The base-package is supplied with an algorithm that your implementation can use to monitor overpayment approval requests that have been waiting too long for approval. This algorithm can complete the current To Do entry and create a new one for a different role when the time-out threshold defined on the algorithm's parameters is exceeded. If you've configured the system to send email for approval, this algorithm can also send x reminder emails (where x is defined on the algorithm's parameters) before the approval request is escalated to the new To Do role. Refer to *C1-OP-CHKOTO* for more information about this algorithm. If you plan to enable this functionality, plug in your configured algorithm on the **Approval In Progress** state on the **C1-OvrpyProcStandard** business object.

Rejecting Transitions the Overpayment Process to a Final State

When an overpayment process is being approved, anyone with the right level of access can reject it. When an overpayment process is rejected, the following takes place:

- The user is prompted for a reject reason.
- The overpayment process's log is updated with the reject reason and the overpayment process is moved to the **Rejected** state.

Refer to *Common Overpayment Procedures* for more information.

Putting an Overpayment Process On Hold

An overpayment process may be put on hold for one or more reasons. The Standard Overpayment Process base business object supports putting an overpayment process on hold due to a Suppression of overpayments for the taxpayer. When the overpayment process is first created and when it transitions from any state where it may have been waiting for a time, an algorithm checks for suppression and if one is found, the overpayment transitions to On Hold, with “suppression” as the reason code. The base BO also supplies a monitor algorithm to check for the expiration of the suppression and will transition it out of On Hold accordingly. The algorithm will do this if the reason for being On Hold is “suppression”.

If an implementation has another condition that should cause the overpayment to be put on hold, the following should be done:

- Design an algorithm that detects the condition and plug that algorithm into the appropriate place in the lifecycle. If the condition is detected, transition the record to On Hold with an appropriate On Hold reason.
- Design a monitor algorithm for the On Hold state that checks if the condition still applies (based on the reason code). When the condition no longer applies, the algorithm should indicate that it should transition out of On Hold.

Retry Overpayment Process

The Standard Overpayment Process base business object supports a “hand shaking” process with the base Direct Deposit bank event business object to support retrying a rejected refund attempt. the following points highlight the supported process:

- If the overpayment process has a direct deposit refund method and is configured to create a bank event, the Refund Approved state will create a bank event. The bank event will refer to the overpayment process in its log as the “created by” process and the overpayment process log will indicate that it created the bank event.
- If the bank event is rejected, an algorithm will create a new overpayment to try to issue the refund again. Some configuration allows an implementation to control some of the business rules:
  - The overpayment process to create is configured on the bank event type. An implementation may choose to configure a simple overpayment process type that doesn’t include any rules for calculating interest or approval or performing offsets, etc.
  - The refund method to use is configured on the bank event type. Some implementations may revert to mailing a paper check if a bank direct deposit failed. Other implementations may want to try to issue a direct deposit to a different bank. Note that the product does not provide any mechanism for determining a different bank. If business rules require contacting the taxpayer or making other attempts to find a different bank, custom algorithms are required. If the retry refund method is also direct deposit, the “retry” overpayment process creates another bank event. Note that it’s possible that this bank event will also be rejected causing yet another “retry” overpayment process.

- The maximum number of retries may be configured on the bank event type. This is mainly applicable for implementations that want to continue to try to issue bank deposits. The bank event rejection algorithm checks the maximum number before creating an overpayment. Each “retry” overpayment process indicates its “retry attempt number” which is used by the bank event algorithm to compare against the maximum.

- The “retry” overpayment process may not require an individual approval. However, it is still expected to be included in a Refund Control before issuing the refund.
- A “retry” overpayment process is stamped with the ID of the overpayment process that created the bank event. Note that if the original overpayment process is canceled, it will attempt to cancel any related “retry” overpayment processes. For example, if a tax form causes an overpayment to be created and then the overpayment creates a bank event that is extracted to the bank. If the bank event is rejected, a new overpayment process is created. If the tax form is reversed at this point, the overpayment process it created will be canceled. That in turn will attempt to cancel the Bank Event (which is not eligible for cancellation) and the related retry overpayment process.

FASTPATH: Refer to *Direct Deposit Bank Event Lifecycle* for more information.

Canceling an Overpayment Process

An overpayment process may be canceled from any non-interim state, including Completed. It is distinguished from Rejected which is expected to be used for a user’s decision to stop the overpayment process. Cancellation is meant for system cancellation, especially when initiated by a different source, such as the initiating process. For example, if a posted tax form creates an overpayment process, a reversal of the tax form should cancel the overpayment process.

NOTE: The business object does not have any restriction on allowing a user to cancel an overpayment process. The implementation should use appropriate security to limit which users are allowed to cancel an overpayment process manually, if any.

When an overpayment process is canceled, algorithms plugged in to the canceled state should attempt to undo actions that may have been triggered by the overpayment process. For example:

- If the amount was written off, the write off is canceled.
- If interest was calculated, it is removed.
- If a bank event is created, it is canceled. Note that if the bank event is in a state that doesn’t support canceled (such as Accepted by the bank), then the overpayment will still be canceled, and a log will be written to indicate that the bank event could not be canceled.

Enabling the System to Use Standard Overpayment Process

To enable this functionality the following configuration tasks are needed:

- Determine if your implementation wants to define an overpayment source to indicate what caused the overpayment process to be created. This is an optional field and is defined using the extendable lookup **Overpayment Process Source**. The values are meant to record the reasons that the obligation is in credit, for example “overpayment” or “benefit payment” or “liability adjustment”. Implementations that wish to use this field must provide appropriate logic to set the value, for example, using a pre-processing algorithm on the business object.
- Various adjustments are used for calculating interest, write off and offset for the overpayment process, if applicable based on business rules. For each of these actions, a suitable *adjustment type* is required.
    - If you wish to specify a minimum amount threshold for the overpayment process, you will need to define a non-calculated adjustment type for the minimum amount write-off.
    - If your implementation rules specify that interest should be calculated and the interest is calculated using a *rate*, you will need to define the following:

        - A calculated adjustment type for offset-able interest, or a calculated adjustment type for non offset-able interest.
        - An associated rate schedule that includes a *RQ rule* for calculating number of days
    - If your implementation rules specify that interest should be calculated and the interest is calculated using a *rate factor*, you will need to define the following:
        - A non-calculated adjustment type for offset-able interest, or a non-calculated adjustment type for non offset-able interest.
        - An associated rate factor.
    - If your implementation allows offset, then a non-calculated adjustment type needs to be defined for the offset.
    - To allow refunds via a check, an *A/P adjustment type* needs to be defined.
- Make sure the adjustment types are defined on an associated *adjustment profile* that is linked to any obligation types that are going to be covered by the overpayment process.
- To allow refunds via direct deposit define a bank event type to use to create the bank event that stages the direct deposit.

    NOTE: Implementations may also use 'negative payments' to implement a direct deposit if desired. To use this mechanism, a *distribution rule* needs to be defined. The distribution rule should be designed to "pay" a specific obligation.

- If approvals are required as part of the overpayment process, a To Do type is needed. The base product is supplied with a To Do type called *C1-OVAPP* that should be used as the basis for approval To Do type.

    In addition, your implementation should determine if the overpayment process should alert a user if it is in this state for too long. If so, an appropriate monitor algorithm should be configured. Refer to the detailed description of the Approval in Progress status in the metadata for this base business object for more information.
- If you wish to have a To Do created when issues are detected during the validation of the overpayment process, you will need a To Do type. Note that the base-package is supplied with a To Do type called *C1-OVISS* that should be used as a basis.
- For each To Do type that you wish to use, you will need appropriate To Do roles.
- The system has been currently configured to calculate non offset-able interest using a rate factor. If you wish to change this you will need to inactivate the base-package algorithm by adding an appropriate inactivation option to the business object. You will then need to plug in a different algorithm based on the logic you wish to implement.
- Review the overpayment and refund validation rules required for your overpayment processes to determine if existing algorithms are applicable or if additional algorithms are needed.

    NOTE: Base plug-ins. Click *here* to see the algorithms types available for the Overpayment Validation system event. Click *here* to see the algorithms types available for the Refund Validation system event.

- Define an overpayment process type for the business object **C1-OvrpyProcTypeAutoCredRef**.

NOTE: **Retry overpayment process types**. If your implementation supports the ability to retry a refund attempt when a bank event is rejected, a "retry" overpayment process is created. The configuration of the "retry" overpayment process type will probably be simpler than the original overpayment process type due to the fact that at this point steps like offset and approval are no longer applicable.

Setting Up Overpayment Process Types

An Overpayment Process Type defines the configuration information that is common to overpayment processes of a given type. The type of information captured on the overpayment process type is governed by the overpayment process type's business object.

To set up an Overpayment Process Type, select **Admin Menu** > **Overpayment Process Type**.

The topics in this section describe the base-package zones that appear on the Overpayment Process Type portal.

Overpayment Process Type - Main

An Overpayment Process Type defines the configuration information that is common to overpayment processes of a given type. The type of information captured on the overpayment process type is governed by the overpayment process type's business object.

To set up an Overpayment Process Type, select **Admin Menu** > **Overpayment Process Type**.

The topics in this section describe the base-package zones that appear on the Overpayment Process Type portal.

Overpayment Process Type List

The following functions are available:

- Click a *broadcast* button to open other zones that contain more information about the adjacent overpayment process type.
- The standard actions of **Edit, Duplicate** and **Delete** are available for each overpayment process type.

Click the **Add** link in the zone's title bar to add a new overpayment process type.

Overpayment Process Type

The Overpayment Process Type zone contains display-only information about an Overpayment Process Type. This zone appears when an Overpayment Process Type has been broadcast from the Overpayment Process Type List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_OP_PROC_TYPE*.

Tax Types

The Tax Type zone displays the list of tax types for which this overpayment process type is valid. Click the Add/Edit button to maintain this list.

Overpayment Process Type - Log

Navigate to the Log tab to view the *logs* for an overpayment process type.

The Big Picture of Refund Control

Implementations typically need to review and approve a group of overpayments that are slated for refund to ensure that there are enough funds in the bank to cover the payments out to the taxpayers.

The product provides an object called Refund Control that allows for a set of overpayments to be grouped for a mass approval. The following topics provide more information about refund control records.

How Are Refund Controls Created?

The product provides two ways to create a refund control and link overpayments to the refund control

- A user may manually create a refund control and define criteria such as tax type, overpayment process type and maximum amount. (Note that the exact criteria is determined by the business object). The refund control is then "validated" where an algorithm uses the criteria to select overpayment processes and then becomes eligible for approval.
- A refund control and its overpayments are created via a web service. This mechanism is used if an implementation chooses to do some analysis in a separate analysis tool and selects a list of overpayments to refund in the external tool. The expectation in this case is that analysis tool produces a file and that an appropriate mechanism like Oracle Service Bus transforms the file and interfaces the information via a web service call.

The refund control type includes configuration to indicate whether criteria is provided or whether the explicit list of IDs are provided. Note that the product does not provide any user interface mechanism for defining overpayment IDs manually. The IDs Provided option is meant for the case where a web service call creates the refund control and its list of IDs.

Regardless of the mechanism for creating refund controls, the record is created in the **Pending** status. The base product attaches a deferred monitor to the pending state so that the validation / selection of the IDs occurs in batch allowing for large volumes. Depending on an implementation's business practice, this deferred monitor may be one to run often during the day.

Once the record is validated, it is routed for approval. Refer to *Refund Control Approval* for more information.

Overpayment Configuration for Refund Control

The refund control functionality requires that the overpayment is configured to use the base business object **C1-OvrpyProcStandard** (Standard Overpayment Process) whose lifecycle includes a **Refund Control Approval in Progress** state. If implementations choose to create a different overpayment process business object and wish to use the refund control, the lifecycle must follow the base BO's lifecycle.

Refund Control Approval

A refund control typically requires one or more levels of approval before any financial activity can take place. The refund control type defines an approval hierarchy, indicating which group of users is authorized to approve the refund control based on its total amount (sum of all the overpayment amounts). The refund control type may be configured to require multiple levels of approval based on the total amount. Or it may indicate that a single group should approve where the group to approve can depend on the total amount. This is analogous to the approval functionality provided for an overpayment process. Refer to *Approval Is Controlled By Its Type* for more information.

If there is some amount that doesn't require approval, the refund control transitions directly to **Approved**. If one or more levels of approval are required, the record transitions to **Approval in Progress** and remains in this state until all approval occur. At any time, a user can choose to **Reject** the refund control at which point they may be prompted for a reject reason, based on the configuration. Once all approvals are done, the record transitions to **Approved**.

NOTE: Separation of Duties. The system includes validation to ensure that the user approving the refund control is not the same user that created the refund control and is not the same user that performed any previous approval.

To Do Entries are created to notify approvers. This analogous to the functionality provided for overpayments. Refer to *To Do Entries Are Created to Notify Approvers* for more information.

NOTE: Refund Control with No Overpayments. It is possible that after validating a refund control, it is found that no **Active** overpayments are linked to the refund control. In this case the record transitions to **Approval in Progress** so that a user can review the record and determine the cause. At that point the user can **Reject** the refund control.

Refunding Overpayments

Once a refund control is approved, its overpayments may be refunded. The approval step stamps the refund control with the batch job that is responsible for progressing the overpayments to issue the refunds. The batch job is taken from the refund control type. The current run number for that batch job is stamped as well. The base product provides a batch job (C1-RCPOR - Process Refund Control Overpayments) that is responsible for finding refund controls for the current run number and progressing all the overpayments with an **Active** link status to its next state.

NOTE: If the batch job determines that any of the overpayments are no longer in the appropriate state, the link status is updated to **Skipped** as an audit.

Configuring Refund Control Options

If your implementation supports refund controls, the following topics highlight configuration requirements.

In addition, your implementation should read the detailed descriptions for the base product business objects provided for Refund Control Type (**C1-RefundControlType**) and Refund Control (**C1-RefundControl**).

Setting Up Refund Control Types

A Refund Control Type defines the configuration information that is common to refund controls of a given type. The type of information captured on the refund control type is governed by the refund control type's business object.

To set up a Refund Control Type, select **Admin Menu** > **Refund Control Type**.

The topics in this section describe the base-package zones that appear on the Refund Control Type portal.

Refund Control Type List

The following functions are available:

- Click a *broadcast* button to open other zones that contain more information about the adjacent refund control type.
- The standard actions of **Edit, Duplicate** and **Delete** are available for each refund control type.

Click the **Add** link in the zone's title bar to add a new refund control type.

Refund Control Type

The Refund Control Type zone contains display-only information about a refund control type. This zone appears when a refund control Type has been broadcast from the Refund Control Type List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *C1_REFUND_CTRL_TYPE*.

Refund Control Batch Jobs

The following batch jobs must be scheduled as per your implementation's business rules:

- **C1-RFCMD** (Refund Control Monitor (Deferred)). This batch job selects refund control records that are **Pending** and validates them. For refund controls where criteria is provided this step selects the overpayment processes to include. For refund controls where IDs are provided, this step validates that the overpayments in the list are in an appropriate state. Once this batch job completes, the processed refund controls are ready for approval. Implementations should schedule this based on the business practice of the users that are creating the refund controls and the users that are expected to approve the refund controls. The assumption is that the approval should be done in a timely manner. So one option is to schedule this job on a frequent schedule, such as every 15 minutes. That way the approvals are processed quickly throughout the day.

  NOTE: Email routing. If your implementation plans to use email routing for the "refund control requires approval" To Do entry, then the background process **F1-TDERR** (To Do External Routing) must be scheduled after the monitor batch job with the same or similar frequency.

- **C1-RCPOR** (Process Refund Control Overpayments). This batch job selects refund controls that are stamped with this batch code and the current run number and processes the "active" overpayments such that the refund is issued. This batch job is creating financial transactions and the assumption is that implementations will run this in their nightly batch run.

  NOTE: The creation of the refund financial transaction is not the final step in issuing the refund. For refunds created via an A/P adjustment, the extract of A/P records to the external accounts payable system must also run. For refunds created via direct deposit, the appropriate batch jobs related to interfacing to the banks must also after the process to refund the overpayments is run.

Refer to *Batch Process Dependencies* for more information.

Implementing Web Service Options

If your implementation plans to use an external system to identify an explicit list of overpayments to include in a refund control, the product provides the following configuration to support it.

The XAI Inbound Service **C1-RefundControlUpload** provides the API for the service script **C1-RfCtlUpld**. This service script includes logic creating a refund control and linking a group of overpayments. Note that the script has been designed to be called iteratively for cases where a large file is received and the interface tool that processes the file can process it in chunks.

FASTPATH: Refer to the description of the XAI inbound service and the above service script for more information.

Individual Taxpayer Overpayment Process

The product provides a business object **C1-OvrpyProcAutoCredRef**, that is no longer recommended for use. It does not support refund control or suppression. It was designed to work with a legacy tax form BO that is no longer supported. It remains supported for upgrade purposes.

Note that its lifecycle differs from the Standard Overpayment Process. In addition to the lack of refund control and suppression support,

- It uses a canceled state for situations where a change in the obligation balance is detected.
- It has a special Minimum Write Off state.
- It has logic to support carry forward and contribution amounts but only if populated by a creation algorithm. A user was not able to populate the information manually.

Chapter 13

Defining Bank Event Options

Bank events capture specific transaction information that is interfaced to financial institutions.

The following sections describe how to set up the tables that control this functionality.

The Big Picture of Bank Events

Bank events are used to interface transactions to financial institutions.

Current base functionality supports the use of bank events for processing direct deposits (refunds).

The following sections provide more information.

Direct Deposits Using Bank Events

The Bank Event object can be used for processing direct deposits or refunds.

Base-package overpayment processing functionality supports the ability to issue refunds using bank events. An Overpayment Process Type can be configured to create bank events, by specifying a Bank Event Type. Refund creation logic can then check that configuration to determine whether or not to create a bank event.

The base-supplied business object **C1-OvrpyProcStandard** includes refund processing using bank events, when applicable. Refer to this business object's description and the online help for Overpayment Processing for more details.

FASTPATH: Refer to *The Big Picture of Overpayments* for more details on overpayment processing.

A direct deposit bank event typically triggers the financial effect of the overpayment/refund on the related obligation's balance. An Adjustment can be created when the direct deposit is extracted or accepted, depending on the business rules.

Determining Bank Details

Each direct deposit bank event specifies the details of the bank account, to which funds will be sent. Current base logic assumes that the bank details are determined from the related overpayment process.

The **Bank Details Determination** algorithm on the Installation Record is used to retrieve the currently effective bank details that are used for the direct deposit. When called in ‘Read’ mode, this algorithm returns bank details such as Bank Account Number, Bank Routing Number, Account Name, Bank Account Type, etc.

The algorithm that is supplied with the base package assumes that the bank details are stored in *Account — Auto Pay*. For more information, go to *Installation Options - Algorithms* and scroll to the **Bank Details Determination** system event.

Direct Deposit Bank Event Lifecycle

The base product provides the **C1–DirectDepositBankEvent** business object, which supports the following lifecycle steps:

- A direct deposit bank event is created in a **Ready to Extract** state. When an overpayment process is at the point in its lifecycle where it’s ready to issue the refund, it can create a bank event. When it does, it also populates the extract batch control (from the bank event’s type) and the next run number on the bank event, so that it gets picked up the next time the bank event extract batch process runs.
- An extract batch process runs and picks up all bank events that are stamped with the extract’s batch control code and the current run number. For each bank event, the **Bank Event Extract** algorithm (defined on the bank event’s type) is executed, to map the bank event information into a corresponding transaction record in the extract file. When the bank event is processed successfully, the Extract Date/Time and Monitor Control Date are both set to the process date. The Monitor Control Date controls when the bank event transitions to the next state — in this case, to **Extracted**. Setting this date to the current processing date would cause the Bank Event Deferred Monitor (with Date) to pick up the bank event in the next run and perform the state transition accordingly

  NOTE: Refer to *Downloading Direct Deposits* for more details.

- An **Extracted** direct deposit bank event stays in that state until the financial institution accepts or rejects the transaction. The bank event can be monitored to do any necessary timeout processing or automatic transitions when a response is not received within a specified period of time. In addition, an adjustment (to affect the obligation’s balance) can be created at this point, if business rules dictate that the balance is impacted upon extraction/download.

  NOTE: The base product provides a number of algorithms that are not plugged in on the base business object by default. Implementations will need to configure these algorithms based on their business rules. Refer to the detailed description of the **C1–DirectDepositBankEvent** business object for details on algorithms that can be plugged in on the **Extracted** state.

- When the financial institution accepts the direct deposit, the bank event can transition to **Accepted**. The assumption is that this transition is done from back end services because it involves processing responses from financial institutions. An adjustment (to affect the obligation’s balance) can also be created at this point, if business rules dictate that the balance is impacted only after the direct deposit is accepted.

  NOTE: Since the base product does not provide logic for receiving bank responses, logic for transitioning to **Accepted** is assumed to be implementation responsibility.

  NOTE: Refer to the detailed description of the **C1–DirectDepositBankEvent** business object for details on the **Create Adjustment** algorithm that can be plugged in on the Accepted state.

- When the financial institution rejects the direct deposit, the bank event can transition to **Rejected**. The assumption is that this transition is done from back end services because it involves processing responses from financial institutions.

  NOTE: Since the base product does not provide logic for receiving bank responses, logic for transitioning to **Rejected** is assumed to be implementation responsibility.

- **Rejected** bank events are usually caused by invalid bank details. The base package supports the ability to reprocess a rejected direct deposit. An algorithm can be plugged in on this state, to create a new Overpayment Process. This is to allow for retry processing, which includes attempting to look for a new set of bank details. Another algorithm can also be plugged in, to reset existing bank details so that it they are not used in future direct deposits. A base algorithm is plugged in on this state, for canceling any existing adjustment, i.e. created upon extraction.

  NOTE: Refer to the detailed description of the **C1–DirectDepositBankEvent** business object for details on algorithms that can be plugged in on the **Rejected** state.

- In some exceptional cases, there may be a need to reject a previously accepted direct deposit. This is for cases where the financial institution encounters specific problems after accepting the transaction. The lifecycle allows for this transition.
- **Ready to Extract** and **Extracted** direct deposit bank events can be **Canceled**. If canceling from **Extracted** state, any existing adjustment is also canceled. The lifecycle supports system cancellation from a related process. The Standard Overpayment Process business object supports the ability to cancel a related bank event when the overpayment process is canceled. The **Allow System Cancel (C1AC)** status category value is used to determine which states allow for system cancellation. The **Ready to Extract** state is configured with this value. Implementations can add this value to other states (e.g. **Extracted**) as needed. Note that logic to alert a user about cancellation is not provided by base.

Downloading Direct Deposits

Direct deposit bank events are extracted/downloaded using a specific background process that picks up bank events that are ready for extraction and formats the direct deposit information into an output file.

This background process invokes the **Bank Event Extract** plug-in on the bank event type, to map and format each bank event's information into the output direct deposit record.

The base product provides a sample background process that downloads SEPA (Single Euro Payments Area) Credit Transfer transactions. A corresponding **Bank Event Extract** algorithm is also supplied. The following sections describe this process.

C1-SCTIE - Download SEPA Credit Transfer Instructions

This process reads all bank events marked with the **C1-SCTIE** batch control code and a given run number and creates the flat file that is passed to the SEPA clearing and settlements mechanism. Refer to *SEPA Credit Transfer Record Layouts* for details of the record layouts.

NOTE: **You can rerun this process.** You can reproduce the flat file at any time. Simply request this job and specify the run number associated with the historic run.

For each bank event, this process invokes the **Bank Event Extract** algorithm that is defined on the associated bank event type, to map and format the bank event information into the output credit transfer record. Click *here* to see the algorithm types available for this system event. The base product provides the *C1–SEPA-FCT* algorithm, which formats the credit transfer information using the ISO 20022 standard XML schema specifications.

If you require a different flat file format or if your mapping rules differ, you must create a customized extract program and/or a customized **Bank Event Extract** algorithm. The base programs may be used as a starting point.

Refer to *Batch Process Dependencies* for more information on when this process is run.

SEPA Credit Transfer

The SEPA (Single Euro Payments Areas) Credit Transfer (SCT) scheme allows an originator to initiate credit transfers to the beneficiary.

The direct participants are the following:

- Originator
- Originator's bank
- Beneficiary
- Beneficiary's bank

The originator and beneficiary must each hold an account with a payment service provider located within SEPA. The scope is limited to payments in Euro.

The indirect participants are the following:

- Clearing and Settlement Mechanisms (CSMs) such as an automated clearinghouse
- Intermediary Banks that offer intermediary services to originator and/or beneficiary banks

SEPA Credit Transfer Record Layouts

The topics in this section describe the layout of the records created by *C1-SCTIE - Download SEPA Credit Transfer Instructions*.

NOTE: The following record layouts are based on the CustomerCreditTransferInitiation message (pain.001.001.03) specifications, as described in the Payments Maintenance Message Definition Report in the ISO 20022 standards.

Group Header Record

The SEPA credit transfer flat file must have one record of this type.

Since this record includes the number of transactions processed, this record is written at the end of the file after all transaction records have been processed.

NOTE:
The base product provides a data area **C1-SEPAGroupHeader** that contains the group header record layout. All fields are alphanumeric.

The following table lists the fields that the **C1-SCTIE** extract is mapping. To see the rest of the fields, refer to the data area's schema definition.

Field Name	*Description*	*Source / Value*
MsgId	Message Identification	SEPA Master Configuration: generalMasterConfiguration/ messageIdentification
CreDtTm	Creation Date/Time	System Date/Time
NbOfTxs	Number of Transactions	Computed at the end of extract processing
InitgPty/Nm	Initiating Party -Name	SEPA Master Configuration: Derived from the Initiating Party (Person).

Payment Information Record

The SEPA credit transfer flat file must have one record of this type for every obligation. It includes one or more credit transfer transactions.

NOTE:
The base product provides a data area **C1-SEPACreditTxferPaymentInfo** that contains the payment information record layout. All fields are alphanumeric.

The following table lists the fields that the **C1-SCTIE** extract is mapping. To see the rest of the fields, refer to the data area's schema definition.

Field Name	***Description***	***Source / Value***
PmtInfId	Payment Information Identification	From 'Payment Information Identification' characteristic value on Obligation, where the characteristic type is defined in the SEPA Master Configuration. Obligation ID is taken from the Bank Event.
PmtMtd	Payment Method	'TRF'
PmtTpInf/SvcLvl/Cd	Service Level (Code)	‘SEPA’
PmtTpInf/LclInstrm/Cd	Local Instrument (Code)	Blank.
ReqdExctnDt	Requested Collection Date	Payment Event.Effective Date
Dbtr/Nm	Debtor Name	SEPA Master Configuration: Derived from the Initiating Party (Person).
Dbtr/PstlAdr	Debtor Postal Address	SEPA Master Configuration: Derived from the Initiating Party (Person). If an address exists: StrtNm is mapped to Address1, PstCd is mapped to Postal (prefixed with State code, if applicable), TwnNm is mapped to City and Ctry is mapped to Country.
DbtrAcct/Id/IBAN	Debtor Account Identification	Bank Account.Account Number, where Bank Account is specified in the SEPA Master Configuration.
DbtrAgt/FinInstnId/BIC	Debtor Agent BIC	Bank Account.DFI ID. where Bank Account is specified in the SEPA Master Configuration.
CdtTrfTxInf/PmtId/EndToEndId	Credit Transfer Transaction: Payment Identification - End To End Identification	Bank Event ID
CdtTrfTxInf/PmtTpInf/SvcLvl/Cd	Credit Transfer Transaction: Service Level	‘SEPA’
CdtTrfTxInf/Amt/InstdAmt	Credit Transfer Transaction: Instructed Amount	Bank Event.Amount
CdtTrfTxInf/CdtrAgt/FinInstnId/BIC	Credit Transfer Transaction: Creditor Agent's BIC	Bank Event.Bank Routing Number
CdtTrfTxInf/Cdtr/Nm	Credit Transfer Transaction: Creditor Name	Bank Event.Bank Account Name
CdtTrfTxInf/CdtrAcct/Id/IBAN	Credit Transfer Transaction: Creditor Account	Bank Event.Bank Account Number

Setting Up Bank Event Options

This section describes how to configure bank event types.

Setting Up Bank Event Types

Bank Event Types contain the rules that control how a specific type of bank event is processed.

To set up a Bank Event Type, select **Admin Menu** > **Bank Event Type**.

The topics in this section describe the base-package zones that appear on the Bank Event Type portal.

Bank Event Type List

The Bank Event Type *List zone* lists every bank event type. The following functions are available:

- Click a *broadcast* button to open other zones that contain more information about the adjacent bank event type.

- The standard actions of **Edit, Delete** and **Duplicate** are available for each bank event type.

Click the **Add** link in the zone's title bar to add a new bank event type.

Bank Event Type

The Bank Event Type zone contains display-only information about a Bank Event Type. This zone appears when a Bank Event Type has been broadcast from the Bank Event Type List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *C1_BANK_EVENT_TYPE*.

Chapter 14

Defining Overdue Processing and Collection Options

The system periodically monitors how much your taxpayers owe to ensure they haven't violated your collection rules. When a violation is detected, the system initiates the appropriate activities (e.g., letters, collection agency referrals, and eventually write off). The topics in this section describe how to configure the system to manage your overdue processing and collection requirements.

NOTE:
Collecting on unpaid debt. The overdue processing module has been designed to collect on virtually anything from an outstanding obligation to a specific financial transaction. You tell the system what you collect on by configuring the various overdue processing control tables.

CAUTION:
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 taxpayer, the age of debt, the type of tax, etc.), your setup process will be more challenging.

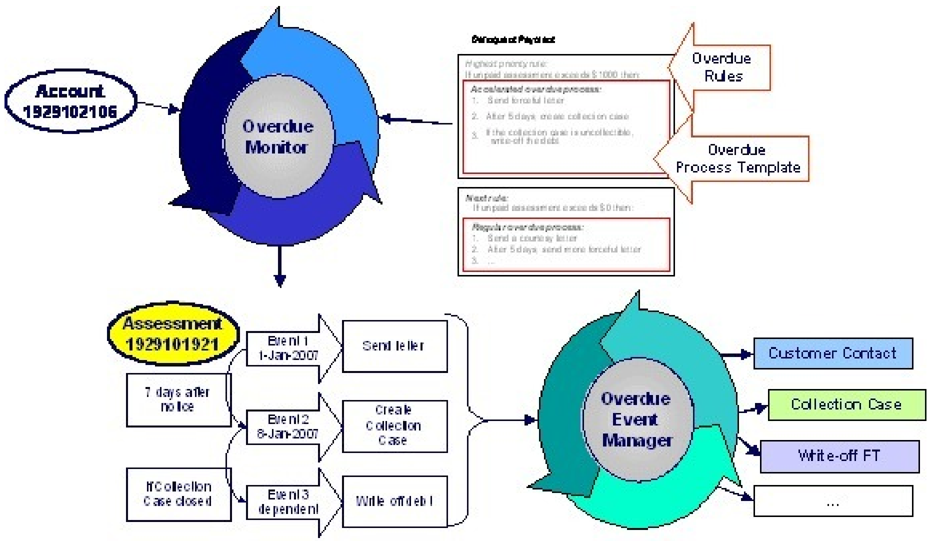
Note: The terms "customer" and "taxpayer" may be used interchangeably in this chapter.

Case Study - Collecting On Overdue Debt

The topic in this section introduces a case study that describes how overdue processing can be used to collect on overdue debt. 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.

Monitoring Overdue Debt

The following diagram illustrates the objects and processes involved with collecting overdue debt.



There are many important concepts illustrated above:

The Overdue Monitor checks if your accounts have debt that violates 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. Collection classes are monitored by the Overdue Monitor. This chapter describes the Overdue Monitor.
Overdue rules define when and how unpaid debt is 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. When you set up a monitoring algorithm, you define the type of overdue process that should be created when overdue debt is detected. You do this by defining the appropriate "overdue process template".
An overdue process template defines how to handle overdue debt	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 taxpayer 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 product.
Multiple objects can be associated with a single process	The above diagram shows a single assessment linked to an overdue process. It should be noted that an overdue process is capable of referencing multiple assessments (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 FTs and obligations under a single overdue process. The type of object managed by an overdue process is defined on its *overdue process template*.

If a taxpayer pays the debt, the overdue process is cancelled	If the overdue debt 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 Event Manager activates and triggers overdue events	The *Overdue 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 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 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.

How Does The Overdue Monitor Work?

This section describes how the Overdue Monitor background process uses your overdue rules to collect overdue debt.

Recommendation. We suggest that you familiarize yourself with the concepts described in the *case study* before reading this section.

Different Overdue Rules For Different Taxpayers

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 may have different collection rules for different jurisdictions (i.e., divisions).
- You may have different collection rules for different taxpayer segments. You differentiate your taxpayers in respect of the overdue via the **collection class on the taxpayers' accounts**. An account's initial collection class is defaulted from its account type. You may override an account's collection class at will.
- You may have different criteria for every currency in which you work because the monitoring process always compares a taxpayer's debt against some value and this value must be denominated in the taxpayer's currency. A taxpayer's currency is defined using a **currency code on the account**.

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, 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*

> **NOTE:**
> **Additional rules.** Your implementation can have an **Overdue Monitor Rule** that caters for credit balances on obligations. For an example of an algorithm that creates an overpayment process for an obligation in credit, please refer to the base algorithm type *C1-CC-INITOP*.

When Is An Account Monitored?

The Overdue Monitor determines if an account violates your overdue rules. It can be run in one of two modes.

If the Overdue Monitor process is run with the ‘Restrict by Periodic Review’ parameter set to true (batch control **C1-ADMVP**), it will review all accounts that have not been reviewed for at least X days, where X is defined on the *Account Type - Controls* associated with the account's account type and division (in the field Min Compliance Review Freq (Days)).

If the Overdue Monitor process is run with the ‘Restrict by Periodic Review’ parameter left blank (batch control **C1-ADMOV**), it examines account debt 'on demand' by looking at a special table called Compliance Review Schedule, which contains one or more specific account IDs to be selected and the date on which each account should be reviewed. Processes can insert a record into this table if something happens to the account. Some examples of base processes that insert a row in this table include:

- FT freeze. A row is inserted for the account for the current date if the obligation's balance is considered overdue or for the payment due date plus the Compliance Review Grace Days specified on account type if the obligation's balance is not yet due.
- If a payment is canceled with a cancellation reason that indicates non-sufficient funds.

The role of the ‘on demand’ Overdue Monitor is to process the accounts for which some activity is known to have occurred as soon as possible after the activity that triggered the review. The recommendation is to run the monitor using batch control **C1-ADMOV** on a nightly basis.

The role of the ‘periodic’ Overdue Monitor is to ensure that all accounts are reviewed regularly, regardless of whether there has been specific activity. The times at which the monitor should be run in this mode (using batch control **C1-ADMVP**) will depend on your organization’s business practice and the nature of your account types.

> **NOTE: 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 Compliance Review Schedule table (CI_ADM_RVW_SCH) when the event occurs, populating the account ID and an appropriate review date. Note that the Compliance Review Grace Days on the account type is available for use by any process when determining an appropriate 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 the following options:

- The accounts are monitored by the Overdue Monitor (this is described in this chapter).
- The accounts are not monitored for overdue debt.

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.

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 topics in this section describe the major components of an overdue process.

Overdue Objects

When an overdue process is created, the system links the overdue object(s) to the process. For example, if an overdue assessment FT is detected, the assessment FT 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 assessment FTs, you define a foreign key characteristic type that references the FT object.

Overdue Events

An overdue process has one or more overdue events. These events are the actions designed to encourage the taxpayer 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 compliance rating
- Create a collection case to manage actions such as creating pay plans and referring debt to collection agencies
- … (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 Event Manager*, is responsible for activating, monitoring, and triggering overdue events. Activation of an event causes the system to do whatever the event indicates; for instance, send a letter, send a To Do entry to a user or write-off debt.

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-product 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.

Experimenting With Alternative Overdue Process Templates

The system allows you to determine the efficacy of proposed overdue process templates using a small subset of taxpayers before implementing the templates on the entire taxpayer 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 individual taxpayers starts with a "gentle reminder" letter followed 10 days later by a letter threatening to place a lien to secure the debt 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.

After setting up the above, the *Overdue Rule Plug-In* will use the challenger template X% of the time rather than the champion template.

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-product 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-product algorithm (*C1-OBASM-AMT*) if you collect on overdue assessments or obligations. The base algorithm uses the **Obligation Type - Determine Detailed Balance** algorithm to calculate the amounts.

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 rules. If you plug in the base product Account Type - FT Freeze algorithm (*C1-CFTZ-CMRV*), an account's overdue processes will be reviewed for cancellation whenever a credit FT is frozen for the account.

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.

NOTE: **Why two algorithms?** The reason two algorithms are involved in cancellation is that we want the cancellation logic to be encapsulated in one place so it can be called during both manual and automated cancellation.

NOTE: **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.

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*.

NOTE: Bottom line. Most overdue events are created by the system when it creates an overdue process for delinquent debt. 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*.

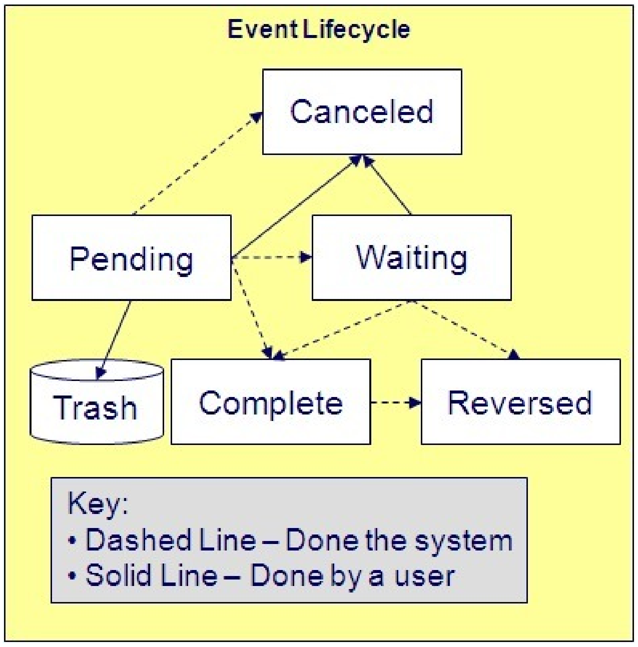
Overdue Event Information Is Overridable

- "Overdue event info" is the concatenated string of information that summarizes an overdue event throughout the system. The base-product 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 topics in this section describe an event's lifecycle.

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 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 Event Manager real-time, they can click a button on *Overdue Process - Main* to activate (and recursively trigger) events online / real-time.

Some implementations may wish to activate events using Overdue Control rather than the Overdue Event Manager, which allows a user to target overdue event types and count limits for activation. Refer to *The Big Picture of Overdue Control* for more information.

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 Event Manager sets the trigger date on such an event when it detects that all of its dependent events are complete / canceled.

NOTE:
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 another process, such as an appeal by the taxpayer, is outstanding. Then, when the other process is complete, the overdue process can start up where it left off. The base provides an algorithm type (*C1-OVRD_SUPP*) to suspend an overdue process while the account or person is linked to a *The Big Picture of Suppressions* configured to suppress overdue processes.

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 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 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).

NOTE:
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 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 product does not contain the algorithms you need, simply develop new ones using the base-product 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 product cancel algorithms insert a *log entry* when a user manually cancels an event.

Reversing Events

If for any reason, the action taken by event activation needs to be reversed, the **Event Reversal** algorithm may be executed to perform logic that reverses the effect of the activation.

Note that some event types may not be reversible. For example, P&I charges don't need to be reversed. The P&I calculation logic corrects itself, so it simply needs to be recalculated. Also, an event that creates a collection case doesn't support reversal given that a user is involved with a collection case. It is not clear what can / should be reversed.

FASTPATH: Refer to *overdue correction* link for more information.

Overdue Correction

Some implementations require the ability to correct an overdue process that is in progress. For example, a revenue authority may detect that a large group of accounts have been impacted by some incorrect information and steps are needed to attempt to reverse and restart collections for those account. This is an unusual circumstance that shouldn't happen very often.

NOTE: The product does not support this functionality for a single overdue process. It is only supported en masse using an entity correction record to manage the list of overdue processes effected and the action to take. Refer to *The Big Picture of Entity Correction* for more information.

The overdue correction business object provided (**C1-OverdueCorrectionControl**) supports the following options:

- Cancel Only. This option means that the identified overdue processes should simply discontinue processing. However, any actions already taken are considered done. The overdue correction process will cancel all **Pending** and **Waiting** events for the linked overdue processes. In addition the overdue processes themselves will be canceled.
- Reverse. This option means that in addition to discontinue future events for the identified overdue processes, the correction should also attempt to reverse actions taken by already activated events. This option requires an overdue event type to be provided. The overdue correction process will run the *event reversal* algorithm for any completed or waiting events for the linked overdue processes from the most recent event back to the input event. Any pending events are canceled along with the overdue processes themselves.
- Reverse and Restart. This option means the correction should attempt to reverse actions taken by already activated events and then recreate the template's events from a given event type. This option requires an overdue process template and a restart date along with the overdue event type to reverse to. The overdue correction process will perform the reversal logic as described above. In addition, it will rebuild the events for the overdue processes from the input overdue event type using the information on the overdue process template to rebuild subsequent events. Event dependencies are built as per the template. The trigger date for any new pending event that does not have a dependency is set to the input restart date.

  NOTE: Not all events may allow for restart. For example, if a given overdue process has a collection case in progress, a request to reverse and restart this process may not make sense. The *overdue event type* includes an event restart flag to indicate if events of this type support restart. If an overdue process in an overdue correction list that is configured to reverse and restart has an event in the list with this flag set to **Not Applicable**, the overdue process will not be processed. Refer to the entity correction validation algorithm *C1-OD-CORVAL* for more information.

NOTE: Refer to the correct entity algorithm *C1-OD-COROP* for more information.

The Big Picture of Collection Cases

In many tax authorities collecting unpaid debt is typically done in two phases: a series of unmonitored actions, typically letters, attempting to convince the taxpayer to pay, and a series of user-oriented actions such as contacting the taxpayer, setting up payment plans, and referring debt to a collection agency. The user-oriented actions are handled using a collection case. While not required, it is expected that many overdue processes will create a collection case when the overdue events did not prompt payment of the debt.

The topics in this section provide background information about collection cases.

Collection Case Overview

A collection case provides the functionality to record and track the interactions between the taxpayer and the user responsible for the collection. Many actions can take place once a collection case is created:

- A pay plan can be created
- The debt may be referred to a collection agency
- Letters may be sent to the taxpayer to advise of additional penalties
- A lien may be placed to secure the debt

How Are Collection Cases Created?

The activation of an overdue event creates a collection case. It is possible for an overdue process to be linked to more than one collection case over time but only one collection case can be active for an overdue process at a given time. The base package provides a sample overdue event activation algorithm (*dataDictionary?type=algtype&name=C1-CR-COL-CS*) to create a standard collection case.

Collection cases are created for persons not accounts. It is possible for collection cases that are linked to different overdue processes to be consolidated into a single case.

It is important to note that a collection case exists with respect to one or more overdue processes. A collection case's overdue process defines the objects in arrears. These objects are monitored to determine if the overdue process can be cancelled and the collection case can be closed.

Collection Case Lifecycle

The lifecycle of the collection case depends upon the configuration of the associated business object. The base package includes a sample standard collection case with a simple lifecycle, as follows:

- The collection case is initially created in the **Pending Investigation** state.
- The case will transition to the **Actions In Progress** state the first time a user initiates an action, such as creating a pay plan or sending a letter. Since many collection case actions happen in parallel, the expectation is that the case remains in this state until closed. Monitoring algorithms can be configured to check whether any of the actions in progress for the case have been waiting too long.

- The case may be **Transferred** to another collection case type or to an existing collection case.
- The user can manually transition the collection case to the **Uncollectible** state if they determine that no further action is to collect the debt is possible.

In addition to the standard actions for edit and state transition, a collection case may have special actions that apply to this type of collection, such as creating a pay plan or a collection agency referral. Additional actions are configured via the maintenance UI map. Refer to the base business objects for further details of the sample configuration.

How Are Collection Cases Closed?

Collection cases can be closed at the user's discretion or when the associated overdue process is cancelled. The base package provides a sample **Cancel Logic** algorithm (*C1-CL-COLLCS*) that closes any open collection cases linked to the process provided there are no other active overdue processes for the case.

Overdue Events Wait For The Collection Case To Conclude

Overdue events that create collection cases are perfect examples of events that *wait* for the object they create to complete before they, in turn, **complete.** After the collection case concludes, the originating overdue event will complete thus triggering its dependent events, such as writing off the debt. The base package provides a sample **Monitor Waiting Event** algorithm that checks whether all collection cases associated with the overdue process are in a final state.

Collection Case Type Defines Parameters

For each type of collection case, you must configure a collection case type to capture the appropriate parameters needed by the collection case actions. Typical parameters include:

- The To Do Type used to notify the responsible user that a new collection case has been created
- The default pay plan type to be used for a collection case of this type
- The To Do Types to be used to notify the responsible user if certain actions have been outstanding for too long without a response, such as a collection agency referral with no updates.

The Big Picture Of Collection Agency Referrals

Before debt is written off, many implementations refer the unpaid debt to a collection agency. The topics in this section describe how collection agency referrals are managed.

Collection Agency Referrals Overview

The system creates a *Collection Referral* record for a collection agency to track the debt that is to be collected. A collection referral is linked to an account. Collection referrals have history records that contain the amount of debt referred to the agency. Creating a history record 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.

How Are Collection Agency Referrals Created?

Users can create collection agency referral manually or by configuring an overdue event type with an activation algorithm that creates the referral. The base package sample activation algorithm (*C1-OE-AGYREF*) will refer the total unpaid debt for the overdue process to the collection agency with the least amount of referred debt. If you prefer different logic, you must develop your own algorithm.

In many tax authorities, creating a collection agency referral is one of the actions that would typically take place for a collection case. The base product BO for collection case **C1-StandardCollectioncase** can be used as an example of how to include the functionality to create a collection agency referral and link it to the collection case.

Cancelling Collection Agency Referrals

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). You can cancel a referral manually by simply creating a new collection agency referral history record (with a type of **cancel**).

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. If your implementation chooses to create collection agency referrals via overdue events, we strongly recommend plugging in an algorithm that will cancel the referrals when an overdue process is cancelled. If you choose to manage collection agency referrals via collection cases, the base package provides a sample business object status **Enter** algorithm (*C1-CCC-AGCYR*) to cancel any referrals linked to the collection case when it is closed.

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. If your implementation chooses to create collection agency referrals via overdue events, you set can set up your overdue process template to have an event that creates a collection agency cancellation X days after the referral. The base package provides a sample event activation algorithm (*C1-OE-AGYCAN*) to cancel all active collection agency referrals associated with the overdue process

NOTE:
Log entry. The base-product overdue event activation algorithms that make and cancel collection agency referrals insert rows in the overdue process's *log* to audit these events.

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.

Starting Write-Off Oriented Events

While the system provides overdue event activation algorithms that manage write-off oriented actions, such as referral to collection agency, tax authorities typically handle those actions as part of collection case activity.

Most overdue process templates will be configured to contain an event that writes-off the unpaid balance of the debt. This event should be configured to be dependent on the event that created the associated collection case. A **Monitor Waiting Event**algorithm can detect the collection case is closed and complete the waiting event. This allows the subsequent write off event to be triggered.

Small Amount Write-Downs

Many organizations will write-down a debt whose value is small early in an overdue process. The base package overdue event activation algorithm to write off assessments (*C1-WRITE-OFF*) includes a parameter to support this requirement.

(For example, indicate that you should write down an overdue process's assessment 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.

Write Offs And Overdue Process Cancellation

If an overdue event writes off debt, the state of the process depends on your cancel criteria and where the overdue event is positioned in the overdue process. For example, if an overdue process has an overdue event that writes off small amounts of debt early in the process, a process whose debt meets the threshold criterion will be canceled when the event activates.

Contrast this to an overdue process where the last event writes off the debt. Because there are no other events to activate, the process will complete (i.e., it will not be canceled).

Calendar vs. Work Days

When you set up your overdue templates, you supply information that controls how the 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 2 nd event is triggered 2 days after the 1 st event is complete.
- 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 2 nd event 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 2 nd event is triggered 7 days after the 1 st event, the system will add 7 days to the 1 st 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 2 nd event is triggered 7 days after the 1 st event, the system will count 7 workdays and set the trigger date accordingly.

NOTE:
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.

The Big Picture Of Overdue Control

The Overdue Event Manager (**C1-ODET**) is configured to activate all events whose trigger date has arrived. However, there are some implementations that want to control volumes and types of overdue events that are activated. The product provides an object called Overdue Control that may be used by implementations that prefer to work this way. The topics in this section provide more information about overdue control records.

How Are Overdue Controls Created?

A user manually creates an overdue control and define criteria such as overdue event type, tax type, revenue period and maximum number of events. (Note that the exact criteria is determined by the business object).

NOTE: The base overdue control BO includes criteria for limiting overdue processes by tax type or revenue period. As such, the processing is limited to overdue processes that collect on obligations. The algorithms supplied with the base business object would not work with other collecting on objects, such as assessment.

The overdue control record is created in the **Active** status. The base product attaches a deferred monitor to the active state so that the next step, selection of the overdue processes that satisfy the criteria, occurs in batch allowing for large volumes. Depending on an implementation's business practice, this deferred monitor may be one to run often during the day.

NOTE: Note that waiting events are not included in the selection. The assumption is that waiting events are only waiting for user intervention and that a user will progress the event as appropriate.

The system captures the list of overdue processes in an internal trigger table for technical purposes to aid in performance. The list is not visible to the user. The assumption is that exactly which overdue processes get triggered is not important. The record is simply trying to control volumes.

Once the overdue processes are selected, the record transitions to **Processed**. Note that the overdue control does not need any approval step.

Processing Events

Once an overdue control transitions to processed, the overdue processes that it selected in the trigger table may be activated. The transition from **Active** to **Processed** stamps the overdue control with the batch job that is responsible for activating the overdue events. The batch job is taken from the overdue control type. The current run number for that batch job is stamped as well. The base product provides a batch job (C1-AOPFR - Activate Overdue Process from Triggers) that is responsible for finding overdue controls for the current run number and activating the appropriate events for the overdue processes listed in the trigger table.

The standard event activation logic is used when activating the overdue processes in the trigger table. As per standard activation logic one or more of the following may occur:

- The overdue process' cancel criteria algorithm is executed and if the criteria are satisfied, the overdue process is canceled.
- If applicable, the hold event activation algorithm is executed and if the hold conditions are met, the event activation is skipped.
- Once the initial event to activate is processed, any subsequent dependent events that are eligible to be activated will also be activated.

Creating Overdue Procedures

Your overdue procedures define how your organization collects overdue debt. In this section, we describe how to set up the data that controls these procedures.

CAUTION:
There are numerous ways to design your overdue 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 procedures, we encourage you to gain an intuitive understanding of these options by using the system to prototype the alternatives.

Set Up Tasks

The above topics provided background information about how overdue processing works. The following discussion summarizes the various set up tasks.

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-product 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.
- *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-product.
- *Messages* that are used to define the verbiage in the *log entries*. For example, if you use the base-product 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-product algorithms (this means you should not have to set up new messages).

  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

When setting up *collection classes*, 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-product algorithm.

Collections - Installation Options

For more information about collections installation options, refer to *Installation Options - Collections*.

Standard Collection Case Type

The base product supplies the business object **C1-StandardCollectionCase**, which is designed to cater for collection cases that incorporate standard actions such as creating pay plans and collection agency referrals.

The base product also supplies the business object **C1-StandardCollectionCaseType**, which defines the configuration information used by the standard collection case. To enable this functionality the following configuration tasks are needed:

- Define an appropriate To Do type and To Do role for sending a notification that a new collection case has been created for an overdue process.
- Define a Customer Contact class and type that reference the default letter template to be sent to a taxpayer for a collection case of this type.
- Define an appropriate To Do type and To Do role for sending a notification that the maximum days to wait after sending a letter before additional activity should occur has been exceeded.
- Define an appropriate To Do type and To Do role for sending a notification that the maximum days to wait after referring debt to a collection agency has been exceeded.
- Define a default Pay Plan obligation type for collection cases of this type
- Define appropriate reasons for closing a collection case due to the debt being uncollectible. Uncollectible reasons are defined using a customizable *lookup*. The lookup field name is **C1_COLL_CASE_UNCOLL_RSN_FLG**.
- Define a collection case type for the business object **C1-StandardCollectionCase**

Your implementation can define additional business objects to support collection case processing.

Feature Configuration

You must set up a *Feature Configuration* to define parameters that control various overdue processing options. Find the Feature Configuration record for the **Overdue Processing** feature type. (It may need to be defined if it does not exist).

The following points describe the various **Option Types** that must be defined:

- **Trigger Date Option** This option controls how the system computes the trigger dates on overdue 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.
- **Champion Challenger Option.** 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 rule algorithm 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 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*.

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 **Alert** algorithm (*C1-OD-PROC*). This algorithm is plugged in on the*Installation* record.

Alert To Highlight Active Collection Cases

If you want an alert to appear if the account has active collection cases, you must configure an appropriate **Alert** algorithm (*C1-CCAL-COCS*). This algorithm is plugged in on the*Installation* record.

Overdue Control Tasks

If your implementation chooses to use overdue control, the following tasks must be configured.

- You must define an appropriate *overdue control type*.
- The following batch jobs must be scheduled as per your implementation's business rules:
  - **C1-OCDM** (Overdue Control Monitor (Deferred)). This batch job selects overdue control records that are **Active** and transitions them to **Processed**. This step selects the overdue processes to include and creates the internal activation triggers. It also stamps the batch code for the next step and its current run number.
  - **C1-AOPFR** (Activate Overdue Process from Trigger). This batch job selects overdue controls that are stamped with this batch code and the current run number and calls the standard event activation logic for each overdue process.

NOTE: The recommendation is to schedule these batch jobs in the nightly run.

Setting Up Overdue Processing

The topics in this section describe how to set up the control tables to implement your overdue processing and collection procedures.

Setting Up Overdue Event Types

An overdue event type encapsulates the business rules that govern a given type of overdue event. Open this page by selecting **Admin Menu** > **Overdue Event Type**.

NOTE: **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.

Use **Event Restart** to indicate if the ability to restart an event of this type during overdue correction is **Applicable** or **Not Applicable**.

FASTPATH: Refer to *Overdue Correction* for more information.

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***
Cancel Logic	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.
Event Activation	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.
Event Information	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.
Event Reversal	Optional - only used if your implementation supports overdue correction.	This algorithm is executed to reverse the effect of an overdue event. Refer to *Reversing Events* and *Overdue Correction* for the details. Click *here* to see the algorithm types available for this system event.
Monitor Waiting Events	Optional - only used if events of this type can enter the **Waiting** state	This algorithm is invoked by the Overdue 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 this page by selecting **Admin** > **Overdue Process Template**.

NOTE:

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 assessment FTs, you'd reference a foreign key characteristic that references the FT 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*
Calculate Unpaid & Original Amount	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-product *overdue info string*. Click *here* to see the algorithm types available for this system event.
Cancel Criteria	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.
Cancel Logic	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.
Hold Event Activation Criteria	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 events should be suspended. Refer to *Holding Events* for the details. Click *here* to see the algorithm types available for this system event.
Overdue Process Information	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 **Dep 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 **Dep 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 **Dep 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 Collection Classes

Every account has a collection class. This class is one of several fields that control the collection method applied to the account's debt. Open **Admin Menu, Collection Class** to define your collection classes.

Description of Page

Enter a unique **Collection Class** code and **Description** for each collection class.

Indicate a **Collection Method** of **Overdue** if the accounts belonging to this collection class are subject to overdue collection procedures. If accounts belonging to this collection class are not subject collection, select **Not Eligible For Collection**. Please be aware that these accounts will NOT be reviewed for overdue debt.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_COLL_CL*.

Setting Up Collection Class Overdue Rules

Collection class overdue rules contain algorithm that impact accounts associated with a given collection class, division and currency code are managed. Open this page by selecting **Admin,Collection Class Overdue Rules**.

NOTE:
Recommendation. Before using this transaction, we strongly recommend that you review *Different Overdue Rules For Different Taxpayers*.

Description of Page

Enter the **Collection Class**, **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*
Overdue Monitor Rule	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.

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 this page by selecting **Admin,Overdue Event Cancel Reason**.

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 Collection Case Types

A **Collection Case Type** defines the configuration information that is common to collection cases of a given type. The type of information captured on the collection case type is governed by the collection case type's business object.

To set up a Collection Case Type, select **Admin Menu, Collection Case Type**.

The topics in this section describe the base-package zones that appear on the Collection Case Type portal.

Collection Case Type List

The Collection Case Type List zone lists every collection case type. The following functions are available:

- Click a *broadcast* button to open other zones that contain more information about the adjacent collection case type.
- The standard actions of **Edit, Duplicate** and **Delete** are available for each collection case type.
- State transition buttons are available to transition the collection case type to an appropriate next state.
- Click the **Add** link in the zone's title bar to add a new collection case type.

Collection Case Type Actions

This is a standard actions zone. The Edit, Delete and Duplicate actions and appropriate state transition buttons are available.

Collection Case Type

The Collection Case Type zone contains display-only information about a Collection Case Type. This zone appears when a Collection Case Type has been broadcast from the Collection Case Type List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Collection Case Type Log

This is a standard *log zone*.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *CI_COLL_CASE_TYPE* and *CI_COLL_CASE*.

Setting Up Collection Agencies

You must set up a collection agency for each such organization to which you refer delinquent debt. To define a collection agency, select **Admin Menu, Collection Agency**.

Description of Page

Enter an easily recognizable **Collection Agency** code and **Description** for each collection agency.

A collection agency must be associated with a Person. Choose the **Person ID** of the organization from the prompt.

FASTPATH:
Information about how to set up persons is discussed in *Maintaining Persons*.

Turn on the **Active** switch if the collection agency is actively receiving referrals.

Specify the **Batch Control** that's used to route new and cancelled referrals to the collection agency. The batch control's description is displayed adjacent.

Where Used

Collection agencies are assigned to collection agency referrals when the collection agency referral background process executes. Refer to *The Big Picture Of Collection Agency Referrals* for more information.

Setting Up Overdue Control Types

An Overdue Control Type defines the configuration information that is common to overdue controls of a given type. Refer to *The Big Picture of Overdue Control* for more information.

The type of information captured on the overdue control type is governed by the overdue control type's business object.

To set up an Overdue Control Type, select **Admin Menu** > **Overdue Control Type**.

The topics in this section describe the base-package zones that appear on the Overdue Control Type portal.

Overdue Control Type List

The following functions are available:

- Click a *broadcast* button to open other zones that contain more information about the adjacent overdue control type.
- Click the **Add** link in the zone's title bar to add a new overdue control type.

Overdue Control Type

The Overdue Control Type zone contains display-only information about an Overdue Control Type. This zone appears when an Overdue Control Type has been broadcast from the Overdue Control Type List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *C1_OD_CTRL_TYPE*.

Chapter 15

Defining Bankruptcy Options

A bankruptcy is a legal procedure for dealing with taxpayer debt problems. The goal of a bankruptcy process is to relieve the debtor of some if not all of his/her debt either through liquidation of assets or readjustment of debts.

The Big Picture Of Bankruptcy

The bankruptcy process begins when the debtor files a petition.

The bankruptcy petition can include the following information:

- A list of all creditors and the amount and nature of their claims
- Source, amount and frequency of income
- List of all property
- List of monthly living expenses, such as food, clothing, shelter, utilities, taxes, transportation, medicine, etc.

The petition date usually determines which of the debtor's obligations can be included in the bankruptcy. Any outstanding obligations as of the petition date are typically included.

Types Of Bankruptcy

Bankruptcy law typically classifies bankruptcy cases into certain types.

For instance, in the United States, bankruptcy cases are classified into 'chapters', the common ones being:

- Chapter 7 - Liquidation of assets
- Chapter 11 - Rehabilitation / reorganization for businesses
- Chapter 13 - Rehabilitation for individuals

The other chapters are:

- Chapter 9 - Bankruptcy for municipalities
- Chapter 12 - Rehabilitation for family farmers and fishermen

- Chapter 15 - Ancillary / international cases

Bankruptcy Courts

The bankruptcy petition is filed with the appropriate bankruptcy court, which assigns a case number/identifier, reviews the petition and eventually makes a ruling on the bankruptcy.

Key information about a court includes: court name, address, phone numbers, email addresses, web addresses, etc.

Bankruptcies Cover Obligations

A bankruptcy defines which of the debtor's obligations are covered.

The rules for determining eligible obligations are configurable in a plug-in spot on bankruptcy type. The base sample algorithm selects the taxpayer's obligations as of the petition date based on a specified reference date - e.g. obligation start date, obligation end date or obligation filing due date (for filing-based obligations only).

See *Setting Up the 'Determine Eligible Obligations'* for more information on how to configure this algorithm.

Other Related Persons On A Bankruptcy

Bankruptcies also identify other parties that are related to the case - e.g. attorney, trustee, administrator, etc. The information captured for each of these persons include: name, address, phone numbers, email addresses, etc.

Lifecycle Of A Bankruptcy

The steps in the bankruptcy process vary based on local bankruptcy laws.

The base product provides a bankruptcy business object **C1-Bankruptcy** that includes a number of common steps.

The bankruptcy is created in a **Pending** state that allows for all pertinent information to be set up before the bankruptcy takes effect. A **Pending** bankruptcy can be **Canceled**.

The bankruptcy goes to a **Review In Progress** step and stays in that state while the case is waiting for a decision from the court. This is the state where the bankruptcy is considered to be in effect. Therefore, any related suppressions on collection activity or penalty & interest are typically in place by this time. Proofs of claim can also be created at this point.

A bankruptcy case could result in any of the following decisions:

- Additional actions prior to discharge / dismissal - Depending in the type of bankruptcy, an agreed-upon payment plan may be established, to allow the debtor to repay some or all of his/her debt. The bankruptcy sits in an **Actions In Progress** state, while the payment plan is ongoing. The fulfillment of the payment plan typically leads to discharge. On the other hand, if the payment plan is not fulfilled, the bankruptcy could be dismissed or discharged, depending on the debtor's situation.
- **Discharge** - This action releases the debtor from personal liability from specific debts and prohibits creditors from taking any action against the debtor to collect those debts. In other words, the debt is written off. Certain types of debts may be deemed non-dischargeable, and thus, continue to be collectible. Any related suppression(s) are end-dated when discharge occurs.
- **Discharge Denial** - The case can be denied discharge for reasons involving fraud, such as: transferring / destroying / concealing property within a certain period before or after filing bankruptcy, false oath / claim, concealed / falsified books on financial position, etc. No write offs are done in this case.

- **Dismissal** - The case can be dismissed for reasons such as failure to provide requested tax documents, failure to make payments under the confirmed plan, failure to pay administrative fees, debtor previously received a discharge for a similar bankruptcy case within a certain time period, etc. No write-offs are done in this case.

The bankruptcy can be **Closed** after all necessary post-decision tasks have been completed.

Refer to the **C1-Bankruptcy** business object metadata for an illustration of the bankruptcy lifecycle.

Creating Proofs Of Claim

A proof of claim is a written statement describing the reason(s) a debtor owes the creditor money. The creditor files this document with the court and/or the administrator of the bankruptcy case. In most cases, the proof of claim ensures that every creditor is given due consideration when resolving the debt. Some types of debts are given more priority than others. So the proof of claim is one of the most important documents that are reviewed in bankruptcy proceedings.

The proof of claim may be deemed filed if the debt appears in the bankruptcy petition's list of creditors. Creditors can file additional proofs of claim for any debt that may not be included in the petition.

The **C1-Bankruptcy** business object provides for an ability to create proofs of claims as customer contacts. The extract logic, however, is assumed to be the implementation's responsibility.

Bankruptcies Can Cause Suppression

When obligations are included in a bankruptcy case, penalty and interest and collection activity can be suppressed for these obligations.

The type of bankruptcy determines whether or not suppression is applicable. Bankruptcy types that cause suppression should specify a suppression type.

Suppressions can be automatically created when the bankruptcy goes into a certain state. The **C1-Bankruptcy** business object has an algorithm *C1-BK-CRTSPR* that creates and activates suppression when the bankruptcy enters the **Review In Progress** state.

During the course of the bankruptcy case, obligations may get added to or removed from the bankruptcy. In some exceptional cases, the petition date may change, which may cause some obligations to become ineligible. If suppression already exists, adding / removing obligations to / from the bankruptcy will also add / remove them to / from the suppressed entities.

Suppressions can be automatically released when the debt is discharged. Your implementation will need to build your discharge processing logic (e.g. write off) based on your specific business rules. That logic can be plugged in directly on the Discharged state or configured in overdue processing.

The base product provides two algorithms that you can use, depending on the option you choose:

- *C1-BK-RLSSPR* releases the related suppression when the bankruptcy enters a certain state. Plug this in the **Discharged** state.
- *C1-OE-RLBKSU* is an overdue event activation algorithm that releases the related suppression when the overdue event is triggered.

Suppression can be automatically canceled when the bankruptcy is canceled. The **C1-Bankruptcy** business object has an algorithm *C1-BK-CNLSPR* that does this.

Refer to *The Big Picture of Suppression* for more information about suppression.

Bankruptcies Can Create Pay Plans

Payment plans can be created as a result of bankruptcy. Certain types of bankruptcy, specifically those that seek readjustment of debts typically require that the debtor fulfill an agreed-upon payment plan.

The debt is usually not discharged until the payment plan is fulfilled. In some exceptional cases, the debtor may declare hardship, in which case, the court may decide to discharge the debt anyway.

The **C1-Bankrutpcy** business object provides for an ability to create pay plans for the bankruptcy. Creating a pay plan for the bankruptcy results in the cancellation/stopping of any pay plans that existed before the bankruptcy and that cover any of the obligations that are included in the bankruptcy. It also has logic to prevent automatic discharge when the bankruptcy-related pay plans are not yet fulfilled.

Assigning Bankruptcies To Responsible Users

One or more tax authority users may be assigned as responsible users on a bankruptcy case.

A responsible user can be assigned automatically when the bankruptcy is created. The base algorithm *C1-BK-ADDUSR* assigns the user who created the bankruptcy as a responsible user. This algorithm is plugged in on **C1-Bankrutpcy** business object's **Pending** state.

Setting Up Bankruptcy Options

The topics in this section describe the objects that must be defined as part of bankruptcy processing setup.

Setting Up Bankruptcy Types

Bankruptcy Types contain the rules that control how bankruptcies are processed.

To set up a Bankruptcy Type, select **Admin Menu** > **Bankruptcy Type**.

The topics in this section describe the base-package zones that appear on the Bankruptcy Type portal.

Bankruptcy Type List

The Bankruptcy Type *List zone* lists every bankruptcy type. The following functions are available:

- Click a *broadcast* button to open other zones that contain more information about the adjacent bankruptcy type.
- The standard actions of **Edit, Delete** and **Duplicate** are available for each bankruptcy type.
- Click the **Add** link in the zone's title bar to add a new bankruptcy type.

Bankruptcy Type

The Bankruptcy Type zone contains display-only information about a Bankruptcy Type. This zone appears when a Bankruptcy Type has been broadcast from the Bankruptcy Type List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *C1_BANKRUPTCY_TYPE*.

Setting Up The 'Determine Eligible Obligations' Algorithm

The 'Determine Eligible Obligations' algorithm suggests a list of eligible obligations during bankruptcy maintenance.

A base sample algorithm *C1-BKTY-SEOP* selects obligations as of the petition date.

To use this algorithm, open **Admin Menu** > **Algorithm**. Select the base algorithm type and configure the parameters accordingly.

Plug in the algorithm on your bankruptcy types.

Setting Up Suppression Types

Bankruptcy types that create suppression must specify a suppression type. Bankruptcy-related suppression types must specify a suppression entity level of **Obligation** .

Refer to *Suppression: Setting Up Suppression Types* for more details on setting up suppression types.

Setting Up Customer Contact Types For Proofs of Claim

To create proofs of claim using customer contacts, specify the customer contact type on the bankruptcy type. The Create Proof of Claim action on the bankruptcy will use this customer contact type.

You will need to build the extract logic for your 'proof of claim' customer contact types.

Refer to *Setting Up Customer Contact Options* for more details on customer contact types.

Setting Up Pay Plan Types And Pay Plan Recommendation Rules

Bankruptcy types that allow pay plans must specify a pay plan obligation type. In addition, if you need the system to suggest a payment schedule for bankruptcy-related pay plans, configure your pay plan recommendation rules accordingly.

Refer to *Setting Up Pay Plan Options* for more details on configuring for pay plans.

Setting Up Overdue Process Templates For Bankruptcy Discharge

If you opt to use overdue processing for discharging debt, you can specify the overdue process template for discharge on the bankruptcy type.

You can also use the following algorithms

- *C1-OE-RLBKSU* releases the related suppression when the overdue event is triggered. Plug this in your overdue process template as one of the last steps.
- *C1-BK-CRTODP* creates an overdue process when the bankruptcy enters a certain state. Plug this in the bankruptcy's Discharged state.

  Refer to *Creating Overdue Procedures* for more information on configuring overdue processing options.

Setting Up Bankruptcy Courts

Courts are set up as special types of Persons in the system. To configure court information:

- Set up a Person Type for courts. This could be a general one for all types of courts or a specific one for bankruptcy courts. You can reference the base business objects **C1-BusinessPersonType** or **C1-BusinessPerson** or your own business objects in this person type. Refer to *Defining Person Types* for more information.
- Create Person records for your bankruptcy courts, using the special Person Type you configured.

Setting Up Bankruptcy Filing Types

If you classify your bankruptcies into broad or specific categories, you can configure those categories as Bankruptcy Filing Type extendable lookup values.

- Open **Admin Menu** > **Extendable Lookup**.
- Search for and select the **Bankruptcy Filing Type** extended lookup business object.
- The list of existing bankruptcy filing types are displayed in a standard *List zone*.
- Choose an existing bankruptcy filing type to view, edit, delete or duplicate.
- Use the **Add** link in the zone header to create a new bankruptcy filing type.

Setting Up Bankruptcy Claim Types

If you classify obligations into specific type of claims (e.g. for 'proof of claim' purposes), you can configure those types as Bankruptcy Claim Type extendable lookup values.

- Open **Admin Menu** > **Extendable Lookup**.
- Search for and select the **Bankruptcy Claim Type** extended lookup business object.
- The list of existing bankruptcy claim types are displayed in a standard *List zone*.
- Choose an existing bankruptcy claim type to view, edit, delete or duplicate.
- Use the **Add** link in the zone header to create a new bankruptcy claim type.

Setting Up Milestone Types

You can define the types of milestone dates that can be captured on your bankruptcies as Milestone Type extendable lookup values.

- Open **Admin Menu** > **Extendable Lookup**.
- Search for and select the **Milestone Type** extended lookup business object.
- The list of existing milestone types are displayed in a standard *List zone*.
- Choose an existing milestone type to view, edit, delete or duplicate.
- Use the **Add** link in the zone header to create a new milestone type.

Setting Up Bankruptcy Relationship Types

Base values for Bankruptcy Relationship Types are supplied. You can add your specific relationship types by updating the value for the BANKRUPTCY_REL_TYPE_FLG *lookup* field.

Setting Up Assignment Roles

Base values for responsible user's Assignment Roles are supplied. You can add your specific assignment roles by updating the value for the ASSIGNMENT_ROLE_FLG *lookup* field.

Setting Up An Alert For Highlighting Open Bankruptcies

To show an alert when the person in context has any open bankruptcies, define an algorithm for the *C1-BNKR-OPN* base algorithm type and configure the alert in Installation Options.

To configure the alert, open **Admin Menu** > **Installation Options - Framework** and configure the algorithm you defined in the **Alert** system event.

Chapter 16

Defining Audit Case Options

Tax audits are conducted to examine the accuracy of reported tax information. It could be as simple as reviewing the accuracy of the assessment on a single tax return or as complex as reviewing a number of obligations for specific filing periods.

Audits are also conducted when the tax authority finds that a taxpayer has failed to file certain tax returns that he/she is expected to file.

The Big Picture Of Audit Cases

A tax authority could have a number of reasons for initiating an audit. Internal reports and reviews usually identify specific taxpayers and/or accounts that need to be audited. The selection of taxpayers and/or accounts is based on specific criteria that the tax authority defines.

External interfaces could also send reports / information that prompt a tax authority to subject certain taxpayers to audits. The most common situation is when income information is reported for a taxpayer, who has either not filed a return or has filed but did not include that information.

Obligations Are Audited

An audit case identifies the primary account that is being audited. Any of the obligations of the primary taxpayer and other financially responsible persons on that primary account can be examined during the audit.

Other Related Persons On An Audit Case

Audit cases can also specify other parties that are related to the case. For instance, if the taxpayer is represented by an attorney, accountant, enrolled actuary, enrolled agent, paid preparer, etc., that representative is a related person on the case.

The information captured for the related persons can include: names, addresses, phone numbers, email addresses, etc.

Preparing For An Audit

An auditor may need to perform a number of tasks prior to the audit proper. These tasks can include: examining the taxpayer's information, looking at the overall picture of accounts, examining the latest tax forms, identifying any missed tax returns, creating obligations, etc.

Initial correspondence or meetings may also be conducted, to explain the audit process and request for additional documentation that will be needed during the audit.

Conducting Audits

There are three general methods of conducting audits:

Desk Audits

This method uses correspondence / letters to ask for additional information or to notify the taxpayer of a change in the assessment. In the latter case, the taxpayer responds by either sending in a payment or filing for an appeal.

Office Audits

In this method, the taxpayer or an authorized representative is asked to go to the tax authority's office and bring the necessary documentation.

Field Audits

This is considered the most aggressive method. A field audit officer goes to the taxpayer's or the authorized representative's residence or office. Tax-related information is examined on site.

Lifecycle Of An Audit Case

The steps in an audit case may vary depending on the type of audit case.

The base product provides an audit case business object **C1-AuditCase** that includes a number of common steps.

The audit case is created in a **Pending** state to allow for any preparatory tasks prior to starting the audit.

When the audit process starts, the audit case goes to the **Audit In Progress** state. If suppression is applicable, the suppression is typically created and activated at this point. The audit case stays in this state until findings are identified. The most common result of an audit is a new assessment. This happens when either an existing tax form is audited or when the auditor creates an audit form for tax returns that the taxpayer failed to file.

The findings of the audit can be subjected to any required reviews and/or approvals while the audit case is in the **Review In Progress** state. If there are multiple review steps, a separate Review process can be created from the audit case. Refer to *Creating Audit Case Reviews* for more information on audit case reviews.

If the findings do not pass the review - e.g. the proposed audit assessment is incorrect - the audit case can go back to the **Audit In Progress** , so that the necessary changes can be made. The audit case can cycle through the **Audit In Progress** and **Review In Progress** states until the findings pass the reviews, in which case, the audit case goes to the **Final Decision** state.

Any related suppressions are released when the audit case is **Closed** .

Audit cases that are in the **Pending** , **Audit In Progress** or **Review In Progress** states can be **Canceled** . Audit case cancellation also cancels any related suppressions.

Refer to the **C1-AuditCase** business object metadata for an illustration of the audit case lifecycle.

Audit Cases Can Cause Suppression

Certain processes or activity may be suppressed while an audit case is ongoing. These processes can include penalty & interest calculation, overdue processing and overpayment processing.

The type of audit case determines whether or not suppression is applicable. Audit case types that cause suppression should specify a suppression type.

Suppressions can be automatically created when the audit case goes into a certain state. The **C1-AuditCase** business object has an algorithm *C1-ADCS-CRSP* that creates and activates suppression when the audit case enters the **Audit In Progress** state.

Changes to the audit start date cause the related suppression's start date to also change.

While an audit is in progress, obligations may get added to or removed from the audit case. If suppression already exists, adding / removing obligations to / from the audit case will also add / remove them to / from the suppressed entities.

Suppressions can be automatically released when the audit case is closed. The **C1-AuditCase** business object has an algorithm *C1-ADCS-RLSP* that does this.

Suppression can be automatically canceled when the audit case is canceled. The **C1-AuditCase** business object has an algorithm *C1-ADCS-CNSP* that does this.

Refer to *The Big Picture of Suppression* for more information about suppression.

Audit Assessments

When the audit identifies necessary changes to the information that the taxpayer originally reported, it usually results in the calculation of a new assessment.

The most common way of determining this new assessment is through an audit form. The form could be the same form type as the one that the taxpayer filed (except that the form is marked as an 'audit' form) or it could be a special audit form type. In either case, the form is brought to a state where the user can review the details of the proposed assessment, see the potential impact to the obligation's balance and make additional changes, as needed.

In some exceptional cases, the adjustment may be created manually (i.e. not via tax form) or certain changes to the taxpayer information are made to trigger a change in the assessed tax (e.g. removing an exemption).

There are two common scenarios for initiating audit forms:

- The taxpayer being audited did not file a tax return. In this case, the auditor creates the tax form.
- An existing tax return is audited. The auditor creates the audit form by copying the details from the existing form and making the necessary changes. When the audit form posts, the assessment adjustment is created for the difference between the prior assessment and the new assessment.

The **C1-ParentTaxForm** business object provides an Audit action that allows a user to create an audit form from an existing posted form. Any tax forms that inherit from this business object will have the Audit action available. Refer to *Auditing Tax Forms* for more information on auditing existing tax forms.

The audit form is usually not posted until it has gone through proper approval / review and until the taxpayer has accepted the new assessment.

Should the taxpayer disagree with the changed assessment, he/she can file for an appeal. The tax authority may or may not choose to keep the audit open while the appeal is in progress. The audit assessment is not applied to the obligation's balance until a decision is made on the appeal. If the appeal is upheld, the new assessment is not created. On the other hand, if the appeal is denied, the new assessment is posted. Refer to *The Big Picture of Appeals* for more information on appeals.

Audit Case Reviews

The **C1-AuditCase** business object provides for an ability to create reviews from an audit case. This action is available when the audit case's current state supports it.

Refer to *Creating Audit Case Reviews* for more information on how to create audit case reviews.

Reviews have their own lifecycles, which depend on the associated business object. The **C1-AuditCaseReview** business object supplied in base defines a typical lifecycle for an audit case review. It follows a basic lifecycle, as described in *Review Lifecycle*. This business object also has logic to automatically transition the audit case when the last review completes.

Refer to *The Big Picture of Reviews* for more information on reviews.

Assigning Audit Cases To Responsible Users

One or more tax authority users may be assigned as responsible users on an audit case.

A responsible user can be assigned automatically when the audit case is created. The base algorithm *C1-ADCS-ASUR* assigns the user who created the audit case as a responsible user. This algorithm is plugged in on **C1-AuditCase** business object's **Pending** state.

Setting Up Audit Case Options

The following sections describe the objects that must be defined as part of the audit case processing setup process.

Setting Up Audit Case Types

An Audit Case Type defines the configuration information that is common to bankruptcies of a given type. The type of information captured on the audit case type is governed by the audit case type's business object.

To set up an Audit Case Type, select **Admin Menu** > **Audit Case Type**.

The topics in this section describe the base-package zones that appear on the Audit Case Type portal.

Audit Case Type List

The Audit Case Type *List zone* lists every audit case type. The following functions are available:

- Click a *broadcast* button to open other zones that contain more information about the adjacent audit case type.
- The standard actions of **Edit, Delete** and **Duplicate** are available for each audit case type.
- Click the **Add** link in the zone's title bar to add a new audit case type.

Audit Case Type

The Audit Case Type zone contains display-only information about an Audit Case Type. This zone appears when an Audit Case Type has been broadcast from the Audit Case Type List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *C1_AUDIT_CASE_TYPE*.

Setting Up Suppression Types

Audit case types that create suppression must specify a suppression type. Audit-case-related suppression types must specify a suppression entity level of **Obligation** .

Refer to *Suppression: Setting Up Suppression Types* for more details on setting up suppression types.

Setting Up Customer Contact Types For Letters

Define the customer contact types for each type of letter that can be generated from the audit case. Build the extract logic for these customer contact types and define the letter templates accordingly.

Refer to *Setting Up Customer Contact Options* for more details on customer contact types.

Setting Up Review Process Controls For Audit Case Reviews

Audit case types that require reviews must specify a review process control, which defines the types of reviews that are required and the sequence in which they are conducted. The audit case's review process control will default from the value on the audit case type when the audit is created. This default can be overridden.

Refer to *Setting Up Review Options* for details on setting up Review Types and Review Process Controls.

Setting Up Audit Case Reasons

Identify the various reasons for creating an audit case and configure those reasons as Audit Case Reason extendable lookup values.

- Open **Admin Menu** > **Extendable Lookup**.
- Search for and select the **Audit Case Reason** extendable lookup business object.
- The list of existing audit case reasons are displayed in a standard *List zone*.
- Choose an existing audit case reason to view, edit, delete or duplicate.
- Use the **Add** link in the zone header to create a new audit case reason.

Setting Up Audit Case Sources

If your audit cases can be triggered from a number of different sources, you can configure those sources as Audit Case Source extendable lookup values.

- Open **Admin Menu** > **Extendable Lookup**.
- Search for and select the **Audit Case Source** extendable lookup business object.
- The list of existing audit case sources are displayed in a standard *List zone*.
- Choose an existing audit case source to view, edit, delete or duplicate.
- Use the **Add** link in the zone header to create a new audit case source.

Setting Up Audit Case Relationship Types

Base values for Audit Case Relationship Types are supplied. You can add your specific relationship types by updating the value for the AUDIT_CASE_REL_TYPE_FLG *lookup* field.

Setting Up Assignment Roles

Base values for responsible user's Assignment Roles are supplied. You can add your specific assignment roles by updating the value for the ASSIGNMENT_ROLE_FLG *lookup* field.

Setting Up An Alert For Highlighting Open Audit Cases

To show an alert when the account in context has any open audit cases, define an algorithm for the *C1-CCAL-ADCS* base algorithm type and configure the alert in Installation Options.

To configure the alert, open **Admin Menu** > **Installation Options - Framework** and plug in the algorithm you defined in the **Alert**system event.

Chapter 17

Defining Appeal Options

An appeal is an administrative review process used to manage a request from a taxpayer for a change to an official decision by a tax authority. The topics in this section describe how to configure the system to manage appeals.

The Big Picture of Appeals

A taxpayer can lodge an appeal for a variety of reasons. They may wish to dispute certain specific assessments or they may wish to appeal a number of charges related to an assessment such as systemic penalty and interest amounts. If tax returns are adjusted by tax authority staff or new obligations created as a result of an audit case, new assessments may be created that could be disputed.

An appeal process is used to capture the information related to an appeal, track the events involved in reviewing the appeal and record the decision made by each review process.

Taxpayer and Other Related Persons On An Appeal

An appeal process must be linked to a taxpayer. The taxpayer who lodged the appeal can nominate themselves as the primary contact or delegate to an authorized representative such as an accountant. The base package provides the ability to link other persons to the appeal, such as persons who are also financially responsible for the disputed charges or additional representatives such as attorneys. Each related person has a nominated relationship type.

Appeals Are Assigned to Responsible Users

One or more tax authority users may be assigned as responsible users on an appeal.

A responsible user can be assigned automatically when the appeal is created. The base algorithm *C1-AP-ADDUSR* assigns the user who created the appeal as a responsible user with an assignment role of **Primary** .

Appeals Can Cause Suppression

The obligations related to an appeal are usually suppressed from any collection activity or overpayment processing. The system provides the ability to specify which obligations should have activity suppressed as a result of the appeal.

Appeal types that require suppression need to specify a suppression type. The base algorithm *C1-AP-CRTSPR* automatically creates and activates a suppression object for the obligations linked to the appeal when the appeal enters a given state. The **C1-Appeal** business object is configured to create suppression when the appeal enters the **Ready For Review** state.

If an obligation is added to or removed from the appeal, the suppression object is updated accordingly. Refer to base algorithm *C1-AP-PSUCHG* for more details.

The base algorithm *C1-AP-CNLSPR* automatically cancels all suppression objects linked to the appeal when the appeal enters a given state. The **C1-Appeal** business object is configured to create suppression when the appeal enters the **Canceled** state.

The base algorithm *C1-AP-RLSSPR* automatically releases all suppression objects linked to the appeal when the appeal enters a given state. The **C1-Appeal** business object is configured to create suppression when the appeal enters the **Closed** state.

Refer to *Defining Suppression Options* for an overview of Suppression and details on setting up Suppression options.

Related Objects On An Appeal

An appeal can be linked to a number of related objects. Examples include the adjustments or assessments that the taxpayer is contesting, related documents, audit cases or audit forms.

You define appeal related object types by defining the characteristic used to reference the object and linking that characteristic to the appeal related objects entity. For example, to link an adjustment to an appeal you would first define a foreign key characteristic that references the adjustment object. Refer to *Characteristic Types & Values* for more information on setting up characteristic types.

The objects related to an appeal have an associated category. Valid values are defined using the Appeal Related Object category lookup field.

Validating an Appeal

Tax authorities may have rules governing whether an appeal can be accepted for processing. An appeal may be initially assessed to see whether it complies with these rules to determine if it should be rejected without commencing the review or evaluation process. The **C1-Appeal** business object includes a validation state to support this.

It is common for tax authorities to impose time limits on lodging an appeal. The base algorithm *C1-AP-CHKATL* provides an example of logic that validates an appeal's timeliness via a call to an Oracle Policy Automation rule. Refer to *Configuring the Appeal Timeliness Rule* for more details.

If issues are reported during validation, the **C1-Appeal** business object will transition to the **Issues Detected** state. The base algorithm *C1-AP-CRTODO* may be configured on the **Issues Detected** state to send a notification that the appeal is invalid. If the appeal passes validation, the business object enters the **Ready For Review** state.

The **C1-Appeal** business object allows an appeal to be manually transitioned to the **Ready For Review** state from the **Issues Detected** state if a user wishes to override the validation rules. Your implementation may choose to restrict this action to specific user groups using standard application service security configuration.

Canceling an Appeal

The **C1-Appeal** business object provides the ability to cancel an appeal at any point prior to commencing the review process. A monitoring algorithm may be configured to automatically cancel an appeal that has failed validation. The base algorithm *C1-AP-TRNISS* provides an example of logic that transitions an appeal to a canceled state if the appeal has failed due to a specific issue and sufficient time has elapsed to allow for follow up.

Appeals Follow a Defined Review Process

Once an appeal is accepted, it will be reviewed internally by the tax authority. This may involve a conference or hearing with the taxpayer. Once the tax authority has made a decision, the taxpayer may have the option of referring the appeal to other boards of review or judicial courts. The types of reviews available and the order in which the taxpayer can escalate the appeal normally follow a defined sequence.

The system provides the ability to define different types of reviews and a process control that specifies what review types are allowed and in what order for a given appeal process. Appeal types define the default review process control but the user may override this for a particular appeal.

The **C1-Appeal** business object provides a **Create Review** action that allows the user to add a new review sub-process. Users are presented with a list of the next allowable review steps, based on the reviews already undertaken for the appeal. Only one review can be active for an appeal at a time, so the action is only available if the most recent review linked to the appeal is closed.

An appeal moves from the **Ready For Review** state to **Review In Progress** when the first review is created. The appeal remains in that state, allowing the user to continue to add successive review events until the taxpayer accepts the response to the latest review or the appeal has passed through all possible internal and external review types.

Refer to *Defining Review Options* for more information on configuring reviews.

Appeal Reviews

The lifecycle of a review depends upon the configuration of the associated business object. The base **C1-AppealReview** business object defines a typical review for an appeal process. It follows a basic lifecycle, as described in *Review Lifecycle*.

In addition to the common functionality, a base algorithm is provided to create a customer contact for an appeal based on the response type. The base algorithm type *C1-RVW-CCCRR* is designed to be configured on the appeal **Response Recorded** state.

Closing Appeals

Once the last review process is complete, the appeal can be manually transitioned to the **Final Decision** state. The base algorithm *C1-AP-CINCRE* provides the ability to prevent an appeal from being transitioned to **Final Decision** before the most recent review is complete.

The appeal remains in the **Final Decision** state while users make the appropriate corrections to the taxpayer's account, depending on the outcome of the appeal. When no further action is required, the appeal can be transitioned to **Closed** .

The base algorithm *C1-AP-CPTODO* can be configured on the **Closed** or **Canceled** states to complete any outstanding to do entries for the appeal .

Monitoring Appeals

The actions required to progress an appeal are mostly manual. An appeal may be waiting for a review to be scheduled. Appeals can remain open while the taxpayer has the ability to escalate the appeal in response to the most recent decision. Periodic monitoring provides the ability to check whether an appeal has not progressed as expected.

The base algorithm *C1-AP-APWL* provides the ability to send a notification that the appeal has been waiting in a given state for too long. The algorithm is designed to be configured on the **Ready For Review** and **Final Decision** states.

The base algorithm *C1-RVW-CCCRR* provides the ability to send a notification that the appeal has been waiting in the **Review In Progress** state for too long after the latest review was completed.

Setting Up Appeal Options

The following sections describe the objects that must be defined as part of the appeal processing setup process.

Setting Up Appeal Types

Appeal Types contain the rules that control how appeals are processed.

To set up an appeal type, select **Admin Menu** > **Appeal Type**.

The topics in this section describe the base-package zones that appear on the Appeal Type portal.

Appeal Type List

The Appeal Type *List zone* lists every appeal type. The following functions are available:

- Click a *broadcast* button to open other zones that contain more information about the adjacent appeal type.
- The standard actions of **Edit, Delete** and **Duplicate** are available for each appeal type.
- Click the **Add** link in the zone's title bar to add a new appeal type.

Appeal Type

The Appeal Type zone contains display-only information about an appeal type. This zone appears when an appeal type has been broadcast from the Appeal Type List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *C1_APPEAL_TYPE*.

Setting Up Review Process Controls

An appeal type must be linked to a review process control, which defines the types of reviews required to process the appeal and the sequence in which they are conducted. The appeal's review process control will default to this value when an appeal of this type is added but may be overridden.

Refer to *Setting Up Review Options* for details on setting up Review Types and Review Process Controls.

Setting Up Suppression Types

An appeal type may be optionally linked to a suppression type, which defines the suppression entity level and the processes to be suppressed. If an appeal type references a suppression type, a suppression object of that type is automatically created when the appeal enters the appropriate state. Appeals suppress activity on selected obligations, so appeal types should reference suppression types with an entity level of **Obligation** .

Refer to *Setting Up Suppression Options* for details on setting up Suppression Types.

Setting Up Customer Contact Types

The base Review Response Type extendable lookup business object allows users to specify an optional customer contact type and class to be used when notifying taxpayers of the response given to their appeal review.

To set up a customer contact type, select **Admin Menu** , **Customer Contact Type** . Refer to existing help on setting up customer contact types.

Setting Up an Alert to Highlight Open Appeals

The base algorithm type *C1-AP-OPEN* provides the ability to create an Alert if open appeals exist for the person or account in context.

To configure the alert, navigate to **Admin** > **Menu, Installation Options - Framework** and configure the base algorithm using system event **Alert**.

Configuring the Appeal Timeliness Rule

In order to use the base algorithm type that invokes the Oracle Policy Automation rulebase to validate an appeal's timeliness, you must define additional components, as follows:

- Define a web service adapter for the rule. Refer to *OPA configuration* for more details.
- Define a message category and number to be used to indicate that the appeal has failed the rule in the appeal's issues list. To set up a message, navigate to **Admin** > **Menu, Message** and add a new message for the 'Implementer's Messages' category.
- Create an algorithm for algorithm type *C1-AP-CHKATL* that references the web service adapter, message category and number as input parameters.
- Configure the algorithm on the **Validated** state for the appropriate appeal business objects, using the **Enter** system event.

Setting Up Appeal Relationship Types

Each related person on an appeal must be assigned to a relationship type. Valid types include **Accountant**, **Attorney**, **Court**, **Financially Responsible** and **Trustee**.

Go to **Admin Menu** > **Lookup** and search for field **APPEAL_REL_TYPE_FLG** to add specific relationship types for your implementation.

Setting Up Assignment Roles

Each responsible user on an appeal must have an assignment role. Valid roles include **Primary**, **Secondary** and **Supervisor**.

Go to **Admin Menu** > **Lookup** and search for field **ASSIGNMENT_ROLE_FLG** to add specific assignment roles for your implementation.

Setting Up Appeal Related Object Categories

Each related object on an appeal must be assigned to a category. Go to **Admin Menu** > **Lookup** and search for field **APPEAL_REL_OBJ_CAT_FLG** to add specific categories for your implementation.

Chapter 18

Defining Review Options

Some common case management processes such as appeals and audits, include steps in which the case is evaluated either internally by different groups within the tax authority, or externally by other agencies, such as judicial courts. These review processes typically follow their own lifecycle and may be part of a sequence of reviews that are escalated to higher levels of authority. The topics in this section describe how to configure the system to manage reviews.

The Big Picture of Reviews

A review provides the functionality to record key dates and determinations of a review process. Reviewers may be required to make a decision by a certain date and taxpayers may be given a fixed period of time to respond. A predefined list of responses is used to ensure review decisions are described and classified consistently. External reviews may be linked to the person or entity conducting the review, such as a court.

Defining Allowable Review Types and Sequences

A case management process may trigger more than one review. The types of reviews that can occur differ according to the process type. In addition, the order in which the allowable review types are conducted often follows a defined sequence. For instance, if a taxpayer lodges an appeal with a tax authority and disagrees with their decision, they may have the right to refer the case to one or more external courts. Audit cases may need to be approved by more than one supervising group before any action is taken.

Review process controls are used to define a particular sequence or hierarchy of review types. Users can specify what review types are allowed and in what order for a specific process control. The review process control also allows the user to indicate whether a given review type may be bypassed or repeated.

Creating Reviews

A review is created as a result of a user-initiated action invoked from a case management process. The base package supports the creation of reviews from an appeal or audit case. The **Create Review** action in the appeal or audit case portal zone guides the user to select the next available review type as governed by the associated review process control. The applicable review process control is defined on the appeal or audit case type.

It is important to note that a review exists as a sub-process linked to another object and cannot be added or viewed independently of the primary process. Refer to the documentation on *Reviewing an Appeal* and *Creating Audit Case Reviews* for more details on how to create and maintain reviews.

Review Lifecycle

The lifecycle of the review depends upon the configuration of the associated business object. The base package includes review business objects that are specific to appeals and audit cases that have the same simple lifecycle, as follows:

- The review is initially created in the **Pending** state.
- A **Pending** review may be manually transitioned to **Canceled**.
- The user can manually transition the review to the **In Progress** state once key information has been recorded or dates have been set. A review that has transitioned to **In Progress** cannot be canceled—it must progress through the **Response Recorded** and **Closed** states. A monitoring algorithm can be configured to check whether the review has been waiting too long in the pending or in progress states. Base algorithm *C1-RVW-TDRPD* provides an example of this logic.
- The user can manually transition the review to the **Response Recorded** state when a decision has been reached and the review has been updated with the appropriate review response code. A monitoring algorithm can be configured to automatically transition the review to the **Response Recorded** state when the review decision date has been populated and is on or before the business date. Base algorithm *C1-RVW-TRTRR* provides an example of this logic.
- The user can manually transition the review to the **Closed** state once all activities are complete. An algorithm can be configured to complete all open to do entries that are linked to the review. Base algorithm *C1-RVW-CMPTD* provides an example of this logic.

Setting Up Review Options

The following sections describe the objects that must be defined as part of the review processing setup process.

Setting Up Review Types

Review Types contain the rules that control how reviews are processed.

To set up a Review Type, select **Admin Menu** > **Review Type**.

The topics in this section describe the base-package zones that appear on the Review Type portal.

Review Type List

The Review Type *List zone* lists every review type. The following functions are available:

- Click a *broadcast* button to open other zones that contain more information about the adjacent review type.
- The standard actions of **Edit, Delete** and **Duplicate** are available for each review type.
- Click the **Add** link in the zone's title bar to add a new review type.

Review Type

The Review Type zone contains display-only information about a Review Type. This zone appears when a Review Type has been broadcast from the Review Type List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *C1_REVIEW_TYPE* .

Setting Up Review Process Controls

Review types are logically grouped together in a review process control. The sequence of the review types in the review process control governs the order in which reviews may be created for processes associated with that control.

To set up a Review Process Control, select **Admin Menu** > **Review Process Control**.

The topics in this section describe the base-package zones that appear on the Review Process Control portal.

Review Process Control List

The Review Process Control *List zone* lists every review process control. The following functions are available:

- Click a *broadcast* button to open other zones that contain more information about the adjacent review process control.
- The standard actions of **Edit, Delete** and **Duplicate** are available for each review process control.
- Click the **Add** link in the zone's title bar to add a new review process control.

Review Process Control

The Review Process Control zone contains display-only information about the review process control. This zone appears when a review process control has been broadcast from the Review Process Control List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *C1_REVIEW_PROC_CTRL*.

Setting Up Review Response Types

Review response types are defined using an extendable lookup. To view or create review response types:

- Open **Admin Menu** > **Extendable Lookup**
- Search for and select the **Review Response Type** extendable lookup business object.
- The list of existing review response types is displayed in a standard *List zone*.
- Choose an existing review response type to view, edit, delete or duplicate.
- Use the **Add** link in the zone header to create a new review response type.

The base Review Response Type extendable lookup business object allows users to specify an optional customer contact type and class. Refer to *Appeal Reviews* for details on how customer contacts are used in appeal reviews.

Chapter 19

Defining Suppression Options

Suppression is a common function to stop certain events from taking place on a taxpayer's account. Common causes of suppression are processes such as bankruptcy cases, appeals or other internal investigations such as audit cases. The topics in this section describe how to configure the system to manage suppression.

The Big Picture of Suppressions

A tax authority may want to suppress activity on a taxpayer's account for a number of different reasons. In some cases it is a matter of internal policy, where the tax authority suppresses certain activities while processing an appeal or audit case. In other instances, it may be a statutory requirement imposed by the regulations surrounding a process initiated externally, such as a bankruptcy proceeding.

Some common types of activities that may be suppressed are penalty and interest calculation, overdue processing and overpayment processing. Levels of suppression can be large or small, encompassing a specific account, a tax role or a single obligation. Suppressions may be initiated and released manually or created and released automatically by other processes.

A suppression object is used to capture the specific entities and processes that are subject to a given suppression event.

Suppression Types

Suppression types define the rules governing how suppressions are created and managed. Certain types of suppressions may be created manually while others should only be created automatically by other processes. The processes affected by a suppression event are defined on the suppression type. Users may choose to add to or remove suppressed processes when the suppression is added manually.

Some types of suppression may be configured to be automatically released after a given period of time whereas other must be released manually.

Suppressing Process Activity

The activities that need to be suppressed can vary according to the reason for suppression - for example, if a tax authority receives notice that a bankruptcy proceeding is in place that affects taxes owed, penalty and interest (P&I) calculation and overdue processing activity must be suppressed until the bankruptcy case is closed.

In most cases, processing is put on hold starting at the suppression's start date and resumes on the suppression's release date. Penalty and interest calculations are an exception to this, as P&I may be recalculated to accommodate retroactive changes to the obligation's balance. Refer to *How Suppression Affects Penalty and Interest* for more details on P&I specific processing.

A suppression event does not actively "hold" work but rather affects how the associated processes work . It is the responsibility of the associated processes to consider effective suppressions whenever actions are initiated. The way in which suppression affects processing can differ according to the activity being suppressed. For example:

- P&I provides the ability to turn off P&I rules for the suppressed period
- Overdue Processing has a plug-in designed to hold event activation
- Business object -based processes such as overpayment processing need algorithms that suppress actions like state transitions

The system provides sample algorithms that can be used to suppress some common activities. Refer to *Configuring Overdue Processes For Suppression*, *Configuring Overpayment Processes For Suppression*, and *Configuring Obligation Types For P&I Suppression* for a description of how these processes can be suppressed.

Suppressed Entity Level Defines Suppression Scope

Suppression can apply at one of four entity levels - person, account, tax role or obligation. The base package provides common logic to determine if suppression is in effect for an entity. This logic considers all suppression records for the entity and its higher level entities.

Suppression scope can be summarized as follows:

- A person-level suppression object will also cause suppression on all the accounts for which that person has financially responsibility, all tax roles for those accounts and all obligations for those accounts.
- An account-level suppression object will also cause suppression on all tax roles for that account and all obligations for that account.
- A tax role-level suppression object will also cause suppression on all obligations for that tax role.
- An obligation-level suppression object will cause suppression only on that obligation.

It is possible for a given entity, such as an obligation, to be affected by suppressions with differing periods or at differing levels. For example, a suppression record may be manually added for an account with one or more obligations. A subsequent bankruptcy case may be created that includes one of those account's obligations. The system assumes that both suppressions are in effect for the obligation and that the period of suppression starts from the start of the earlier suppression record (for the account) until the later of the two release dates.

Note that the release date of a suppression event is the date on which the suppression ceases to have effect. Processes that are put on hold due to suppression will restart processing on the release date - for example, overdue events will be triggered on the day of release. For P&I calculations, P&I will be suppressed from the start date of the suppression until the day prior to the release date, inclusive.

Refer to the base business service **C1-GetEffectiveSuppressions** for more details on the common logic used to retrieve suppression details and periods.

How Suppression Affects Penalty and Interest

P&I processing caters for retroactive changes to an obligation's balance. It needs to consider suppressions that were in effect during the recalculation period even though the suppressions may now be released . The logic that gets effective suppressions considers both Active and Released states for P&I processing - these states are marked as 'P&I In Effect'. The result is that P&I rules are permanently turned off for the period of the suppression event.

If a suppression event is cancelled, it is regarded as if it had been deleted and no longer has an effect. Penalty and interest calculations are updated to reverse any prior effect of the suppression.

The base package provides a **Business Object - Post-processing** algorithm and a **Business Object Status - Enter** algorithm for the **C1-Suppression** business object that calculates P&I for entities affected by the suppression whenever key events occur. Refer to base algorithms *C1-SUPP-PI* and *C1-SUPP-UPI* for more details.

Your implementation may wish to prevent certain adjustments or charges being made to a taxpayer's obligations will suppression is in effect. The system provides an Adjustment Type Validation algorithm designed to return an error if suppression applied to the adjustment's obligation. Refer to *Configuring Adjustment Types For P&I Suppression* for more details.

Other Processes Create Suppression

Many suppression events will be created automatically as the result of initiating another process, such as a bankruptcy case or an appeal.

The process that causes the suppression is responsible for creating the suppression object with its associated entities and processes. The initiating process is also responsible for ongoing maintenance of the suppression object, including:

- Updating the related suppression date when a key date on the process is changed, such as an appeal submission date
- Adding or removing suppressed entities if the related entities are removed or added to the initiating process
- Cancelling or releasing the suppression at the appropriate time

Refer to *The Big Picture Of Bankruptcy*, *The Big Picture Of Appeals*, and *The Big Picture Of Audit Cases* for examples of processes that create and manage suppressions.

Auditing Suppressions

Because suppression events can cause retroactive changes to penalty and interest, key changes are audited. The base package provides **Business Object - Audit** algorithms for the **C1-Suppression** business object that log changes to the suppression dates and suppressed entities. Refer to base algorithms *C1-SUPP-DTCH* and *C1-SUPP-EDEL* for more details.

Suppression Lifecycle

The lifecycle of a suppression object depends upon the configuration of the associated business object. The base package includes a sample **C1-Suppression** business object with a simple lifecycle, as follows:

- The suppression is initially created in the **Pending** state. The suppression has no effect while in this state - the details may be edited without triggering other processing such as P&I updates. A deferred monitoring algorithm can be configured to automatically transition the suppression to **Active** if the start date is on or before the system date. Base algorithm *C1-SUPP-AC* provides an example of this logic.

- The user can manually transition the suppression to the **Active** state once the details are finalized. Suppressions that are created by other processes will typically be activated by the initiating process. An enter algorithm may be configured to trigger validation specific to the **Active** state. The base algorithm *C1-SUPP-VAAC* provides an example of this logic.
- A monitor algorithm may be configured on the **Active** state to check for suppressions that have been open too long. The base algorithm *C1-SUPP-MAXD* provides an example of this logic. An exit algorithm can be configured to complete all open monitoring to do entries when the suppression is no longer active. Base algorithm *C1-SUPP-CTDO* provides an example of this logic.
- A suppression object can be released by transitioning to the **Released** state or by updating the end date and having the suppression transition automatically. Suppressions that are created by other processes will typically be released by the initiating process. A monitor algorithm may be configured on the **Active** state to check for suppressions whose end date is populated and automatically transition them to **Released** . The base algorithm *C1-SUPP-REL* provides an example of this logic. An enter algorithm may be configured to provide a default end date or validate the existing end date for a suppression transitioned to **Released** . The base algorithm *C1-SUPP-END* provides an example of this logic.
- A suppression object can be **Canceled** from the **Pending** or **Active** states. Suppressions that are created by other processes will typically be canceled by the initiating process if appropriate.
- If the suppression affects penalty and interest, P&I should be updated for the related obligations when the suppression period starts and ends. An enter algorithm may be configured to trigger P&I updates when the suppression enters the **Active** , **Released** or **Canceled** states. Base algorithm *C1-SUPP-UPI* provides an example of this logic

Setting Up Suppression Options

The topics in this section describe the objects that must be defined as part of suppression processing setup.

Setting Up Suppression Types

Suppression Types contain the rules that control how suppressions are processed.

To set up a Suppression Type, select **Admin Menu** > **Suppression Type**.

The topics in this section describe the base-package zones that appear on the Suppression Type portal.

Suppression Type List

The Suppression Type *List zone* lists every suppression type. The following functions are available:

- Click a *broadcast* button to open other zones that contain more information about the adjacent suppression type.
- The standard actions of **Edit, Delete** and **Duplicate** are available for each suppression type.
- Click the **Add** link in the zone's title bar to add a new suppression type.

Suppression Type

The Suppression Type zone contains display-only information about a Suppression Type. This zone appears when a Suppression Type has been broadcast from the Suppression Type List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *C1_SUPPRESSION_TYPE*.

Configuring Overdue Processes for Suppression

The base package includes a sample algorithm that provides an example of logic that determines if overdue event activation should be placed on hold when suppression applies. It is designed to be used for overdue processes that collection on obligations or assessments.

In order to use the base algorithm type to suppress overdue events, you must perform the following configuration:

- Create an algorithm for algorithm type *C1-OVRD-SUPP* . Refer to the algorithm type description of an explanation of the parameters required for the algorithm and their use.
- Navigate to **Admin Menu** > **, Overdue Process Template** and configure the algorithm using the **Hold Event Activation Criteria** system event for each of the overdue process templates to which suppression applies.

Refer to *Setting Up Overdue Process Templates* for more information.

Configuring Overpayment Processes for Suppression

Business object based processes should be configured to check for suppression before triggering processing that should be placed on hold while suppression applies.

The base package includes sample algorithms for the Overpayment Process maintenance object that demonstrate how this can be achieved.

The base algorithm type *C1-OP-SUPPCH* provides an example of an enter algorithm that will issue an error if suppression applies to the overpayment process's obligation. This algorithm type would typically be configured on any state for the overpayment process business object that would affect the associated obligation's financial balance. Its purpose is to prevent online transitions to the invalid states.

The base algorithm type *C1-OP-SUPPMN* provides an example of a monitor algorithm that will indicate that transition should not take place if suppression applies to the overpayment process's obligation. This algorithm type would typically be configured on any state that is set up to automatically transition to a state for the overpayment process business object that would affect the associated obligation's financial balance. Its purpose is to prevent background transitions to the invalid states.

Refer to *Business Objects* for more information on business object lifecycles and configuration.

Configuring Obligation Types for P&I Suppression

The base functionality that supports penalty and interest calculations is designed to recognize discrete time periods in which differing calculation rules apply. This includes identifying periods where penalty and interest should not accrue. The base package supports functionality to suppress P&I calculation for a given period by providing a sample algorithm that identifies all the suppressed periods that apply to the overall calculation period and turns off all P&I rule processing for those date ranges.

In order to use the base functionality that suppresses P&I, you should configure algorithm *C1-PI-PR-SUP* on all appropriate obligation types. Refer to the algorithm type description for a detailed description of the way in which the periods of suppression are determined.

Refer to *The Big Picture Of Penalty and Interest* for more information on configuring penalty and interest rules.

Configuring Adjustment Types for P&I Suppression

In order to use the base algorithm type that validates whether a particular adjustment type can be created when suppression applies, you must perform the following configuration:

- Create an algorithm for algorithm type *C1-RVW-CCCRR* or use the base algorithm that applies the validation rule to all suppressed process types.
- If you wish to restrict the validation to suppressions of particular process types, create multiple algorithms specifying those process types as their input parameter.
- Navigate to **Admin Menu** > **Adjustment Type** and configure the appropriate algorithm using the **Validate Adjustment** system event for each of the adjustment types that should not be created. Multiple **Validate Adjustment** algorithms may be configured to cover multiple processes for the same adjustment type.
- Refer to *Adjustment Type - Algorithms* for more information.

Setting Up Suppressed Process Types

The types of processes that can be affected by suppression are defined using an extendable lookup. To view or create suppressed process types:

- Open **Admin Menu** > **Extended Lookup**.
- Search for and select the **Suppressed Process Type** extendable lookup business object.
- The list of existing suppressed process types is displayed in a standard *List zone* zone.
- Choose an existing suppressed process type to view, edit, delete or duplicate.
- Use the **Add** link in the zone header to create a new suppressed process type.

Setting Up Suppression Creation Reasons

Suppression creation reasons are defined using an extendable lookup. To view or create suppression creation reasons:

- Open **Admin Menu** > **Extended Lookup**.
- Search for and select the **Suppression Creation Reason** extendable lookup business object.
- The list of existing suppression creation reasons is displayed in a standard *List zone* zone.
- Choose an existing suppression creation reason to view, edit, delete or duplicate.
- Use the **Add** link in the zone header to create a new suppression creation reason.

Alert to Highlight Active Suppressions

The base algorithm type *C1-CCAL-ASE* provides the ability to create an Alert if active suppressions exist for the person or account in context.

To configure the alert, navigate to **Admin Menu** > **Installation Options - Framework** and configure the base algorithm using system event **Alert**.

Chapter 20

Defining Work Management Options

This section describes miscellaneous functionality provided by the product to manage business processes that are not mainstream processes.

- Process flow is a generic maintenance object provided for implementations to build a business process for situations where the base product doesn't include a dedicated maintenance object.
- Entity correction is a maintenance object provided to support creation or updating of a group of object in bulk.

NOTE: This chapter also includes documentation for Case functionality, which is an older object no longer recommended. Process flow was provided to replace case functionality.

Configuring Process Flows

Process Flow is a business object based maintenance object provided to support workflow-type business processes or tasks when existing base product maintenance objects are not well suited for the task. The topics in this section describe the generic Process Flow entity and how it can be customized.

The Big Picture of Process Flows

The topics in this section describe process flow functionality.

Process Flow Is A Generic Entity

The Process Flow maintenance object is a generic entity that can be configured to represent custom entities and support automated workflows for a variety of applications. Each process flow references a business object to describe the type of entity it is. The base package does not provide any pre-configured business objects for Process Flow. Implementations configure their own business objects and related user interface to orchestrate the desired custom business process.

A status column on the process flow may be used to capture its current state in the processing lifecycle controlled by its business object.

The maintenance object also supports a standard characteristic collection as well as a CLOB element to capture additional information.

Process Flow's Business Object Controls Everything

A process flow's business object controls its contents, lifecycle and various other business rules:

- Its schema defines where each piece of information resides on the physical Process Flow maintenance object.
- It may define a lifecycle for all process flow instances of this type to follow. Each process flow must exist in a valid state as per its business object's lifecycle definition.
- It may define validation and other business rules to control the behavior of process flows of this type.

Using Process Flow Type To Control Process Flows

Each process flow must reference a process flow type. The process flow type will reference business objects that define:

- The structure and processing options of the process flow type itself.
- The structure, lifecycle, processing rules and processing options of the actual process flow.

The process flow type may be used to define parameters for the associated process. Examples of such parameters are:

- Account Types, Obligation Types or Tax Types applicable to the process.
- Adjustment Types for any financial transactions that are created as part of the process.
- Customer Contact Classes / Types for any correspondence triggered from the process flow.

Person / Account / Tax Role / Obligation References

Some types of processes may be person-oriented, others may be obligation-oriented or tax role-oriented, and still others may be account-oriented. Any combination of person, account, tax role and obligation is permitted on a process flow. The associated business object may be designed to include only those objects that are relevant to the particular type of process being modeled.

Process Flow Supports A Log

The Process Flow maintenance object supports a log. Any significant event related to a Process Flow may be recorded on its log. The system automatically records a log record when the process flow is created and when it transitions into a new state. In addition, any custom process or manual user activity can add log entries.

Access Rights

You can take advantage of the system's business object security functionality to restrict processes of a given type and states within the process lifecycle to certain user groups.

Additional Documentation

For more information about the various configuration tools available to customize the process flow maintenance object, refer to the Configuration Tools documentation.

Refer to *The Big Picture of Business Objects* for details of how to configure content, lifecycle and business rules. Refer to *State Transitions Are Audited* for more information on business object logs. Refer to *Granting Access To Business Objects* for more information on security options.

For more information about user interfaces, refer to *the Configurable User Interface* documentation.

Setting Up Process Flows

The following sections describe how to prepare process flow types.

Setting Up Process Flow Types

Each process flow requires a process flow type. To set up a process flow type, select **Admin Menu** > **Process Flow Type**.

The topics in this section describe the base-package zones that appear on the Process Flow Type portal.

Process Flow Type List

The Process Flow Type List zone lists every process flow type. The following functions are available.

- Click a broadcast button to open other zones that contain more information about the adjacent process flow type.
- Click the **Edit** button to maintain the process flow type.
- Click the **Duplicate** button to copy the process flow type.
- Click the **Delete** button to delete the process flow type.
- Click the **Add** link in the zone's title bar to add a new process flow type.

Actions

This is a standard actions zone. The **Edit, Delete** and **Duplicate** actions are available.

Process Flow Type

The Process Flow Type zone contains display-only information about a process flow type. This zone appears when a process flow type has been broadcast from the Process Flow Type or if this portal is opened via a drill down from another page.

The UI Map used for the display is determined by the Display UI Map option on the associated business object.

The base package does not provide a business object for Process Flow Type. Each implementation must define its own business objects and user interfaces.

The Big Picture of Entity Correction

Entity correction is an object provided to perform an action on a group of records. The product provides base support for the following business use cases:

- Mass cancellation of tax forms.
- Mass cancellation of payment events.
- Mass cancellation / correction of overdue events for a group of overdue processes.
- Mass creation of suppressions for a group of taxpayers.
- Mass release of suppressions for a group of taxpayers.

- Mass retry for overpayment processes.

Implementations may follow the pattern to introduce additional business use cases. The topics in this section describe entity correction functionality.

How Are Entity Corrections Created?

The product expects that the entity correction is created via a web service. The expectation is that analysis is performed using an appropriate reporting or analysis tool to determine the records that require a special action to be performed. The reporting / analysis tool can then create a file and an appropriate mechanism like Oracle Service Bus transforms the file and interfaces the information via a web service call. The web service is used to create the new entity correction and link the selected entities.

The product does not provide any logic for manually defining which entities to include in the entity correction.

The entity correction type includes configuration to indicate whether the list of IDs provided in a new entity correction represent internal system IDs or rather external references. Not all objects in the system have an external identifier. For example, when creating an entity correction for mass payment cancellation, there is no external identifier for payment events.

Entity Correction Lifecycle

Entity corrections require information unique to the particular instance that is typically not provided by the interface, for example, the reason for the correction. Some types of entity corrections require even more specific details in order to produce the mass action. For example, the mass suppression entity correction type requires a suppression reason and suppressed entity to be defined. The product supports two different entity correction lifecycles based on whether or not the additional information is needed prior to the validation of the entities.

- For use cases where the addition information is not required for validation of the entities, the business object **C1-EntityCorrectionControl** (Entity Correction) is provided.
  - A record is created from an external source and can be validated without any input from a user.
  - Once the record is validated, it is set to **Approval in Progress** at which time a user group is alerted via a To Do entry.

  If a given business use case does not require any additional information besides a correction reason and comments, this business object may be used as the transaction BO. The mass tax form cancellation and mass payment cancellation use cases use this business object. If a given business use case requires additional data where the user can provide the data when approving the record (i.e. it's not needed for validation) then this business object may be extended. The base business object **C1-MassSuppressionForTaxpayers** (Mass Suppression for Taxpayers) extends this business object.
- For use cases where the additional information is required for validation of the entities, the business object **C1-EntityCorrectionExtraInfo** (Entity Correction - Detail Validated) is provided. Once the additional data is provided the user transitions the record to pending to be picked up for validation. Once the record is validated, it is set to Approval in Progress at which time a user / group of users can be alerted via a To Do entry.
  - This business object is not expected to be used for any business use cases because by definition, additional data is expected from a user prior to validating. As such the expectation is that this BO is only used as a lifecycle BO for other business objects.
  - For a business use case that requires additional data prior to validation, this business object is extended. The business objects **C1-OverdueCorrectionControl** (Overdue Correction) and **C1–MassSuppressionRelease** (Mass Suppression Release) extend this business object.

Entity Correction Algorithms

The validation and processing of the entity correction is performed by algorithms linked to the entity correction type. This allows for various business use cases to be implemented without requiring different business objects. A specific business object for a type of entity correction is only required if there are explicit elements that need to be provided for a given use case.

- The validation algorithms are executed for the entity correction "parent" record. Validation algorithms are responsible for checking that the linked entities for the entity correction are valid records in the system and that the mass action is appropriate for each record. Any record that the validation algorithm detects should not be processed should be updated with an appropriate Link Status and for a link status of **Skipped**, an appropriate Skip Reason should be populated. Refer to the Entity Link Status section of *Common Entity Correction Functionality* for more information.
- The correction algorithm is called for each Active entity for the entity correction by the mass update background process. The correction algorithm should reverify any condition checked by the validation algorithm that could have changed since that algorithm was run. For example, if the validation algorithm checked that the status of the entity was valid, the entity's status could have changed while waiting for approval so it should be checked again. If there is any reason that the entity should not be processed, the algorithm should return to the batch job the appropriate Link Status and for a link status of **Skipped**, an appropriate Skip Reason should be populated. The batch job will update the record. Otherwise, the algorithm should perform the appropriate mass action.

NOTE: Refer to the base provided algorithms for examples.

Entity Correction Approval

Once the validation of the entities linked to the entity correction has completed a user must review the entity correction and determine whether to approve or reject. The entity correction type defines the To Do type and To Do role used to alert the appropriate users that a entity correction is ready for review.

Mass Update of Entities

Once an entity correction is approved, the appropriate mass action may be performed on its related entities. The approval step stamps the entity correction with the batch job that is responsible for processing the mass action. The batch job is taken from the entity correction type. The current run number for that batch job is stamped as well. The base product provides a batch job (**C1-ENCOR** - Entity Correction Process) that is responsible for finding entity correction records for the current run number and running the entity correction type's Correct Entity algorithm for each entity with an **Active** link status.

Configuring Entity Corrections

If your implementation supports entity corrections, the following topics highlight configuration requirements.

In addition, your implementation should read the detailed descriptions for the base product business objects provided for Entity Correction Type (**C1-EntityCorrectionType**) and all the base Entity Correction business objects.

For additional detail about the overdue correction functionality, refer also to *Overdue Correction*.

Configuring Correction Reasons

Entity corrections include a correction reason that users can populate to record why the mass correction needs to occur. To view or create entity correction reasons:

- Open **Admin Menu** > **Extended Lookup**.
- Search for and select the **Entity Correction Reason** extended lookup business object.
- The list of existing entity correction reasons are displayed in a standard *List zone*.
- Choose an existing entity correction reason to view, edit, delete or duplicate.
- Use the **Add** link in the zone header to create a new entity correction reason.

Configuring Skipped Reasons

When a linked entity is a valid record in the system, but for some reason is not in an appropriate situation for the requested action, the validation and correction algorithms should mark the entity's link status as **Skipped** along with a Skip Reason so that users may understand the reason the record was skipped. The product provides skipped reasons that may be used. Implementers may also choose to add additional values. To view or create skipped reasons:

- Open **Admin Menu** > **Extended Lookup**.
- Search for and select the **Entity Skipped Reason** extended lookup business object.
- The list of existing skipped reasons are displayed in a standard *List zone*.
- Choose an existing skipped reason to view, edit, delete or duplicate.
- Use the **Add** link in the zone header to create a new entity skipped reason.

Configuring Entity Correction Algorithms

The system provides two plug-in spots on the entity correction type to perform logic that is specific to an entity correction use case.

- The validation plug-in is executed when an entity correction of this type transitions from pending to validated. It is responsible for ensuring that appropriate internal entity IDs are determined and valid for each entity linked to the entity correction.

  NOTE: Base plug-ins. Click *here* to see the algorithms types available for this system event.

- The correct entity plug-in is executed for each entity linked to an approved entity correction of this type and is responsible for correcting each entity as per business rules.

  NOTE: Base plug-ins. Click *here* to see the algorithms types available for this system event.

Setting Up Entity Correction Types

An Entity Correction Type defines the configuration information that is common to entity corrections of a given type. The type of information captured on the entity correction type is governed by the entity correction type's business object.

To set up an Entity Correction Type, select **Admin Menu** > **Entity Correction Type**.

The topics in this section describe the base-package zones that appear on the Entity Correction Type portal.

Entity Correction Type List

The following functions are available:

- Click a *broadcast* button to open other zones that contain more information about the adjacent entity correction type.
- Click the **Add** link in the zone's title bar to add a new entity correction type.

Entity Correction Type

The Entity Correction Type zone contains display-only information about an Entity Correction Type. This zone appears when an Entity Correction Type has been broadcast from the Entity Correction Type List zone or if this portal is opened via a drill down from another page.

The information displayed in the zone is dictated by the entity correction type's business object . However, the expectation is that among other information, the Correct Entity and Validation algorithms are defined by every business object.

Please see the zone's help text for information about this zone's fields.

Where Used

Follow this link to open the data dictionary where you can view the tables that reference *C1_ENTITY_CORR_TYPE*.

Entity Correction Batch Jobs

The following batch jobs must be run to progress an entity correction record. Because the mass actions performed by an entity correction may be unusual and rare, implementations may choose not to schedule these jobs on a regular basis, but rather rely on communication between the business users and the technical support staff to run these batch jobs on an "on demand" basis.

- **C1-ENCTD** (Entity Correction Monitor (Deferred)). This batch job selects entity correction records that are **Pending** and validates that the record by calling the validation algorithms on the entity correction type. Once this batch job completes, the processed entity corrections are ready for approval.

  NOTE: Email routing. If your implementation plans to use email routing for the "entity correction requires approval" To Do entry, then the background process **F1-TDERR** (To Do External Routing) must be scheduled after the monitor batch job with the same or similar frequency.

- **C1-ENCOR** (Entity Correction Process). This batch job selects entity corrections that are stamped with this batch code and the current run number and calls the correct entity algorithm on the entity correction type.

Implementing Web Service Options

The product provides the following configuration to support interfacing a list of entities to link to an entity correction via a web service.

The XAI Inbound Service **C1-EntityCorrectionUpload** provides the API for the service script **C1-EnCrrUpld**. This service script includes logic creating an entity correction record and linking a group of entity IDs. Note that the script has been designed to be called iteratively for cases where a large file is received and the interface tool that processes the file can process it in chunks.

FASTPATH: Refer to the description of the XAI inbound service and the above service script for more information.

Defining Case Options

The Case functionality was originally provided as a highly configurable tool for managing business processes in situations where the base product didn't include a dedicated maintenance object. This functionality has now been superseded by either specific functionality (e.g. Appeals, Bankruptcy), or by the use of *Process Flow*.

The topics in this section describe how to configure the system to manage cases for those implementations that are continuing to use the legacy case functionality.

FASTPATH:
Refer to *Case Management* for a description of how end-users use cases.

The Big Picture Of Cases

The topics in this section provide background information about how to configure cases.

Case Type Controls Everything

Whenever a user creates a case, they must specify the type of case. 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.

Person / Account / Location Applicability

Some types of cases may be person-oriented, others may be location-oriented, and still others may be account-oriented. When you set up a case type, you define if its cases must reference a person, account, and/or location. Note, any combination of these objects is permitted on a case.

Contact Information Applicability

When a case is created as a result of contact from an outside party, 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*). 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.

NOTE:
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. 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 lifecycle whose rules are governed by the base-package and cannot be changed. For example, an obligation starts out in the **Pending Start** state and eventually becomes **Closed** when it's been completely paid. You can't change the system to allow an obligation to starts its life in the **Closed** state.

The lifecycle of cases is not governed by the base product. Rather, you define the lifecycle of your cases when you set up their case types.

The topics in this section describe important lifecycle concepts.

Valid States versus State Transition Rules

A given case can have one or more potential valid states. State transition rules govern the states a case can move to while it's in a given 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. This means that a user will never see the case in this state. If the other states were marked as **non-transitory**, and an error were to occur during the transition from a transitory state, the case would roll back any changes to data made in the entered state (Enter Processing) along with the changes made in the transitory state, and would end up in the last non-transitory state prior to the transitory state.

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.

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.

The "final" states are used by the system to differentiate between open and closed cases. For example, an alert highlights when the person / account / location 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).

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.

NOTE:
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 Allow A User 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. The case will transition to the appropriate state depending the button the user presses.

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*.

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 topics in this section describe business rules that can be configured for each 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. 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.
- 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.

FASTPATH:
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 case must reference a cancel reason before it enters a **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 **Assigned** 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 *To Do entry* created when a case enters a given state. This additional processing is held in algorithms that are plugged in on the case type.

You can also incorporate state transitioning 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. 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 collection 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 case leaves a certain 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 using auto transition rules. For example, if you have a case that sends a letter to a taxpayer, it can be configured to transition to the **Follow Up** state 1 week after the letter is sent. 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 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** 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.

NOTE:
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.

CAUTION:
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. 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

NOTE:
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 formal appeal.

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 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 formal appeal that requires investigation. You can set up the system to execute a specific script when a user selects this To Do 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 functionality with *To Do entries*.

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. To implement this, you can set up the case type as follows:

- Plug-in an *entry processing* algorithm on the status where you wish a To Do entry to be created or completed.
- Plug-in an *exit processing* algorithm on the state where you wish the To Do entry to be created or completed.

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 formal appeal 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).

Setting Up Case Options

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

CAUTION:
The topics in this section assume you thoroughly understand the concepts described under *The Big Picture Of Cases*.

Installation Options

Case Info May Be Formatted By An Algorithm

The case information displayed throughout the system is controlled by a plug-in.

The system first looks to see if the case type references a case business object and if the business object defines an information plug-in. If a BO is not provided or if that BO does not define an information algorithm, the system looks for an information algorithm plugged into the case maintenance object.

If no plug-ins are found on the BO or the MO, the system looks for a plug-in algorithm on the *Case Type*.

If such an algorithm is not plugged-in on the Case Type, the system looks for a corresponding algorithm on the *installation record*.

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 / location 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.

FASTPATH:
Refer to *The Big Picture Of Cases* for more information about how a case type encapsulates the business rules that govern a case.

Case Type - Main

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

Open the case type page by selecting **Admin Menu** > **Case Type**.

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.

Location Usage controls the applicability of a location on cases of this type. Select **Required** if a location must be defined on this type of case. Select **Optional** if a location can optionally be defined on this type of case. Select **Not Allowed** if a location 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*
Case Information	Optional	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" may be constructed using an algorithm plugged in here. Refer to *Case Info May Be Formatted By An Algorithm* for more information on when this algorithm is used. Click *here* to see the algorithm types available for this system event.

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 lifecycle. 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 **Admin Menu** > **Case Type** and navigate to 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 taxpayers 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*
Auto Transition	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.
Enter Processing	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.
Enter Validation	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.
Exit Processing	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.
Exit Validation	This algorithm holds validation logic that executes before a case can by 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.
Script Launching	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.

Chapter 21

Background Processes Addendum

This chapter is an addendum to the general *Defining Background Processes* chapter. This addendum describes the background processes that are provided with Oracle Enterprise Taxation and Policy Management.

The System Background Processes

The system provides base background processes that are an important component of the system. Use the batch control page to view the detailed descriptions of each batch control. In addition, the list of background processes provided in the base product may be viewed in the *application viewer*'s *batch control* viewer.

NOTE: Regenerate application viewer. If your implementation adds batch control records, you may *regenerate* the application viewer to see your additions reflected there.

The base processes fall into one of the following categories:

- **Process What's Ready**. These are background processes that create and update records that are "ready for processing". The definition of "ready" differs for every process. For example,
  - The payment upload process creates payments for every record that is pending
  - The overdue event monitor activates pending overdue events that have reached their trigger date

  Some processes of this type use a business date in their determination of what is ready. If the requester of the process does not supply a specific business date, the system assumes that the current system date should be used. If you need to use a date other than the current date, simply supply the desired date when you request the batch process.
- **Monitor Processes**. This is a subset of the "process what's ready" category. A periodic monitor batch process is provided for any maintenance object whose business object defines a *lifecycle*. In addition deferred monitor batch process is provided if a business object supplied in the base product required a deferred process for one of its states. Note that monitor processes can be an important part of your implementation. For example, processing forms requires executing the monitor processes for form batch header, form upload staging, tax form and registration form.
- **Extract Processes**. These processes extract information that is interfaced out of the system. Processes of this type typically extract records marked with a given run number. If the requester of the process does not supply a specific run

number, the system assumes that the latest run number should be extracted. If you need to re-extract an historical batch, you can. Simply supply the respective run number when you request the batch process.

- **Adhoc Processes**. These are background processes that are run on an ad hoc basis based on business needs.
- **To Do Entry Processes**. These are background processes whose main purpose is to generate To Do Entry records based on a certain condition. Refer to *To Do Entries Created By Background Processes* for the details.
- **Purge Processes**. There are a small number of background processes provided to purge historical records from certain objects that generate a large number of entries and may become unwieldy over time.
- **Object Validation Processes**. These background processes are used to validate the master data objects. These programs are typically only run as part of the conversion and upgrade processes.

  NOTE: Another use for these programs. In addition to validating your objects after conversion or an upgrade, the validation programs listed below have another use. For example, you want to experiment with changing the validation of a person and you want to determine the impact of this new validation on your existing persons. You could change the validation and then run the person validation object - it will produce errors for each person that fails the new validation.

  FASTPATH: Refer to *Validate Information In The Staging Tables* for more information about these processes and where their errors appear.

- **Referential Integrity Validation Processes**. These background processes check the validity of foreign keys on various objects. These programs are typically run as part of the conversion and upgrade processes.

  FASTPATH: Refer to *Validate Information In The Staging Tables* for more information about these processes and where their errors appear.

- **Conversion Processes**. There are a series of background processes provided for loading historical data into your production database.

  FASTPATH: Refer to *The Conversion Process* for more information about these processes.

Refer to *Batch Process Dependencies* for guidelines on setting up your production schedule for the base background processes.

Batch Process Dependencies

The topics in this section illustrate the periodicity and dependencies between the various background processes described above.

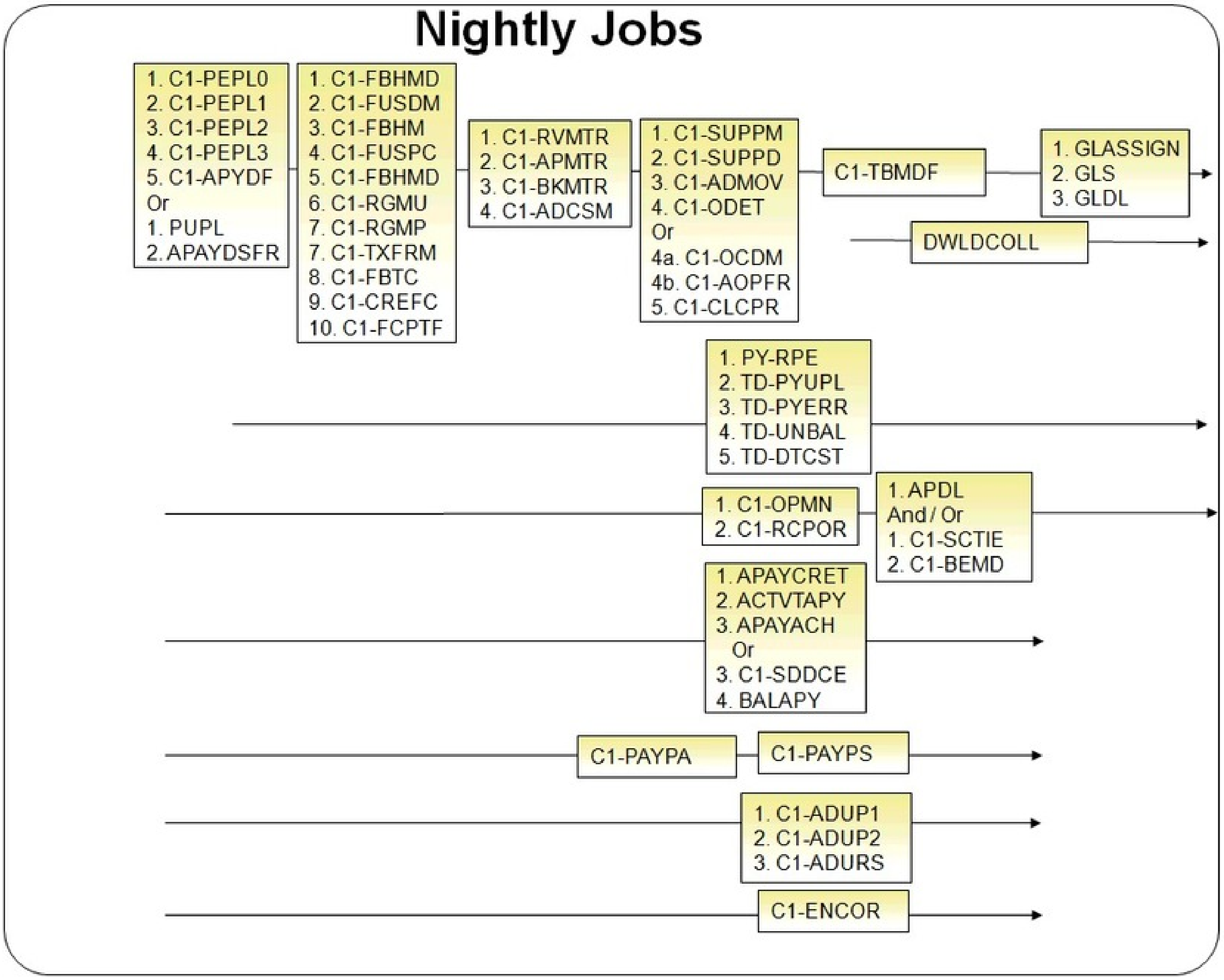
Batch Schedulers and Return Codes

If you use a batch scheduler (e.g., Control-M, Tivoli) to control the execution of your batch processes, it will be interested in the possible values of each process's return code. The return code is a number that indicates if the process ended successfully. All product processes will return one of the following return code values:

- **0** (zero). A value of zero means the batch process ended normally.
- **2**. A value of 2 means the batch process detected a fatal error and aborted.

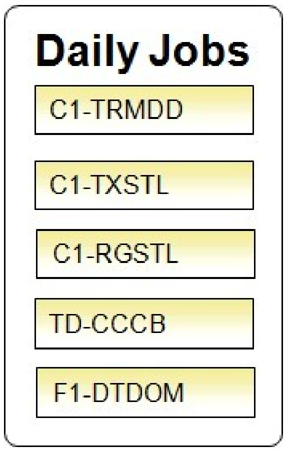
The Nightly Processes

The following diagram illustrates the dependencies between the batch processes.



The mnemonics in the boxes refer to the batch control code. Use the application viewer or the Batch Control page in the application to review the details of each batch. When a box contains multiple processes, these processes must be run sequentially. When multiple boxes exist on a timeline, all processes in an earlier box must execute before the subsequent box is executed. Those timelines that appear beneath the first job stream's timeline indicate when the timeline's respective processes can be executed in respect of the first job stream.

The following diagram illustrates the daily batch processes for which there are no dependencies.

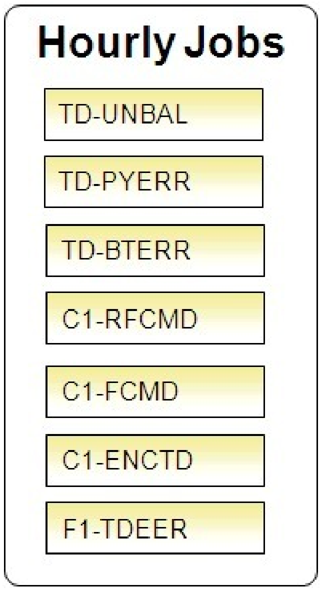


The mnemonics in the boxes refer to the batch control code.

NOTE:
No dependencies exist. As you can see, there are no dependencies between the boxes (meaning they may be run in parallel).

The Hourly Processes

The following diagram illustrates the dependencies between the hourly batch processes.



The mnemonics in the boxes refer to the batch control code. Use the application viewer or the Batch Control page in the application to review the details of each batch.

NOTE: No dependencies exist. As you can see, there are no dependencies between the boxes (meaning they may be run in parallel).

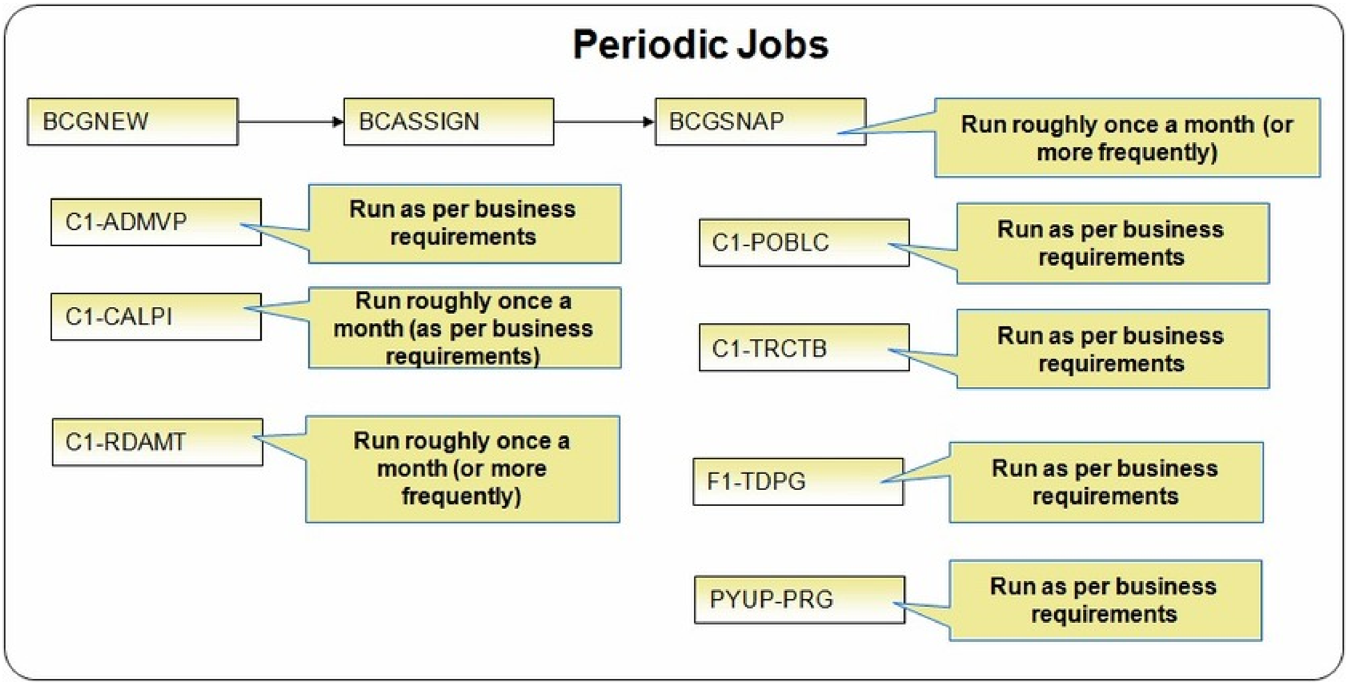
The Letter Processes

To extract information for your various letters, only one background process, **LTRPRT**, is required regardless of the different types of letters you have. This process simply calls an algorithm plugged-in on the respective letter template to construct its flat-file content.

While this process should be run at least on a daily basis, you may want to consider running it more frequently (depending on how frequently you produce letters).

The Periodic Processes

The following diagram illustrates the dependencies between the periodic background processes. While many of these processes should be run at least on a monthly basis, you may want to consider running them more frequently (depending on business requirements).


Periodic Jobs
BCGNEW
BCASSIGN
BCGSNAP
Run roughly once a month (or more frequently)
C1-ADMVP
Run as per business requirements
C1-CALPI
Run roughly once a month (as per business requirements)
C1-RDAMT
Run roughly once a month (or more frequently)
C1-POBLC
Run as per business requirements
C1-TRCTB
Run as per business requirements
F1-TDPG
Run as per business requirements
PYUP-PRG
Run as per business requirements

The mnemonics in the boxes refer to the batch control code. Use the application viewer or the Batch Control page in the application to review the details of each batch.

NOTE: Few dependencies exist. As you can see, there are few dependencies between the boxes (meaning they may be run in parallel).

The Big Picture of Sample & Submit

Sample and Submit refers to the ability to create Activity Requests. This is functionality that enables an implementer to design an ad-hoc batch process using the configuration tools.

Some examples of such processes are:

- Send a letter to customers that use credit cards for auto pay and the credit card expiration date is within 30 days of the current date.
- Stop auto pay for customers that use credit cards as the form of payment if the credit card has already expired. Notify the customer that their auto pay agreement has been terminated and that they need to call to reinstate.
- Select auto pay accounts that have more than X non-sufficient fund penalties, stop the auto pay agreement and notify the customer.

NOTE: The terms activity request and sample & submit request may be used interchangeably.

Activity Type Defines Parameters

For each type of process that your implementation wants to implement, you must configure an activity type to capture the appropriate parameters needed by the activity request.

Preview A Sample Prior To Submitting

To submit a new activity request, a user must select the appropriate activity type and enter the desired parameter values, if applicable.

After entering the parameters, the following actions are possible

- Click **Preview** to see a sample of records that satisfy the selection criteria for this request. This information is displayed in a separate map. In addition, the map displays the total number of records that will be processed when the request is submitted. From this map you can **Save** to submit the request, go **Back** to adjust the parameters or **Cancel** the request.
- Click **Cancel** to cancel the request.
- Click **Save** to skip the preview step and submit the request.

When an activity request is saved, the job is not immediately submitted for real time processing. The record is save in the status **Pending** and a monitor process for this record's business object is responsible for transitioning the record to **Complete**.

As long as the record is still **Pending**, it may be edited to adjust the parameters. The preview logic described above may be repeated when editing a record.

The actual work of the activity request, such as generating customer contact records to send letters to a set of customers, is performed when transitioning to **Complete** (using an enter processing algorithm for the business object).

Credit Card Expiration Notice

The base product supplies a sample process to find customers that use credit cards for auto pay and the credit card expiration date is within x days of the current date.

To this functionality the following configuration tasks are needed:

- Define an appropriate *customer contact class* and *type* to use.
- Define appropriate activity request Cancellation Reasons. Cancellation reasons are defined using a customizable *lookup*. The lookup field name is **C1_AM_CANCEL_RSN_FLG**.
- Define an activity type for the business object **C1-NotifyExpiringCreditCardTyp**. You may define default parameter values for the number of days for expiration and customer contact class and type.

Exploring Activity Request Data Relationships

Use the following links to open the application viewer where you can explore the physical tables and data relationships behind the activity request functionality:

- Click *C1-ACM-ACTTY* to view the activity type maintenance object's tables.
- Click *C1-ACM-ACTRQ* to view the activity request maintenance object's tables.

Defining a New Activity Request

To design a new ad-hoc batch job that users can submit via Sample and Submit, first create a new Activity Type business object. The base product BO for activity type **C1-NotifyExpiringCreditCardTyp** may be used as a sample.

The business object for the activity request includes the functionality for selecting the records to process, display a preview map for the user to review and to perform the actual processing. The base product BO for activity request **C1-NotifyExpiringCreditCardReq** may be used as a sample. The following points highlight the important configuration for this business object:

- Special BO options are available for activity request BOs to support the *Preview Sample* functionality.
  - Activity Request Preview Service Script. This script is responsible for retrieving the information displayed when a user asks for a preview of a sample of records.
  - Activity Request Preview Map. This is the map that is invoked to display the preview sample results.
- The enter algorithm plugged into the **Complete** state is responsible for selecting all the records that satisfy the criteria and processing the records accordingly.

Setting Up Activity Types

Activity types define the parameters to capture when submitting an activity request via Sample and Submit. To set up an activity type, open **Admin Menu** > **Activity Type**.

The topics in this section describe the base-package zones that appear on the Activity Type portal.

Activity Type List

The Activity Type *List zone* lists every activity type. The following functions are available:

- Click a *broadcast* button to open other zones that contain more information about the adjacent activity type.
- The standard actions of **Edit, Duplicate** and **Delete** are available for each activity type.
- State transition buttons are available to transition the activity type to an appropriate next state.

Click the **Add** link in the zone's title bar to add a new activity type.

Activity Type

The Activity Type zone contains display-only information about an activity type. This zone appears when an activity type has been broadcast from the Activity Type List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Maintaining Sample & Submit Requests

Use the Sample and Submit transaction to view and maintain pending or historic activity requests. Navigate using **Main Menu** > **Batch** > **Sample & Submit Request**.

Sample & Submit Request Query

Use the *query portal* to search for an existing sample & submit request. Once a request is selected, you are brought to the maintenance portal to view and maintain the selected record.

Sample & Submit Request Portal

This portal appears when a sample & submit request has been selected from the Sample & Submit Request Query portal.

The topics in this section describe the base-package zones that appear on this portal.

Sample & Submit Actions

This is a standard actions zone. Use the **Edit** button to modify the parameters. Refer to *Preview A Sample Prior to Submitting* for more information.

If the activity request is in a state that has valid next states, buttons to transition to each appropriate next state are displayed.

Sample & Submit Request

The Sample & Submit zone contains display-only information about an activity (sample & submit) request.

Please see the zone's help text for information about this zone's fields.

Sample & Submit Request Log

This is a standard *log zone*.

Chapter 22

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 Public Sector Revenue Management.

Implementing Account Security

CAUTION: **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 taxpayers 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 businesses and individual taxpayers, you may need to restrict access to these different taxpayer segments based on the skill set of the users. For example, some users are skilled in dealing with business taxpayers, while others are skilled in dealing with individual taxpayers.

By granting a user access rights to an account, you are actually granting the user access rights to the account's bills, payment, adjustments, etc.

FASTPATH:
Refer to *If You Do Not Practice Account Security* for setup instructions if your organization doesn't practice account security.

NOTE: **Account security may also affect persons and locations.** Refer to *Persons Can Also Be Secured* for how access to person information is also restricted by account security. Refer to *Locations Can Also Be Secured* for how access to location information is also restricted by account security.

The topics in this section describe how to implement 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.

Locations Can Also Be Secured

It's important to be aware that locations can also be secured as a result of "account security". It works like this:

- If a location is linked to at least one account, users will not be allowed access to the location (or the location's related information) unless they have access to at least one of the location's accounts.
- If a location is not linked to an account (a rare situation), then all users may access the location.

NOTE: This is only applicable to implementations that continue to use *legacy address* functionality.

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 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 obligations associated with accounts to which they have access rights.
- A user can only see financial transactions associated with obligations that are, in turn, associated with accounts to which they have access rights.

NOTE:
Person and location searches are also impacted. Keep in mind that information will be suppress from both person and location-oriented searches if the person / location 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 *Locations Can Also Be Secured* for how access to location 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 payment in error to a user who doesn't have access to the payment's account. This user will then see the related To Do entry in their Payment 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 payment segment that's in error). This happens because the drill down causes the payment segment search logic to execute. This logic inhibits the selection of payment 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.
- **Location-oriented.** This notation is used if the respective transaction uses location-oriented account security. Refer to *Locations 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***
360 Degree Search	Account-oriented and Person-oriented
Account	Account-oriented
Account Bill / Payment History	Account-oriented
Account Financial History	Account-oriented
Account Payment History	Account-oriented
Account Person Replicator	Account-oriented
Adjustment	Account-oriented
Adjustment Calculation Line Characteristics	Account-oriented
Bill	Account-oriented
Bill Segment	Account-oriented
Case	Account-oriented, Person-oriented and Location-oriented
Collection Agency Referral	Account-oriented
Customer Contact	Person-oriented
Financial Transaction	Account-oriented
Financial Transaction on a Payment	Account-oriented
Match Event	Account-oriented
Overdue Process	Account-oriented
Pay Plan	Account-oriented
Payment	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
Location	Location-oriented
Obligation Billing History	Account-oriented

Transaction	***Type of Account Security***
Obligation Cash Accounting Balance	Account-oriented
Obligation Financial History	Account-oriented
Obligation	Account-oriented
Tax Role	Account-oriented

Account Security Case Study

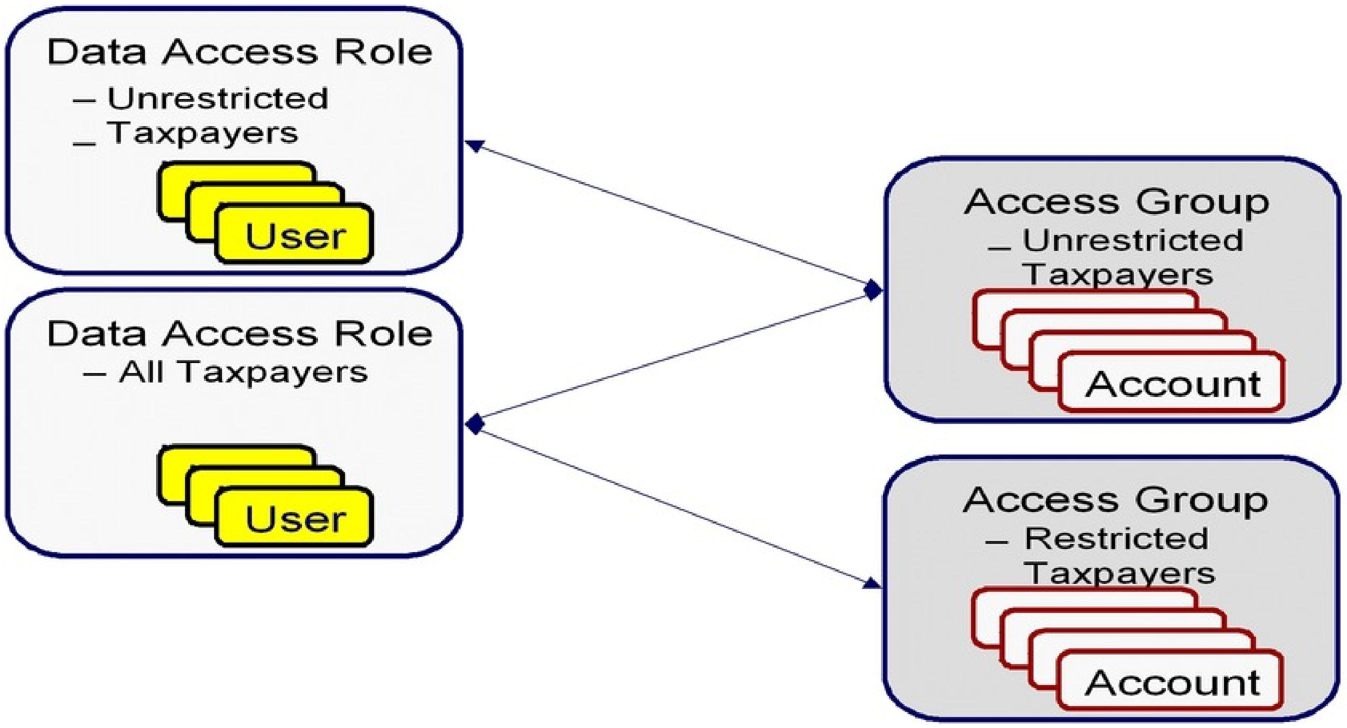
This section contains an example of how to implement account security. Use this example to form an intuitive understanding of these objects. Once this intuition is obtained, you'll be ready to design the account security objects for your own company.

Securing Accounts Based On Account Type

Assume the following security requirement exists:

- You have two broad groups of accounts:
  - Unrestricted Taxpayer accounts for the general public.
  - Restricted Taxpayer accounts for individuals whose tax information is highly sensitive (politicians, celebrities, employees of the tax authority, etc.).
- Users can be classified as have one of the following access rights:
  - May access all accounts.
  - May only access the Unrestricted Taxpayer accounts.

The following diagram illustrates the access groups and data access roles required to implement these requirements:


Data Access Role – Unrestricted Taxpayers
User
Data Access Role – All Taxpayers
User
Access Group – Unrestricted Taxpayers
Account
Access Group – Restricted Taxpayers
Account

Notice the following about the above:

- There are two access groups because access to accounts is based on whether the taxpayer's account is unrestricted or restricted.
- The Unrestricted Taxpayers data access role is only linked to the Unrestricted Taxpayers access group.
- The All Taxpayers data access role is linked to both the Unrestricted Taxpayers and Restricted Taxpayers access groups. Users with this role can therefore access all accounts.

The Default Access Group

The base package defaults an account's access group based on the user who adds the account. It uses the user's *default access group* to do this.

CAUTION:
Please be aware that the user who adds an account must have access to this access group.

NOTE:
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 taxpayer's social security number or bank account number. If your implementation intends to mask any of the information that appears in the *Taxpayer Information Zone*, please navigate to this zone's documentation for special instructions.

Inquiry Audit

There are times when an organization needs to capture audit information when a user views certain information. Examples of these scenarios include:

- A user is under investigation or disciplinary action, and supervisors need to find out what information the user has been looking at.
- Although VIP accounts may be protected from users by account security, supervisors and fraud investigators may still want to know who has been trying to look for forms or other records with a given criteria.
- Sometimes information about a taxpayer is leaked to the newspaper. For example "Town Councilman Smith didn't pay his taxes on time for the last n years". Supervisors want to know who has recently viewed Councilman Smith's tax returns.

The audit information captured includes:

- What - the specific record that was viewed.

- Who - the user viewing the record.
- Where the user performed the inquiry - which portal, zone or page was used.
- How - which search criteria the user entered for a specific query.

For more on the information provided for the audited records, see *Inquiry Audit Query*.

As part of implementing inquiry audit, you will need to decide which portals, zones and pages require inquiry audit. The following section describes how to configure inquiry audit based on how the page or portal is implemented.

Setting Up Inquiry Audit

The topics in this section describe how to set up inquiry audit for the various types of queries.

Setting Up Audit for Portals and Zones

The zone parameter **Audit Service Script** controls if information about the record being inquired upon will be captured for audit. In order to turn on inquiry audit for a zone, you will need to update the **Override Parameter Value** with the following:

- The name of the service script responsible for capturing the audit data and storing it on the Inquiry Audit table.
  - The base includes the service script **C1-ZoneAudit**
  - Additional scripts can be developed if your implementation requires something more specific such as add audit records only if VIP records are viewed or where certain criteria is used.
  - The mnemonic **ss=** is used to define the name of the service script. For example ss='C1-ZoneAudit'.
- Define the input elements to the service script by using the mnemonic **input=**. These can include the user id, zone, portal, user filter values, hidden filter values, any literal value, any portal or global context field.

NOTE: If a zone is called as a pop-up search, the portal is not available to provide to the script. The portal is available to provide to the script when the zone is linked directly to the portal when it is invoked.

See the zone's help on more information.

Setting Up Audit for a Search Page

A **user exit** is needed to turn on inquiry audit for an old style search page or control central. You will need to do the following:

- Determine the service name of the page.
- Create a search page extension to capture the audit information. . The following extension codes are provided as samples:
  - **ext_ multiSearchNameData.jsp** - Control Central Search by Name
  - **ext_ multiSearchAddressData.jsp** - Control Central Search by Address
  - **ext_ accountSearchData.jsp** - Account Search
- The search pages have grid elements that display the result rows. To capture the search criteria, the service script **C1-PageAudit** should be called upon load of the grid element, to add the inquiry audit record.
- The business object to use when calling the service script is **C1-PageAuditInquiry**

NOTE: Control central is not longer recommended for use because of its limitation in the ability to be extended. 360 Degree Search is the product recommended 'central' searching portal. 360 Degree search supports defining inquiry audit using the override zone parameter.

Setting Up Audit for a Maintenance Page

A **user exit** is needed to turn on inquiry audit for an old style maintenance page. You will need to do the following:

- Determine the service name of the page.
- Create a page maintenance page extension to capture the audit information. . The following extension code is provided as a sample:
  - **CMAccountMaintenanceExtension.java** - Account Page Maintenance
- The page maintenance extension method **afterRead** should be overridden to call the service script **C1-PageAudit** to add the inquiry audit record.
- The business object to use when calling the service script is **C1-PageAuditInquiry**

Inquiry Audit Restrictions

Certain portals are designed to include a zone with a list of records and buttons that allow quick editing of a record. This technique invokes a BPA script that displays the data in a UI map, rather than broadcasting a value from a query or info zone. This bypasses the ability to capture the audit for this record. There is currently no ability for an implementation to include a user exit in a BPA script to allow storage of an audit record.

The above applies to any administrative portal that uses the "all in one" portal design that includes a list zone showing all existing records. The list zone typically includes edit, duplicate and delete buttons.

Note For customized portals that call a customized BPA script, an implementation has the ability to include a step to store an Inquiry Audit record.

Inquiry Audit Query

Use the *query portal* to search for the inquiry audit records.

The inquiry audit search allows you to search for inquiry records using various criteria such as:

- Date that the inquiry occurred.
- Portal and/or Zone - you can search on inquiries made in a specific portal or a specific zone.
- Service name (used for page inquiries)
- User or User Group - you can search on inquiries made by a specific user or see inquiries made by a group of users.
- The object inquired on, e.g. Account.
- Specific field inquired on, e.g. Name
- Specific field value used in the inquiry. For example, you can look at inquiries made for a specific name, a certain amount, a specific address.
- A combination of the above criteria.

Once a inquiry audit record is selected from the list, the information displayed depends on the type of inquiry that was performed and may differ based on the business object used for capture. Refer to the zone's help text for more information.

Chapter 23

Self Service Integration

Oracle Public Sector Revenue Management ("the product") provides integration with Oracle Public Sector Revenue Management Self Service ("the self service product"). The topics in this section describe the functionality provided in the product to support the integration and outlines the configuration required to support the self service functionality.

The Big Picture

This section describes high level topics related to the integration.

Message Processing Overview

Public Sector Revenue Management interacts with the self service product via web services. An appropriate inbound web service is invoked from the self service product via BPEL. The inbound web service is configured with a processing service script that knows how to process the request. Requests fall into one of the following categories:

- A taxpayer request. These types of requests fall into one or more sub-categories:
  - A taxpayer inquiry. If the taxpayer requests information like to find out the status of a refund, the inbound web service references a service script that performs the appropriate logic. Most of the time the service script associated with the inbound service can perform the logic needed to retrieve the information to present to the taxpayer. However, there may be cases where the service script creates a real time Service Task to retrieve the requested information and return it.

    FASTPATH: Refer to the *Refund Status* for an example of a request for information that results in the creation of a service task.

  - A request to perform some transaction processing. For example, the taxpayer may submit a payment via self service. In this case, the service script associated with the inbound web service creates a Service Task record that will subsequently perform the necessary validation and processing. The taxpayer does not wait for the service task to complete its lifecycle.

- An internal web service request submitted by the self service product to get information. For example, before submitting a payment the self service product may require the taxpayer to provide proof of identity to verify that the taxpayer is known to the system. The taxpayer identification information is sent to this product via a web service and is processed real time.

Naming convention: The product's inbound web services require that the service name starts with 'TS%'. Specific inbound web services are mentioned throughout the supported functionality.

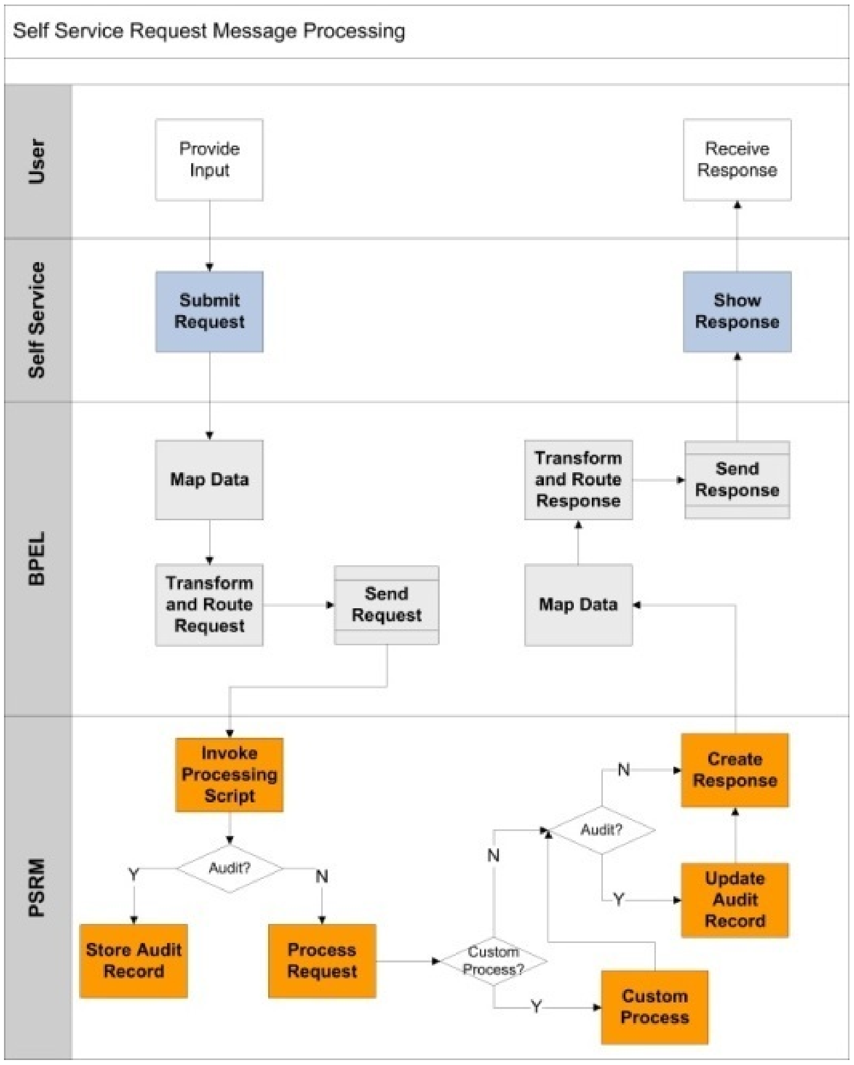


Figure 1: Message Processing

User Identity Verification

The self-service product allows the taxpayer to register as a website user, and enroll to manage taxes online.

A person may work with the self-service portal application in two modes:

- **Casual User.** There is no need to sign in when casually viewing website content or seeking tax-related advice. Certain online services and features are available for casual users.
- **Enrolled User.** A user must log in and be properly authenticated and authorized in order to access sensitive personal and financial information such as historical tax returns, payment history, and other items.

XML request contents are affected as follows:

- Requests initiated by a casual user contain no user or tax account information.
- Requests initiated by an enrolled user contain tax account identifiers.

NOTE: See the *Tax Account Access Information* topic for a detailed explanation of tax account access.

Some online services include an optional taxpayer identification step:

- For casual users, the identification is performed if it's required by business rules.
- For signed-in users, the identification is not performed.

Tax Account Access Information

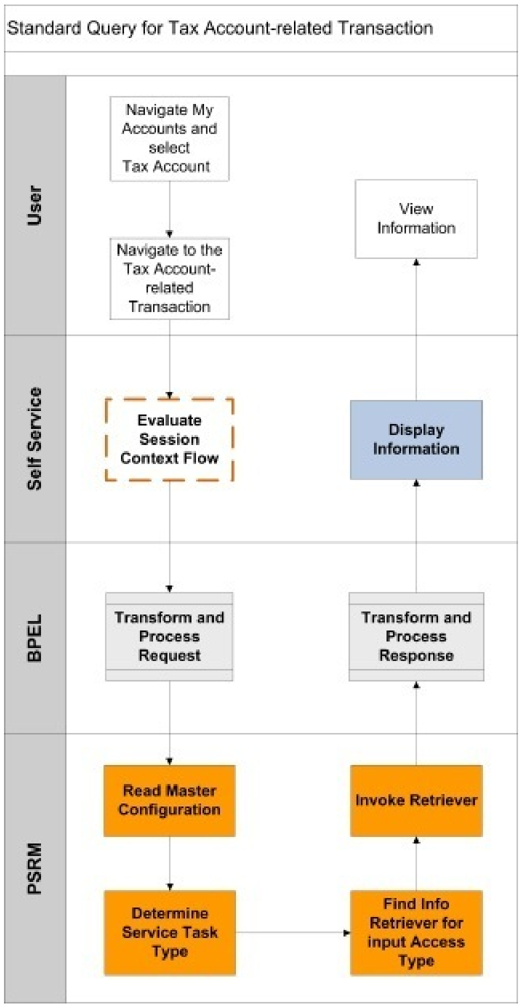
The revenue management authority grants s user access to his/her taxes. The single access “unit” could be a tax role, an account, a tax type, or other entity.

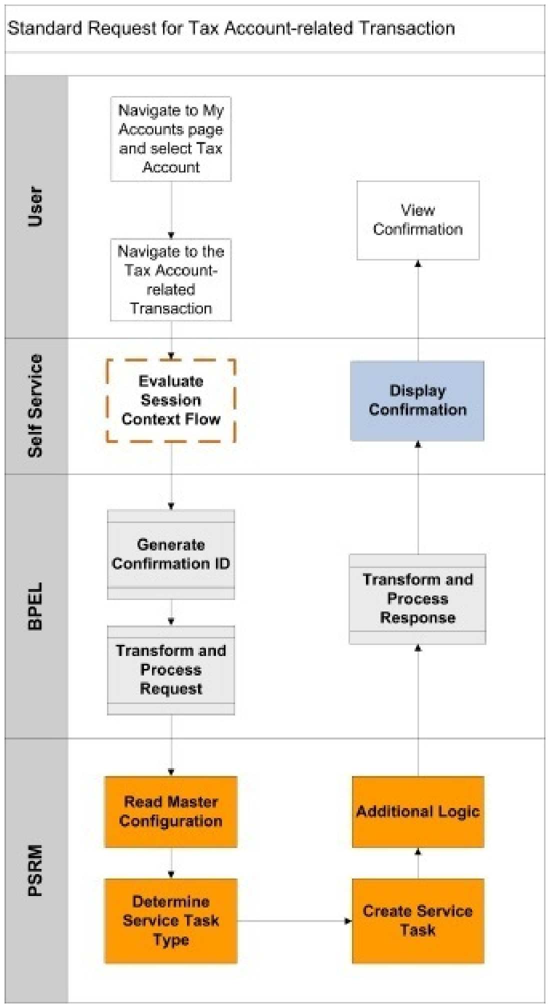
Users can be granted access for many different purposes, including management of a user's tax role, account management, or managing an account for a limited period of time.

When a user is registered and enrolled into online services, the enrollment summary ("tax accounts") is displayed on the user's **My Accounts** portal page. The summary can include tax role(s), accounts, obligations, and other entities.

The request message captures the identifiers of a tax account that is currently being viewed by the self service portal user.

The following diagrams illustrate the process flow for tax account-related inquiry and service requests.


Standard Query for Tax Account-related Transaction
User
Self Service
BPEL
PSRM
Navigate My Accounts and select Tax Account
Navigate to the Tax Account-related Transaction
View Information
Evaluate Session Context Flow
Display Information
Transform and Process Request
Transform and Process Response
Read Master Configuration
Invoke Retriever
Determine Service Task Type
Find Info Retriever for input Access Type


Standard Request for Tax Account-related Transaction
User
Self Service
BPEL
PSRM
Navigate to My Accounts page and select Tax Account
Navigate to the Tax Account-related Transaction
View Confirmation
Evaluate Session Context Flow
Display Confirmation
Generate Confirmation ID
Transform and Process Request
Transform and Process Response
Read Master Configuration
Determine Service Task Type
Create Service Task
Additional Logic

Access Type

Access type describes the scope of a single tax account managed via the self service portal. An example of access type is **Tax Role**.

The access type is referenced in various self service-related configurations; when the request is processed the system uses the access type to determine the appropriate logic to execute. For example, an outstanding balance for a tax role is retrieved in a different way than the outstanding balance for an account.

The <access type> element is included in every message. Supported access types are represented in the product by the lookup **USER_ACCESS_TYPE_FLG**.

Access Keys

The tax account is identified by the access type and a set of keys. The integration supports an ability to identify the account with up to 10 keys.

The first key is always reserved for the taxpayer ID associated with the tax account.

Line of Business

In the web self-service application, Line of Business is used to broadly group areas of interest. The web pages found within a defined line of business are related to the information and tasks appropriate for that area of interest. Some examples of lines of business for an implementation may include Individual taxpayers, Business taxpayers and Tax Preparers. Another implementation may use even more granularity, to include, for example, charitable or non-profit organizations.

In terms of the product, Line of Business can be viewed as a broad category of taxes managed by the tax authority, but also as a way to define a target audience spanning several taxpayer types. Line of business is referenced in self service-related configurations and allows processes to invoke specific logic when needed. For example, different rules may apply when the system identifies tax accounts owned by individuals and businesses during enrollment.

The <lineOfBusiness> element is included in the requests whose processing logic may depend on its value. Supported lines of business are represented in the product by the lookup **LINE_OF_BUSINESS_FLG**.

Confirmation Message

All web services that represent a taxpayer request and result in the creation of a service task within the product are designed to provide the taxpayer with a confirmation ID and a confirmation message. Additional confirmation details may be added by logic in the product that performs actions. This may be used when a taxpayer wants to inquire about the status of an action or request using the confirmation ID. A web service inquiring on the status of a task can retrieve the confirmation details to display to the taxpayer. In addition, if the taxpayer has questions at a later date about a request, the confirmation ID may be used when contacting the tax authority about the request.

The common structure captured with each message (stored in the data area **C1-SelfServiceConfirmMessage**) is as follows:

- Confirmation ID. This may be generated prior to sending the web service to the product. For example, in many cases BPEL generates a confirmation ID and adds it to the message.
- Confirmation Header. This is populated with the message category / message number used as the acknowledgement message. For example: 'Your request has been submitted successfully, reference ID 1238098'. It is defined on the service task's task type.
- Confirmation Details. This is a collection of messages (message category / message number) that may be added while processing the transaction. For example: 'Tax Clearance Certificate approved', and 'Mailed to the taxpayer on 01-02-2012'

Task Confirmation Inquiry

A taxpayer may inquire on the status of a transaction or request using a confirmation number. If so, a web service is sent to the product. The product provides the inbound service **TSGetConfirmationInformation** to process this web service. It uses the script **C1-GetCnfmID** to process the request. This service script finds the service task associated with the confirmation ID, retrieves the confirmation details stored with the task, and returns the information to the calling system.

Confirmation Email

The product provides an algorithm to allow a tax authority to send an email to the taxpayer once a task that is performing maintenance actions has completed. For example, when a taxpayer submits a payment, the service task business object that processes the payment can be configured to send an email to the taxpayer once the payment is successfully created. The base algorithm includes the confirmation details from the service task in the email along with general email text that is configurable.

FASTPATH: Refer to the algorithm section of *General Configuration Tasks* for more information about this algorithm.

Error Handling

All web services include standard error message elements. If any errors are found in processing a web service, a response is returned with the error message.

NOTE: The system error messages may not be appropriate to display to the web self service user. The message can be translated to one that is more appropriate in the BPEL layer.

Configuration Errors

If the processing script used to process any of the messages finds a configuration problem or unrecognized information from the message, it returns a generic error to the self service application like "Your request can not be processed at this time, please try again later". In addition, a To Do entry and/or an email are sent to an appropriate responsible user.

FASTPATH: Refer to the algorithm section of *General Configuration Tasks* for more information about this algorithm.

Extending the Base Logic

All web services include a "custom information" area that allows an implementation to configure up to 10 additional fields using field name and value pairs that may be populated using a custom service script.

FASTPATH: Refer to the master configuration section of *General Configuration Tasks* for more information.

Standard Message Format

All web services which communicate with the self service application follow a common structure that includes the following information:

- Common data used to track the web user access details: User ID, Name, IP Address and email address. Note that in the base business objects this information is captured in the business object data area as well as in characteristics to allow searching by these values.
- Access keys. The web service supports capturing up to 10 key fields defined with field name and value pairs.
- Access type. In conjunction with access keys, the type identifies the tax account in the system.
- Error Details data area that contains an error message.
- A custom information area for implementation specific values.
- For the requests that perform transaction processing, there is also a standard data area that defines Confirmation Details.
- Additional optional elements included in multiple messages.
  - Action indicates what type of special processing is requested.
  - Linked request element contains the confirmation number of the related web service request.
  - Line of business.

Viewing Raw XML

The 'raw XML' of the web service is captured for audit purposes on any resulting service task. It's possible that a user may need to review the raw XML to troubleshoot problems. Because the data may contain personal and sensitive information from the taxpayer, such as a person identifier, the product provides additional security configuration for viewing the raw XML on service tasks.

NOTE: Refer to the business object descriptions for the base parent business objects **C1-StandardDeferredSSTask** and **C1-StandardRealTimeSSTask** for more information about securing this logic.

Settings in Master Configuration

Many implementation wide settings are needed when configuring the product to work with the web self service application. The product captures these settings in the Self Service Master Configuration record.

In the sections in this document that explain how to configure the system to use the base self service functionality, any settings in the self service master configuration record are noted.

Inquiry Audit

The product supports the ability for implementations to capture audit records of what data users are viewing. The integration with a self service application extends this capability so that audit records may be captured when the system receives requests for information from the self service application.

A special inquiry audit business object (**C1-WebAuditInquiry**) has been provided to capture audits of information requests from the self service product. The business object captures the web user id, the web user name, the email address, the IP address and, if configured, a snapshot of the actual request XML including any response.

NOTE: The user interface for this type of inquiry audit allows a user to view the request / response XML for those inquiry audit records that capture it. Because the data may contain personal and sensitive information from the taxpayer, such as a person identifier, the product provides additional security configuration for viewing the raw XML on inquiry audits. Refer to the business object description for the base business objects **C1-WebAuditInquiry** for more information about securing this logic.

Configuring Inquiry Audit

An implementation configures which web services should cause an inquiry audit record to be created. For example, perhaps web services for a refund status inquiry are audited, but web services for inquiring on the status of a task via the confirmation ID are not. The self service master configuration record allows an implementation to configure the web services that should be audited by indicating an appropriate audit service script to invoke when the web service is processed.

FASTPATH: Refer to the master configuration section of *General Configuration Tasks* for more information.

General Configuration Tasks

This section describes the general tasks required to configure the system to support the integration. It also describes the service task type portal.

Refer to the Supported Functionality section for addition details about configuration required for specific business functionality.

Messages

Review the base messages provided for communication of positive and negative responses to the self service user. These are configured on self service task types.

- Confirmation Header message. The product provides a base message that may be used, with message category 11126 and message number 11723. The base message text may be overwritten. Or a new message in a message category reserved for implementations may be created.
- Error Header message. The product provides a base message that may be used, with message category 11126 and message number 11010. The base message text may be overwritten. Or a new message in a message category reserved for implementations may be created.

NOTE: In the BPEL layer the product's Message Category / Message Number combination is translated into message codes understood by the self service product.

Managed Content

Create a managed content record to define the HTML to use for the various general emails.

- Create one for emails routed to a responsible user when problems are found with the master configuration table.
- If confirmation emails should be sent to taxpayers when tasks with multiple steps are complete, create a managed content entry for that email text. If different service tasks should have different general text in the confirmation email, create a separate record for each variation.

The Managed Content Type should be HTML. Navigate to the Schema tab and enter the following tags and type the appropriate text for the email within the "<span>" tags.

```
<html>
  <body>
    <span> </span>
  </body>
</html>
```

XAI Sender

Verify that an XAI sender is configured for routing email real time.

- Invocation Type should be set to **Real-time**
- XAI Class should be set to **RTEMAILSNDR**
- On the Context tab, the context type **SMTP Host name** should be defined with a valid context value.

NOTE: The XAI Options includes an option for a default Sender for real-time emails that can be configured with this XAI sender.

Field

Define a field that includes the text for the email subject for the notification of master configuration errors email.

Define one or more fields that include the text for the email subject for the confirmation emails to send for the various service tasks.

Algorithm

If any service task should send a confirmation email after all steps are complete, for example once a payment is processed, configure one or more appropriate algorithms for the **C1-SSCONFREM** algorithm type.

- Set the Email Subject field created above for the confirmation email.
- Set the Predefined Text to the Managed Content created above for the confirmation email.
- Enter the From Address that should be used in the generated email
- Enter the XAI Sender for Email defined above.

NOTE: This algorithm must be plugged into appropriate business objects as an enter algorithm on a state where actions done by the task are complete.

Access Type

Base product is provided with the implementation of access type *Tax Role* with keys **PER_ID** and **TAX_ROLE_ID**. If required, additional *access types* should be added to the lookup **USER_ACCESS_TYPE_FLG**.

Line of Business

Base product is provided with the implementation two Lines of Business: **Individual** and **Business**. If required, additional *line of business* entries should be added to the lookup **LINE_OF_BUSINESS_FLG**.

Self Service Master Configuration

Create a Self Service Master Configuration record that holds system wide configuration needed for the self service integration.

The Configuration Support section defines information needed to alert a user that problems were found with configuration. The alert may be sent using a To Do entry or an email or both.

- Define the user that is responsible for technical issues related to self service who should receive an email if there is an issue.
- To Do Type should be set to a standard error To Do type to use when issues are found with the master configuration table. The base product To Do Type **Self Service Master Configuration Issues (C1-SSMCI)** is provided for this purpose.
- To Do Role should be configured to an appropriate default To Do role for the above To Do type.

Populate the following fields if an email should be generated when configuration problems exist.

- Enter the From Email Address that should be used in the email.
- Enter the From Name that should be used in the email.
- Enter the XAI Sender created above or leave it blank to use the default from XAI options.
- Set the Email Content to the Managed Content record created above.
- Set the Email Subject field created above.
- Provide the Configuration Support Email Script that produces the email. The product has supplied **C1-SSIssueEm** which may be used here.

The Request Processing Control information allows an implementation to define special logic that should occur when a web service is processed. The special logic may be causing an inquiry audit record to be created or it could be to indicate a custom extension to a base product service by retrieving additional details to return in a web service call.

- To Do Type for Audit Script Errors should be set to an appropriate To Do type. The base product To Do Type **Self Service Task Audit Issues (C1-SSAI)** is provided for this purpose.
- To Do Role for Audit Script Errors should be configured to an appropriate default To Do role for the above To Do type.
- To Do Type for Custom Extension Errors should be set to an appropriate To Do type. The base product To Do Type **Self Service Task Custom Extension Issues (C1-SSCSI)** is provided for this purpose.
- To Do Role for Custom Extension Errors should be configured to an appropriate default To Do role for the above To Do type.

For any web service that requires inquiry audit or custom extension, determine the schema associated with the XAI inbound service that processes the web service.

- Schema Type and Schema Name should be set to the appropriate object that the XAI inbound service references.

- For enabling inquiry audit, check the Inquiry Audit Enabled button. If the raw XML of the request and response should be captured, check the Store Response Data checkbox. Indicate the Audit Script. The base product provides the script **C1-AddWebAI** that may be used.
- For providing additional data in a web service response, provide a Custom Extension Script. The schema for the custom extension script should match the schema that it is extending. It populates data in the data area **C1-SSCustomExtensionFields**.

Setting Up Taxpayer Identification

Base product functionality uses a person's birth date as one of the identification details. If your implementation captures the birth date on the person record and would like that to be part of the taxpayer identification logic, the setup described in this topic is required.

NOTE: The product does not provide any functionality for populating the birth date characteristic on the person.

Characteristic Type

Create a business object for individual person that supports display and maintenance of the birth date characteristic.

Business Object for Individual Person

Create a business object for an individual person that supports display and maintenance of the birth date characteristic:

1. Add a child business object for **C1-PersonIndividual**.
2. Copy the schema.
3. Add the element representing a flattened characteristic of the type created above, placing it in the desired location for the rendered user interface. In addition, include the appropriate **startSection** and **endSection** definitions.

The following example illustrates the schema for a child business object created for the base business object **C1-PersonIndividual** and characteristic type **BIRTHDAT** so that the birth day appears in a new ***Additional Information*** section after the ***Contact Information***.

```
<schema xmlns:uiHint="http://oracle.com/ouafUIHints">
<includeDA name="C1-PersonCommon"/>
<includeDA name="C1-PersonExtendedNameDetails"/>
<includeDA name="C1-PersonId"/>
<includeDA name="C1-PersonPhone"/>
<includeDA name="C1-PersonAddress"/>
<includeDA name="C1-PersonContactInfo"/>
<uiHint:startSection mdField="ADDITIONAL_INFO" sectionColumn="right"/>
    <birthDate mdField="C1_SS_BIRTHDATE" dataType="date" mapField="ADHOC_CHAR_VAL">
        <row mapChild="CI_PER_CHAR">
            <CHAR_TYPE_CD is="BIRTHDAT"/>
            <EFFDT is="%effectiveDate"/>
        </row>
    </birthDate>
    <effectiveDate dataType="date" default="1950-01-01" mapField="EFFDT" rowRef="birthDate" suppress="true"/
>
<uiHint:endSection/>
<includeDA name="C1-PersonPerson"/>
</schema>
```

Person Type

Specify the business object created above on person type(s) that represent individuals in your implementation.

Setting Up Service Task Types

Service Task Types contain the rules that control how service tasks are processed.

To set up a service task type, select **Admin Menu** > **Self Service Task Type**.

The topics in this section describe the base-package zones that appear on the Service Task Type portal.

Self Service Task List

The Service Task Type List Zone lists every self service task type. The following functions are available:

- Click the broadcast button to open other zones that contain more information about the adjacent service task type.
- The standard actions of **Edit, Delete** and **Duplicate** are available for each service task type.
- Click the **Add** link in the zone's title bar to add a new service task type.

Self Service Task Type

The Self Service Task Type zone contains display-only information about a service task type. This zone appears when a service task type has been broadcast from the Self Service Task Type List zone or if this portal is opened via a drill down from another page.

Please see the zone's help text for information about this zone's fields.

Supported Functionality

This section describes the out of the box functionality provided for the integration along with the required configuration tasks.

Enrollment

After registering on a self-service portal and establishing a secure login, the user should explicitly request access to his/her taxes. The request is evaluated, and the user/tax account association(s) are recorded in the system and become available for account access verification.

A single web portal login provides the self service user with access to all tax accounts. In other words an individual who also happens to be a business owner may log in once and view and manage both individual and corporate taxes.

A user enrolled into a tax account is able to view account financial information, file tax returns and make payments related to this account, update contact and correspondence information, and request services.

NOTE: See the *Tax Account Access Information* section for a detailed explanation about tax account access.

User Access Store

The User Access store is a table in which the link between users and tax accounts is stored. It is maintained and accessed exclusively from the integration layer via SOA composites. The integration provides the information to the self service product and other solution components.

- The following information is stored:
- Enrollment ID

- Web User ID
- Line of Business
- Access Type
- Access keys 1 - 10 (Key Name and Key Value)
- Revenue Management System Name
- Status

The records marked with same enrollment ID/web user ID/line of business comprise an enrollment "event".

The possible statuses of the user access records are represented in the product by the lookup **ENROLLMENT_STATUS_ FLG.**

Initial Enrollment and Enrollment Event

The self service user is asked to provide identification details sufficient to determine tax accounts that this user owns. A separate enrollment request is made for each line of business.

The initial enrollment request creates an enrollment "event" that is "known" to two solution components: the integration layer and the revenue management system(s). Service task represents an enrollment event in the base product. User account access data captured in the user access store includes enrollment ID, web user ID, and line of business, thus associating it with a specific enrollment event.

The solution supports configurable identification details and algorithm-based enrollment request processing. In the base product, initial enrollment is implemented using an *Enrollment Service Request*.

The request is handled by a special web service TSEnrollmentServiceRequest. The results of a successful enrollment are:

- Creation of one or more records in the user access store. Each record is associated with enrollment ID. One record is created for each accessible tax account. The status of the record indicates that the access to a tax account is either approved or requires an additional review.

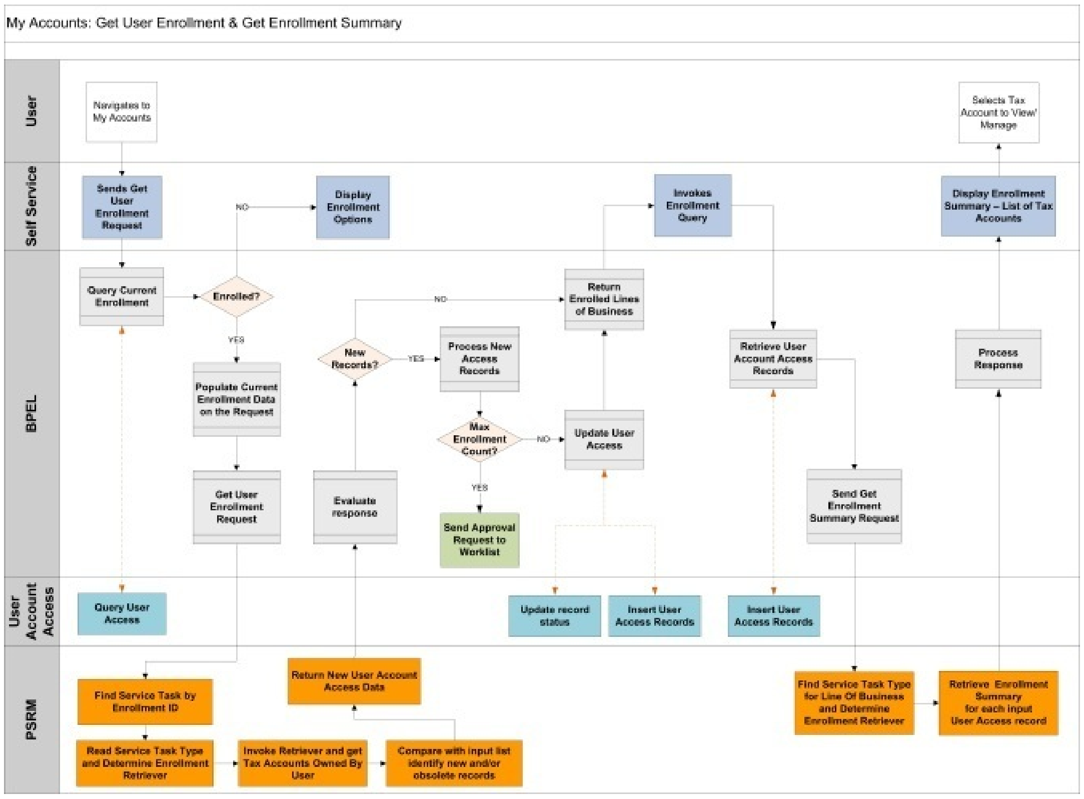
  NOTE: The status of the record is set according to service task type configuration.

- Creation of a service task associated with both enrollment ID and confirmation ID. The initial list of user access records is captured on the service task.

  NOTE: See the *Enrollment Request* section for detailed information about enrollment processes and related configurations.

Enrollment Refresh and Summary

The following diagram illustrates the process flow on the **My Accounts** (Enrollment Summary) portal page.


My Accounts: Get User Enrollment & Get Enrollment Summary
User
Self Service
BPEL
User Account Access
PSRM
Navigates to My Accounts
Sends Get User Enrollment Request
Query Current Enrollment
Enrolled?
NO
YES
Display Enrollment Options
Populate Current Enrollment Data on the Request
Get User Enrollment Request
Query User Access
Find Service Task by Enrollment ID
Read Service Task Type and Determine Enrollment Retriever
Invoke Retriever and get Tax Accounts Owned By User
Compare with input list identify new and/or obsolete records
Return New User Account Access Data
Evaluate response
New Records?
Process New Access Records
Max Enrollment Count?
Send Approval Request to Worklist
Update User Access
Return Enrolled Lines of Business
Invokes Enrollment Query
Update record status
Insert User Access Records
Retrieve User Account Access Records
Send Get Enrollment Summary Request
Find Service Task Type for Line Of Business and Determine Enrollment Retriever
Retrieve Enrollment Summary for each input User Access record
Process Response
Display Enrollment Summary – List of Tax Accounts
Selects Tax Account to View/ Manage

Implicit Enrollment Refresh

User access information is implicitly refreshed upon login. For example, a business owner that opened a new business and registered it with the tax authority would be automatically given an access to this new business account.

The request is handled by a special web service, **TSGetUserEnrollment**.

The integration has the following steps:

- The request is initiated from the self service product. The XML message contains web user ID.
- The SOA Composite is querying the user access store, retrieves all enrollment records and populates them on the request; each record includes tax account identifiers line of business and enrollment ID.
- The request is forwarded to the product. The processing service script finds enrollment request service task associated with each input enrollment ID. It reads service task type configuration under *Enrollment Instructions*, and invokes the enrollment retrievers associated with the input access type. The input list of tax accounts is compared with the retriever's response, and any new entries are added to the output. Entries that are no longer present on the list are added to the output with *Inactive* status.

NOTE: See *New and Updated User Access* for more information. Also refer to the business service **C1-GetEnrollment** for additional details.

- The response is processed by the integration layer and new user's tax accounts are added to the user access store.

The results of a successful enrollment refresh may be a creation of one or more new records in user access store. Each record is stumped with the original enrollment ID/line of business combination.

Enrollment Summary

Upon login the self service product displays the user enrollment summary that contains a list of all tax accounts that the user is eligible to view and manage. This is an inquiry request that defines its own web service (**TSGetEnrollmentSummary**). The integration has the following steps:

1. The request is initiated from the self service product. The XML message contains the user ID.
2. The SOA Composite queries the user access store and retrieves all enrollment records. Each record includes tax account identifiers (access type and access keys), line of business, and enrollment ID.
3. The list of enrollment records is delivered to the system. The processing service script finds the enrollment service task associated with the input enrollment ID, reads the service task type, and calls the account summary retriever. The retriever also marks one enrollment record as a default. When the web service response reaches portal application, this record is automatically pulled into a self service session context.

Summary Contents

The self service product displays tax account highlights on the **My Accounts** portal page. The single tax account enrollment summary includes two components:

- **Title** - a short description of the tax account.
- **Details** - essentials about tax account status.

Both parts of the summary are composed using message text with the substitution parameters.

The base product is expected to supply the actual data items, including dates, numbers, and types. This information is used by the self service product as substitution parameters for the Summary messages.

The enrollment summary retriever is expected to deliver the following information:

- Summary Title parameters list.
- Summary Details parameters list.
- Taxpayer Name (required, used internally by the self service product).
- Account Name (optional, used internally by the self service product).

The enrollment summary retrievers provided with the base product return the following

- **Individual** enrollment summary - returns a summary title parameter *tax type* and summary details parameters: tax role's *start* and *end date* and an *outstanding balance* forecasted for the current date.
- **Business** enrollment summary - returns a summary title parameter *tax type* and summary details parameters: tax role's *start* and *end date* and an *outstanding balance* forecasted for the current date and a *formatted address*.

NOTE: Refer to the business service **C1-GetBusinessTaxRoleSummary** as an example of a tax account enrollment summary retriever.

New and Updated User Access

Enrollment retriever logic compares the result list of tax accounts with the initial enrollment data that is captured on the service task and performs the follows:

- Adds the new entries to the output new keys list.
- If an entry exists on the initial enrollment list but no longer returned by the retriever, the logic interprets it as if the taxpayer is no longer owns the tax account. Such record is added to the new keys list with an **Inactive** status.

NOTE: Refer to the business service **C1-EnrolIndividualAccntTaxRole** as an example of an access type-specific enrollment retriever.

Failed Enrollment Attempts

The self service solution provides a mechanism to monitor enrollment attempts and block suspicious and fraudulent web user activities.

A service task "in error" is created every time an enrollment process ends in error. This allows tracking and auditing of the failed attempts. The results of unsuccessful enrollment attempt are:

- A service task in error is created. It is associated with the enrollment ID.
- The preceding failed enrollment attempts determined as service task(s) in error with the same enrollment ID. If found, they are linked to the current service task as related objects.
- A single record in error status is created in the user access store (this is handled by a SOA Composite in the integration layer).

Enrollment Review

The self service product allows manual review of the problematic enrollment requests. Under certain conditions the SOA Composite may place one or more records in the user access store on hold and create a **Worklist** task.

See the *Oracle Public Sector Revenue Management Self Service Implementation Guide* for more information about how problem enrollment requests are handled.

Configuration Tasks for Enrollment

The following sections list the configuration tasks required to implement the base user enrollment functionality.

Master Configuration

The self service master configuration record includes settings required for enrollment processing.

- **Revenue Management System**. Define the alphanumeric code that will be referenced by records in the user access store.

  NOTE: See the *User Access Store* section for more details.

Service Task Type

Configure Enrollment Request service task type for every line of business supported by your implementation. This service task type contains definitions used by *Initial Enrollment Request*, *Implicit Enrollment Refresh* and *Enrollment Summary* functionality.

See *Configuration Tasks for Enrollment Request* for detailed information on enrollment configurations.

Integration Mapping

Tax types and **taxpayer types** used by the product should have corresponding values defined in the self service product.

Your implementation may decide to use exactly the same values in both products or configure the mapping using domain value mapping in the SOA Composer application

Customizing Enrollment

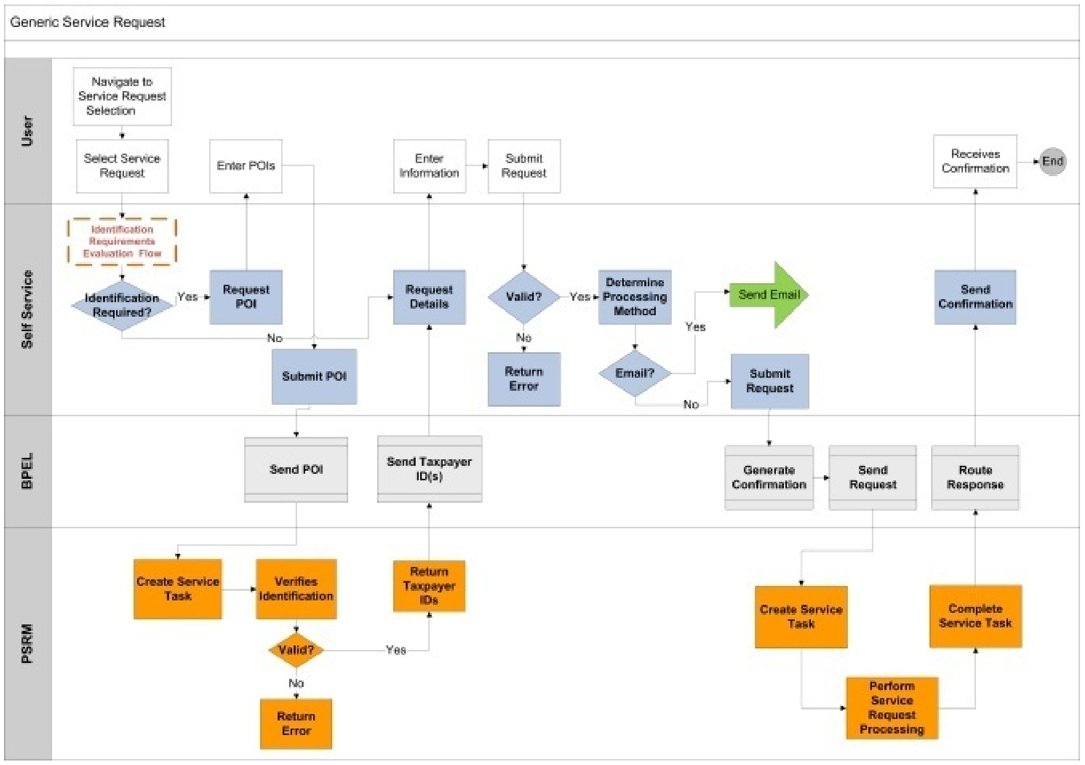
If your business requirements are not satisfied by the functionality provided by the base product consider the following customization options:

- **The information displayed on My Accounts**. To customize the contents of a tax account summary you should create custom Access Type Summary Retriever and adjust the enrollment request service task type configurations and also the corresponding configurations in self service product (message).
- **The way tax accounts owned by the user are determined**. Create custom User Access Type Summary retriever(s) and adjust the corresponding enrollment request service task type configurations.
- **The information required in order to verify self service user identity and credentials** If different from what is supported by a base product, you should create new enrollment service task business object to capture the information and implement an alternative User Enrollment Retriever(s). Reconfigure the corresponding enrollment service task types to reference the new business object and set up new request field mapping.
- **Additional steps should be included in the initial enrollment handling**. Utilize standard business object plug in spots available the enrollment service task.
- **New access type**. Implementation of a new access type should be closely coordinated with the self service product.
  - Define access keys to be used for the new access type. **Note**: key 1 is always reserved for the Taxpayer ID.
  - Determine the lines of business that are applicable for this access type.
  - Implement User Enrollment Retrievers and Access Type Summary Retrievers for all applicable lines of business.
  - Configure enrollment service task types.
  - If the implementation should support an option to pay a balance for the tax account defined by this new access type, implement a new payment destination, including distribution rule, if needed.
  - Create corresponding configurations for Access Type in the self service product and add mapping in the integration layer

NOTE: See *Configuration Tasks for Enrollment Request* for detailed information on enrollment configurations.

Taxpayer Service Request

The self service product supports service requests by the taxpayer. Some requests may be fulfilled by the self service product, such as information about when to file or technical support requests. Other requests may require information from this product. The following diagram illustrates an expected process flow.



The following points highlight the process:

- The service request may or may not require taxpayer identification up front. If it does, the initial processing works as follows:
  - **For a casual user**: The self service application captures information about the taxpayer and sends a web service request to this product to validate the taxpayer (proof of identity - POI).
  - **For an enrolled user**: The identification is not performed.

  NOTE: For additional information, see *User Identity Verification*.

- The product receives a web service request to identify the taxpayer. The product provides the inbound web service **TSTaxpayerIdentification** to process the message. It uses the script **C1-IdentTaxp** to process the request. This should cause an appropriate service task to be created based on the type and populate the relevant data for the service task. The service task should include algorithms to identify the taxpayer and immediately respond to the web service call.

  NOTE:
  For additional information see *Configuration Tasks for Taxpayer Service Request*.

- For any service request that requires action by the product, another web service request is received with information about the request, including the task type to create. The product provides the inbound web service **TSTaxpayerServiceRequest** which uses the script **C1-TaxSvcReq** to process the request. This should cause an appropriate service task to be created based on the type and populate the relevant data for the service task. Depending on the requirements for a particular service request, the service task may be designed to do the following:
  - Give real-time feedback/results (if the request simply involves information retrieval)
  - Perform the task as a separate step without the taxpayer waiting for a reply. In this case the taxpayer should receive a confirmation ID related to the request that is provided in the web service so that at a later date the progress of the service request can be queried by the taxpayer.

NOTE: See *Implementing A New Service Request* for more information.

The base product provides functionality out of the box for various categories of service requests, including a taxpayer's request for a tax clearance certificate. The topics below describe more information about the provided functionality, followed by configuration tasks.

Configuration Tasks for Taxpayer Identification Service Request

The following sections list the configuration tasks required to implement the base taxpayer identification functionality.

Algorithms

Create the following algorithms:

- Create an algorithm for algorithm type **C1-IDTXPSR**. Populate the Birth Date Required parameter as per business rules. Define the characteristic type for the birth date that was created above (also see *Setting Up Taxpayer Identification* in the topic, *General Configuration Tasks*.

Business Object

Update the business object **C1-TaxpayerIdentSvcReqSSTask** to include this algorithm. Navigate to the Lifecycle tab and on the **In Progress** state, add an entry to the Algorithm collection: System Event is **Enter**, Algorithm is the one created above.

Service Task Type

Create a service task type for taxpayer identification to use for service requests. Use the business object **Taxpayer Service Request Self Service Task Type**.

- The related transaction BO should be set to **C1-TaxpayerIdentSvcReqSSTask**.
- Service task class should be set to **Service Request**.
- To Do Type should be set to a standard error To Do type to use when transitioning to error. The base product To Do Type **Self Service Task Issues (C1-SSTTD)** is provided for this purpose.
- To Do Role should be configured to an appropriate default To Do role for the above To Do type. This is optional. If no value is provided the default role for the To Do type is used.
- Configure the Confirmation Header Message Category and Message Number and the Error Header Message Category and Message Number to use for communication to the self service user.

FASTPATH: Refer to the Messages section of the *General Configuration Tasks* for more information.

- The service request fields mapping is used when creating the target service task to correctly populate the requestField list from the list of fields passed in on the web service request XML. For the base transaction business object, the following fields are expected:

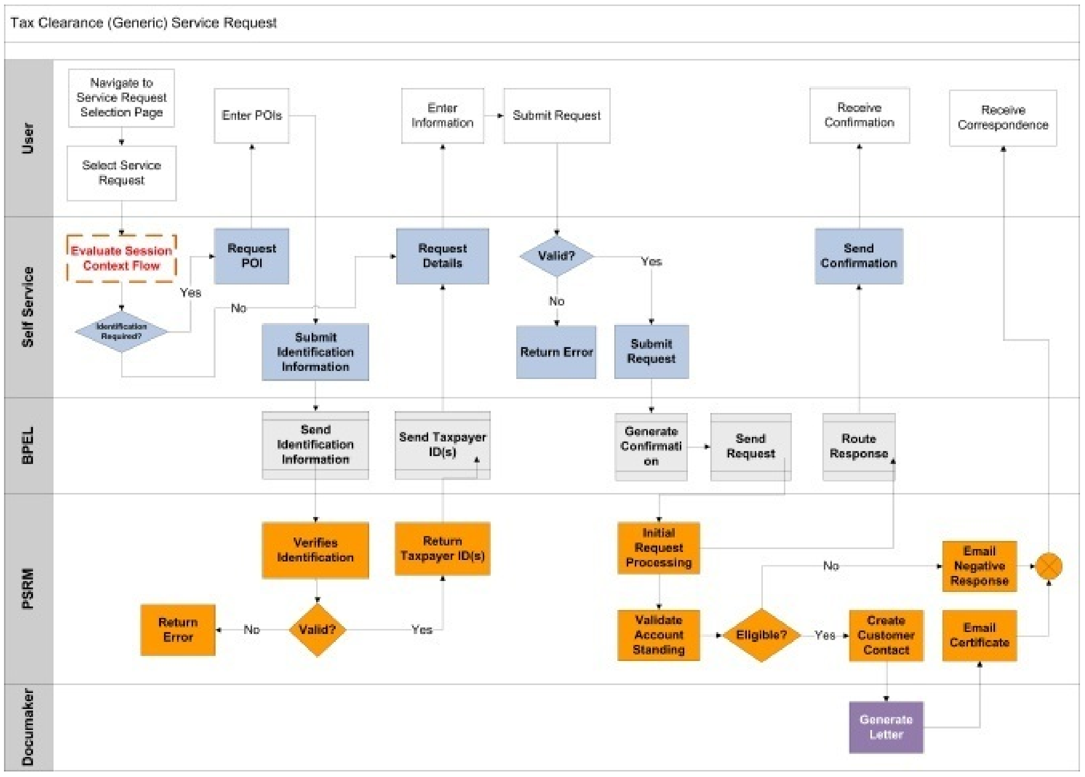
Sequence	Transaction BO XPath
1	requestData/legalName
2	requestData/birthDate
3	requestData/personType
4	requestData/idType

5	requestData/personIdNumber

FASTPATH: Service Task Type Mapping. The service task type defined here must be configured in the service task domain value mapping in the SOA Composer application. Refer to the self service product documentation for details.

Tax Clearance Certificate

On occasion a taxpayer may be required to produce a Tax Clearance Certificate to a third party. This must be requested from the tax authority as a written confirmation that a taxpayer is considered in good standing as of the date of issue of the Certificate. The following diagram illustrates the process flow supported for issuing a tax clearance certificate.



The following points describe in more detail the functionality provided

- This type of service request requires taxpayer identification first.
- The product provides a base service task business object called **C1-TaxClearCertSvcReqSSTask** that processes the request. It calls an algorithm to determine if the account is in good standing. If so, it creates a customer contact to issue the certificate that may be emailed. If not, it creates a customer contact to inform the taxpayer that a certificate is not possible.
- The product provides a letter extract for a sample tax clearance certificate. In addition a letter template for Documaker is included in this functionality.

Configuration Tasks for Tax Clearance Certificate

The following sections list the configuration tasks needed to implement the tax clearance certificate functionality.

Lookup

Using the lookup **C1_TAX_CLEARANCE_PURPOSE_FLG** define the values that are valid for your implementation.

Characteristic Types

Create the following characteristic types:

- **Tax Clearance Purpose**. This char type is used to capture the tax clearance purpose on the customer contact for audit purposes. Define Customer Contact as a characteristic entity. The tax clearance purpose is defined in a lookup field (**C1_ TAX_CLEARANCE_PURPOSE_FLG**). The characteristic type should be defined as a predefined characteristic with the valid values from the lookup repeated.

Letter Template

Create a letter template to represent the tax clearance certificate. Reference the extract algorithm **C1-LTR-TCSDC**. Batch control is required and **LTRPRT** can be entered here.

Customer Contact Class / Type

Define or identify an appropriate Customer Contact Class for the tax clearance certificate customer contact types.

Create a Customer Contact Type for issuing the tax clearance certificate.

- Customer Contact Business Object is not required.
- Contact Shorthand is not applicable.
- Set the Contact Action to **Send Letter** and indicate the letter template created above.
- For the template characteristics, define the Characteristic type created above for Tax Clearance Purpose. In addition, define the (base) characteristic type Expiration Date.

Create a Customer Contact Type for declining the Tax Clearance Certificate.

- Customer Contact Business Object is not required.
- Contact Shorthand is not applicable.
- Leave Contact Action blank.
- For the template characteristics, define the Characteristic type created above for Tax Clearance Purpose.

Managed Content

Create managed content records to define the HTML to use for the email related to the tax clearance certificate. Define one for the email that is sent when issuing the certificate. Define another for the email that is sent when the request is denied.

For both records, the Managed Content Type should be HTML. Navigate to the Schema tab and enter the following tags and type the appropriate text for the email within the "<span>" tags.

```
<html>
  <body>
    <span> </span>
  </body>
</html>
```

Field

Define a field that includes the text for the email subject.

Algorithms

Create the following algorithms:

- Evaluate Outstanding Debt. Create an algorithm for the algorithm type **C1-EVALDEBT**. Populate the Max Outstanding Debt Allowed parameter.
- Evaluate Gap in Filing. Create an algorithm for the algorithm type **C1-EVALGAP**. Populate the Overdue Process Template and the No-gap Filing Period parameters.
- Email Tax Clearance Certificate Letter. Create an algorithm for the algorithm type **C1-EMTXCL**. Populate the parameters:
    - Set the Customer Contact Class created above.
    - Set the Customer Contact Types for Issue Tax Clearance Certificate and for Decline Tax Clearance Certificate created above.
    - Set the Characteristic Type for Tax Clearance Certificate Purpose
    - Set the Predefined Email Text for the Issuing Certificate Email and the Decline Request Email configured as Managed Content above.
    - Set the Email Subject field created above.
    - Enter the Characteristic Type for Certificate Expiration Date: C1-EXPDT
    - Enter the number of days that is the Certificate Valid Period
    - Enter the XAI Sender for Email created as part of the general configuration options.
    - Enter the From Email Address that should be used in the generated email
    - Enter the From Name that should be used in the generated email

Business Object

Update the business object **C1-TaxClearCertSvcReqSSTask**. Navigate to the Lifecycle tab.

- On the **In Progress** state, add an entry to the Algorithm collection: System Event is **Enter**, Algorithm is set to the one created above for Evaluate Outstanding Debt.
- On the **In Progress** state, add an entry to the Algorithm collection: System Event is **Enter**, Algorithm is set to the one created above for Evaluate Gap in Filing.
- On the **Complete** state, add an entry to the Algorithm collection: System Event is **Enter**, Algorithm is set to the one created above to Email the Tax Clearing Certificate Letter.

Service Task Type

Create a service task type for the tax clearance certificate service task with the business object **Taxpayer Service Request Self Service Task Type**.

- The related transaction BO should be set to **C1-TaxClearCertSvcReqSSTask**.
- Service task class should be set to **Service Request**.
- To Do Type should be set to a standard error To Do type to use when transitioning to error. The base product To Do Type **Self Service Task Issues (C1-SSTTD)** is provided for this purpose.
- To Do Role should be configured to an appropriate default To Do role for the above To Do type. This is optional. If no value is provided the default role for the To Do type is used.
- Configure the Confirmation Header Message Category and Message Number and the Error Header Message Category and Message Number to use for communication to the self service user.

FASTPATH: Refer to the Messages section of the *General Configuration Tasks* for more information.

- The service request fields mapping is used when creating the target service task to correctly populate the requestField list from the list of fields passed in on the web service request XML. For the base transaction business object, the following fields are expected:

Sequence	Transaction BO XPath
1	requestData/emailAddress
2	requestData/phoneNumber
3	requestData/taxClearancePurpose
4	requestData/isPaperCopyReq

FASTPATH: Service Task Type Mapping. The service task type defined here must be configured in the service task domain value mapping in the SOA Composer application. Refer to the self service product documentation for details.

Scheduled Reminder

A self-service user may schedule an automatic email reminder. The reminder can be scheduled for an infinite time or expire on a certain date. The monthly reminder functionality is provided by the product. This is a standard taxpayer service request.

NOTE: See *Taxpayer Service Request* for details on the request flow.

The following points describe the functionality provided:

- This type of service request does not require taxpayer identification.
- The product provides a base service task business object called C1-SchedMonthlyReminderSSTask that processes the request. It creates a reminder service task in active state. Periodic monitor batch process selects the active reminder tasks and triggers an algorithm to determine if the expiration date is reached. If not yet, the subsequent algorithm checks if the scheduled day of the month is reached and if so, sends an email reminder. If the expiration date id reached or in the past, the reminder task becomes inactive.

Configuration Tasks for Scheduled Reminder

The following sections list the configuration tasks needed to implement the scheduled reminder request functionality.

Managed Content

Create managed content records to define the HTML to use for the email reminder.

The Managed Content Type should be HTML. Navigate to the Schema tab and enter the following tags and type the appropriate text for the email within the "<span>" tags.

```
<html>
  <body>
    <span> </span>
  </body>
</html>
```

Field

The product is provided with field C1_SCHED_REMINDER_SUBJ Tax Payment Reminder. Use it for the reminder email subject or define a new field that includes the text for the email subject.

Service Task Type

Create a service task type scheduled monthly reminder request. Using the business object Scheduled Reminder Request Task Type:

- The related transaction BO should be set to C1-SchedMonthlyReminderSSTask.
- Service task class should be set to Service Request for To Do.
- To Do Type should be set to a standard error To Do type to use when transitioning to error. The base product To Do Type Self Service Task Issues (C1-SSTTD) is provided for this purpose.
- To Do Role should be configured to an appropriate default To Do role for the above To Do type. This is optional. If no value is provided the default role for the To Do type is used.
- Configure the Confirmation Header Message Category and Message Number and the Error Header Message Category and Message Number to use for communication to the self service user.

  NOTE: **Fastpath:** Refer to the Messages section of the *General Configuration Tasks* for more information.

- The service request fields mapping is used when creating the target service task to correctly populate the requestField list from the list of fields passed in on the web service request XML. For the base transaction business object, the following fields are expected:

Sequence	Transaction BO XPath
1	requestData/name
2	requestData/emailAddress
3	requestData/phoneNumber
4	requestData/dayOfMonth
5	requestData/expirationDate

Configure Email Notification Instructions:

- Set the Email content configured to Managed Content created above.
- Set the Email Subject to Field created above.
- Enter the From Email Address that should be used in the generated email
- Enter the From Name that should be used in the generated email
- Enter the XAI Sender for Email created as part of the general configuration options

NOTE: Service Task Type Mapping. The service task type defined here must be configured in the service task domain value mapping in the SOA Composer application. Refer to the self service product documentation for details.

Generic To Do Service Request

Some services requested by the taxpayer may require manual review and idiosyncratic processing. The integration supports the scenario where the incoming service request results in creation of a service task which simply stores request fields (name-value pair collection) and creates a To Do entry that is referencing the service task.

NOTE: See *Taxpayer Service Request* for details on the request flow.

This approach allows utilizing single service task business object to capture different types of requests.

Configuration Tasks for Generic To Do Request

The following sections list the configuration tasks needed to implement the generic service request functionality.

Service Task Types

Create a service task type for each generic create-To-Do service request. Use the business object Generic Create To Do Request Task Type.

- The related transaction BO should be set to C1-GenericToDoSSTask.
- Service task class should be set to Service Request for To Do.
- To Do Type should be set to a standard error To Do type to use when transitioning to error. The base product To Do Type Self Service Task Issues (C1-SSTTD) is provided for this purpose.
- To Do Role should be configured to an appropriate default To Do role for the above To Do type. This is optional. If no value is provided the default role for the To Do type is used.
- Configure the Confirmation Header Message Category and Message Number and the Error Header Message Category and Message Number to use for communication to the self service user.

  FASTPATH: Refer to the Messages section of the *General Configuration Tasks* for more information.

- Request processor should be set to C1-PopSvRqDt.
- The service request fields mapping is used when creating the target service task to correctly populate the requestField list from the list of fields passed in on the web service request XML. For the base transaction business object, the following fields are expected:

Sequence	Transaction BO XPath
1	requestData/requestComments

To configure request processing instructions:

- Create To Do Type should be set to a To Do type to use when processing the request. The base product provides Generic Service Request To Do (C1-GSRTD) for this purpose.

  NOTE: To distinguish between various generic requests you may prefer to create a designated To Do Type for each one.

- Create To Do Role should be configured to an appropriate default To Do role for the above To Do type. This is optional. If no value is provided the default role for the To Do type is used.

NOTE: **Service Task Type Mapping**. The service task type defined here must be configured in the service task domain value mapping in the SOA Composer application. Refer to the self service product documentation for details.

Refund Status

In this release, the integration supports the ability for a taxpayer to inquire on the status of a tax refund. This is a special kind of taxpayer service request that defines its own web service. The integration has the following steps:

- The self service application captures information about the taxpayer, a way of identifying the filing period for the refund in question and a "shared secret" that ideally only the taxpayer and the tax authority would know (such as the expected refund amount).
- This information is sent to the system, which creates a Refund Status Inquiry service task.
- The task's lifecycle is designed to process the information immediately with no deferred monitor process so that the information can be returned to the self service application real-time.
- The base business object for the Refund Status Inquiry service task includes an algorithm to verify the taxpayer identification, confirm that a form has been received for the filing period, determine the status of the refund and compose an appropriate message to send back to the taxpayer immediately.

Note that the implementation of the refund status inquiry creates a service task for the following reasons:

- The creation of a service task with its business object architecture provides many plug-in spots to allow implementations to easily customize the logic for identifying the taxpayer and determining the refund status.
- The status of a refund is based on conditions in the system that can change over time. Instantiating a service task for the refund status inquiry provides a way of capturing the point in time that the request was received and being able to reconcile this with the information sent to the taxpayer.
- Creating a service task enables the tax authority to provide the taxpayer with a confirmation ID for possible follow up in the future. For example, imagine a taxpayer is told "Your refund has been processed. You should receive it shortly." and several weeks later the taxpayer has still not received the refund. If the taxpayer contacts the tax authority and provides the confirmation ID, the user can view the service task, verify the details and more easily investigate the situation.

The topics below describe more information about the provided functionality, followed by configuration tasks.

Refund Inquiry Service Task

The web service requesting the refund status is processed by the XAI Inbound Service **TSGetRefundStatus**, which invokes the service script **C1-RSISvcReq**. The appropriate service task type to use must be passed in with the other information received from the self service system.

Identifying the Shared Secret Amount

The logic provided in the base product assumes that the shared secret is an amount on the taxpayer's form. Form definition in the system is configurable and each form may be defined with different form lines that capture the amount used for the shared secret. For each form type, the system needs to know which form line is the one used for the shared secret. This information is defined in the self service master configuration.

FASTPATH: Refer to *Configuration Tasks for Refund Status* for information about configuring the system to support these payment destinations.

Determining the Status of the Refund

The lifecycle of creating a tax refund in the system includes the following steps.

- The taxpayer files the form, which is received by the system. The form gets processed and appropriate adjustments are created for the obligation for the filing period to record the affect on the balance.
- Once the taxpayer's obligation has a credit balance, an overpayment process is used to process the refund. How the overpayment process is created depends on the business practice. An account debt monitor may be used to detect obligations with a credit and can create the overpayment process. The form can include logic to immediately create the overpayment process based on the existence of a credit balance based on a form rule.

  NOTE: The base product does not currently supply a form rule to create the overpayment process at posting time. A custom form rule may be developed.

- The overpayment process may require approval. Once approvals are processed, the overpayment process may have logic to use the credit to apply to other debt. Or it may carry the credit forward to a future filing period (typically upon the taxpayer's request). When the overpayment process completes, it has a record of carry forward, offset and refund amounts. If there is any refund, the overpayment creates an accounts payable (A/P) adjustment if the refund should be provided using a paper check. It creates a credit payment with appropriate bank information to process the refund as a direct deposit.

An algorithm plugged into the refund status inquiry service task business object as an enter plug-in on the **In Progress** state is responsible for determining the status of the refund. Refer to the description of the algorithm for more information about the base logic provided.

Configuration Tasks for Refund Status

The following sections list the configuration tasks needed to implement the refund status functionality.

Service Task Type

Create a service task type for the refund status inquiry service task with the business object **Standard Self Service Task Type**.

- The related transaction BO should be set to **C1-RefundStatusInquirySSTask**.
- To Do Type should be set to a standard error To Do type to use when transitioning to error. The base product To Do Type **Self Service Task Issues (C1-SSTTD)** is provided for this purpose.
- To Do Role should be configured to an appropriate default To Do role for the above To Do type. This is optional. If no value is provided the default role for the To Do type is used.
- Configure the Confirmation Header Message Category and Message Number and the Error Header Message Category and Message Number to use for communication to the self service user.

  FASTPATH: Refer to the Messages section of the *General Configuration Tasks* for more information.

FASTPATH: Service Task Type Mapping. The service task type defined here must be configured in the service task domain value mapping in the SOA Composer application. Refer to the self service product documentation for details.

Master Configuration

The self service master configuration record includes several settings required for refund status functionality.

- Shared Secret Form Line. For each form type that is supported for a taxpayer to inquire about a refund, indicate the XPath of the form line reference. Note that there is an ability to view the schema of the form business object associated with the form type to find the correct XPath.

  NOTE: Reported vs. Current. Because the configuration is related to information that the taxpayer has reported, consider whether the "current" element (asCurrent) or "reported" element (asReported) is used. If the taxpayer has made calculation mistakes, form rules or users may adjust the current value of the shared secret field. In this case the reported value is what the taxpayer knows and would enter.

- Overpayment Process Amounts. In order to be able to provide details related to the refund, it is necessary to report amounts that may have been used to offset other debt and amounts that are carried forward to future filing periods. The information is defined as elements in overpayment process business object. For each Overpayment Process Type, indicate the XPath values for the Refund Amount, the Offset Amount and the Carry Forward Amount. Note that there is an ability to view the schema of the overpayment process business object associated with the overpayment process type to find the correct XPath.

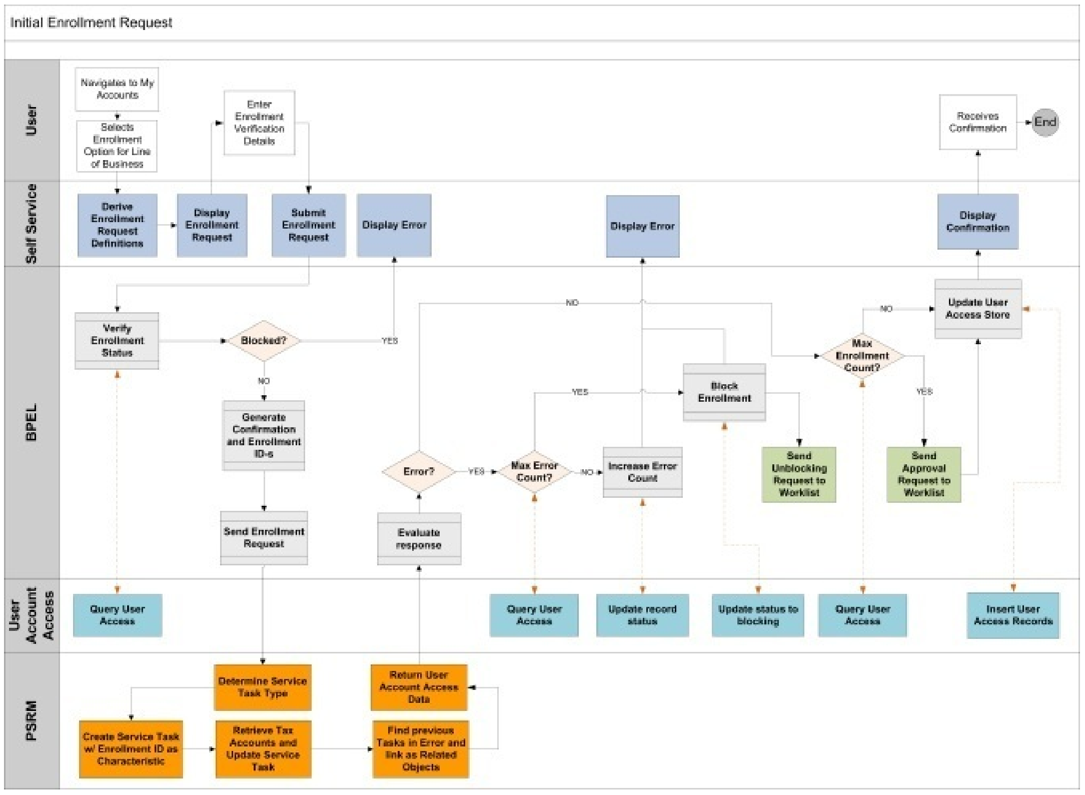
Enrollment Request

This special class of service request supports the ability for a user to request online access to multiple tax accounts and to perform various self-service operations.

One enrollment request is created per line of business.

NOTE: See the *Enrollment* section for more details about this functionality.

This is a special kind of a service request that defines its own web service, **TSEnrollmentServiceRequest**. The web service requesting the enrollment is processed by the XAI Inbound Service, which invokes the service script **C1-EnrSvcReq**. The appropriate service task type to use must be passed in with the other information received from the self service system.


Initial Enrollment Request
User
Self Service
BPEL
User Account Access
PSRM
Navigates to My Accounts
Selects Enrollment Option for Line of Business
Enter Enrollment Verification Details
Receives Confirmation
End
Derive Enrollment Request Definitions
Display Enrollment Request
Submit Enrollment Request
Display Error
Display Error
Display Confirmation
Verify Enrollment Status
Blocked?
YES
NO
Generate Confirmation and Enrollment ID-s
Send Enrollment Request
Evaluate response
Error?
YES
Max Error Count?
NO
YES
Increase Error Count
Block Enrollment
NO
Max Enrollment Count?
NO
YES
Update User Access Store
Send Unblocking Request to Worklist
Send Approval Request to Worklist
Query User Access
Query User Access
Update record status
Update status to blocking
Query User Access
Insert User Access Records
Determine Service Task Type
Return User Account Access Data
Create Service Task w/ Enrollment ID as Characteristic
Retrieve Tax Accounts and Update Service Task
Find previous Tasks in Error and link as Related Objects

The integration has the following steps:

- The self service application captures information about the taxpayer that is sufficient to identify tax accounts related to this taxpayer within the context of a specific line of business.
- The integration layer generates two unique IDs: **Confirmation ID**, which will be later presented to the self-service user, and **Enrollment ID**, which will be used internally between SOA/BPEL layer and the revenue management system.
- This information is sent to the system, which creates an Enrollment Request service task.
- The task's lifecycle is designed to process the information immediately, with no deferred monitor process, so that the information can be returned to the self service application in real-time.
- The base business object for the Enrollment Request Service Task includes an algorithm, **C1-ENR-INIT**, that reads enrollment service task type configurations and orchestrates the process. It invokes access type-specific logic to verify the taxpayer identity, determine what data this user should be able to access and manage, and populate the results on the response message. The response message contains the list of tax account identifiers (access type+access keys). Each entry also includes the enrollment status, revenue management system, and enrollment ID.

NOTE: See *Tax Account Access Information* for a detailed explanation of tax account access, and *Configuration Tasks for Enrollment Request* information about enrollment task type configurations.

- The response is sent back to the self-service application immediately.
- Enrollment entries are processed by SOA Composite in the integration layer. They are captured in the user access store table.
- The final response to the self-service application contains confirmation details.

Note that the implementation of the enrollment request creates a service task for the following reasons:

- The creation of a service task with its business object architecture provides many plug-in spots to allow implementations to easily customize the initial enrollment logic.
- The service task type provides a facility for the various configurations related to enrollment handling.
- The enrollment request service task provides an audit tracking for the enrollment activities and is linked to the unique enrollment ID which is shared between the revenue management system(s) and the self service product. It also allows tracking of failed enrollments attempts.
- The enrollment request service task captures the initial list of tax accounts and provides future reference for the implicit enrollment refresh.

  NOTE: See *Implicit Enrollment Refresh* for more information.

- Creating a service task enables the tax authority to provide the taxpayer with a confirmation ID for possible follow-up . If the taxpayer contacts the tax authority and provides the confirmation ID, the user can view the service task, verify the details, and more easily investigate the situation.

Determining the Tax Accounts Owned by User

The base product supports the enrollment for access type *Tax Role* out-of-box. It is implemented for two lines of business: *Individual* and *Business*.

Access keys used for *Tax Role* are:

PER_ID - taxpayer

TAX_ROLE_ID - tax role

NOTE: See *Message Processing Overview* chapter for more information about access type and line of business.

The enrollment logic depends on the requested line of business.

- **Individual** enrollment - the logic searches for accounts where the identified taxpayer is listed as either a primary taxpayer or a financially responsible person. Non-cancelled tax roles linked to these accounts are returned.
- **Business** enrollment - the logic reads the taxpayer instructions on the service task type and determines the relationships between the requestor and the business. It then determines the business person and finds all accounts where this person is either a primary taxpayer or a financially responsible person. The non-cancelled tax roles linked to these accounts are returned.

For example, the *Taxpayer Instruction* on the service task type lists *Corporate Officer* person-to-person relationship type means that the taxpayer(person) requesting an access to the business tax account should be linked to this business (person) as a corporate officer.

An algorithm plugged into the enrollment service task business object as an enter plug-in on the In Progress state is responsible for determining the tax accounts. Refer to the description of the algorithm for more information about the base logic provided.

See *Configuration Tasks for Enrollment Request* for information about configuring the system to support enrollment requests.

Configuration Tasks for Enrollment Request

The following sections list the configuration tasks required to implement the base taxpayer enrollment request functionality.

Characteristic Type

If your implementation captures the taxpayer's birth date on the person record and would like that to be part of the taxpayer identification for enrollment service requests, create a characteristic type for Birth Date. It should be defined as ad-hoc and should reference Person as the characteristic entity.

Notes:

- It is recommended to use the same characteristic type as the one defined for *Taxpayer Identification Service Request.*
- The product does not provide any functionality for populating the birth date characteristic on the person.

Service Task Type

Create a service task type for the enrollment request service task for each line of business your implementation will support. Using the business object **User Enrollment Self Service Request Type**:

- Select the line of business from the drop-down list.
- The related transaction BO should be set as follows:
  - Line of Business ***Business*** - C1-BusinessEnrollmentRequest.
  - Line of Business ***Individuals*** - C1-IndividualEnrollmentRequest
- Service task class should be set to Service Request.
- To Do Type should be set to a standard error To Do type to use when transitioning to error. The base product To Do Type Self Service Task Issues (C1-SSTTD) is provided for this purpose.
- To Do Role should be configured to an appropriate default To Do role for the above To Do type. This is optional. If no value is provided the default role for the To Do type is used.
- Configure the Confirmation Header Message Category and Message Number and the Error Header Message Category and Message Number to use for communication to the self service user.

  NOTE: See the Messages section of the *General Configuration Tasks* for more information.

- The service request fields mapping is used when creating the target service task to correctly populate the requestField list from the list of fields passed in on the web service request XML. For the base transaction business object, the following fields are expected:

Sequence	Transaction BO XPath
1	requestData/idType
2	requestData/idValue
3	requestData/birthDate

- **Enrollment Instructions**:
  - Char Type for Date of Birth - Specify the characteristic type defined above

- **Name Type** - Specify what type of taxpayer's name should be displayed on enrollment summary. This configuration is used by enrollment summary retrievers.

  NOTE: See *Enrollment Summary* for more information.

- **Enrollment status** - Specify the default status of the user access records. Permissible values are *Approved*, *Pending* and *On Hold*.

Business

- **Taxpayer Instructions:** List person-to-person relationships that may link a user requesting an access to the tax account owner. In other words, define who may be granted an online access to the account: business owner, corporate officer, legislative advisor. List all relationships that should be considered by enrollment logic.
- **Access Type Processors:**

  If your implementation wish to support user account access on the tax role level provided in the base product, configure this section as follows:

  - Select an access type Tax Role.
  - Specify the business service **C1-EnrolIndividualAccntTaxRole** as a User Enrollment Retriever.
  - Specify the business service **C1-GetBusinessTaxRoleSummary** as an Access Type Summary Retriever.

Individuals

- **Taxpayer Instructions:** if your implementation wish to utilize User Summary Retriever included in the base product, skip this section. The individual tax account determination logic is not considering these instructions.
- **Access Type Processors:**

  If your implementation wish to support user account access on the tax role level provided in the base product configure this section as follows:

  - Select an access type Tax Role.
  - Specify the business service **C1-EnrolIndividualAccntTaxRole** as a User Enrollment Retriever.
  - Specify the business service **C1-GetIndividualTaxRoleSummary** as an Access Type Summary Retriever.

  NOTE: Service Task Type Mapping. The service task type defined here must be configured in the service task domain value mapping in the SOA Composer application. Refer to the self service product documentation for details.

Implementing a New Service Request

Your implementation may wish to provide support for additional service requests for your taxpayers. The following information outlines the steps required in the product to support a new service request.

Taxpayer Identification

First decide if the taxpayer must provide identification information as part of the service request. If so,

- Should the system validate the taxpayer before continuing with the actual request? In this case an initial web service call must be sent to the system with appropriate information to validate the taxpayer.
- Can the identification information be provided with the other service request details so that the taxpayer identification can be done with the service request processing?

If the taxpayer identification is done as a separate step, refer to *Configuration Tasks for Taxpayer Service Request* for the base taxpayer identification business object provided with the product. That may be used or a new one based on appropriate business rules may be introduced with this one as a sample.

Business Object

Each service request must have its own Service Task business object. This business object defines the information required for this type of request and defines the algorithms that validate and process the request. The product provides several business objects that may be used as a parent business object for a new service request.

- The Standard Deferred Self Service Task business object (**C1-StandardDeferredSSTask**) supports the ability to defer the full processing of the request to a subsequent step. Defining a service task business object as a child of this business object is recommended if any of the following are true:
  - A user needs to review the taxpayer's request.
  - The request is high volume such that processing the request real-time for many taxpayers may cause a lot of system traffic.
  - The logic required to process the request is such that it doesn't make sense for the taxpayer to wait until the processing is complete.
- The Standard Real Time Self Service Task business object (**C1-StandardRealTimeSSTask**) supports immediate processing of the service request to give a response to the taxpayer real time.
- The Standard Scheduled Reminder Self Service Task business object (C1-ScheduledReminderSSTask) supports the repeatable creation of the reminder, maintaining the reminder task in an active status and expire it at the specific date. Defining a service task business object as a child of this business object is recommended if any of the following are true:
  - A periodic reminder should be issued.
  - The date of the reminder may be derived from the request data or calculated based on some conditions.

Create the new business object defining the appropriate parent business object and navigate to the schema tab. Include the parent BO's schema. Define the additional elements required for this business object's functionality. Refer to the Tax Clearance Certificate business object for an example.

The business object must also include the appropriate algorithms to satisfy the request.

Service Task Type

Create a service task type for the new service request with the appropriate service task type business object; base product provides business objects for **Taxpayer Service Request Self Service Task Type**, **Generic Create To Do Request Task Type** and **Scheduled Reminder Request Task Type**.

- The related transaction BO should be set to the business object created above.
- Service task class should be set to **Service Request**.
- To Do Type should be set to a standard error To Do type to use when transitioning to error. The base product To Do Type **Self Service Task Issues (C1-SSTTD)** is provided for this purpose.
- To Do Role should be configured to an appropriate default To Do role for the above To Do type. This is optional. If no value is provided the default role for the To Do type is used.
- Configure the Confirmation Header Message Category and Message Number and the Error Header Message Category and Message Number to use for communication to the self service user.

  FASTPATH: Refer to the Messages section of the *General Configuration Tasks* for more information.

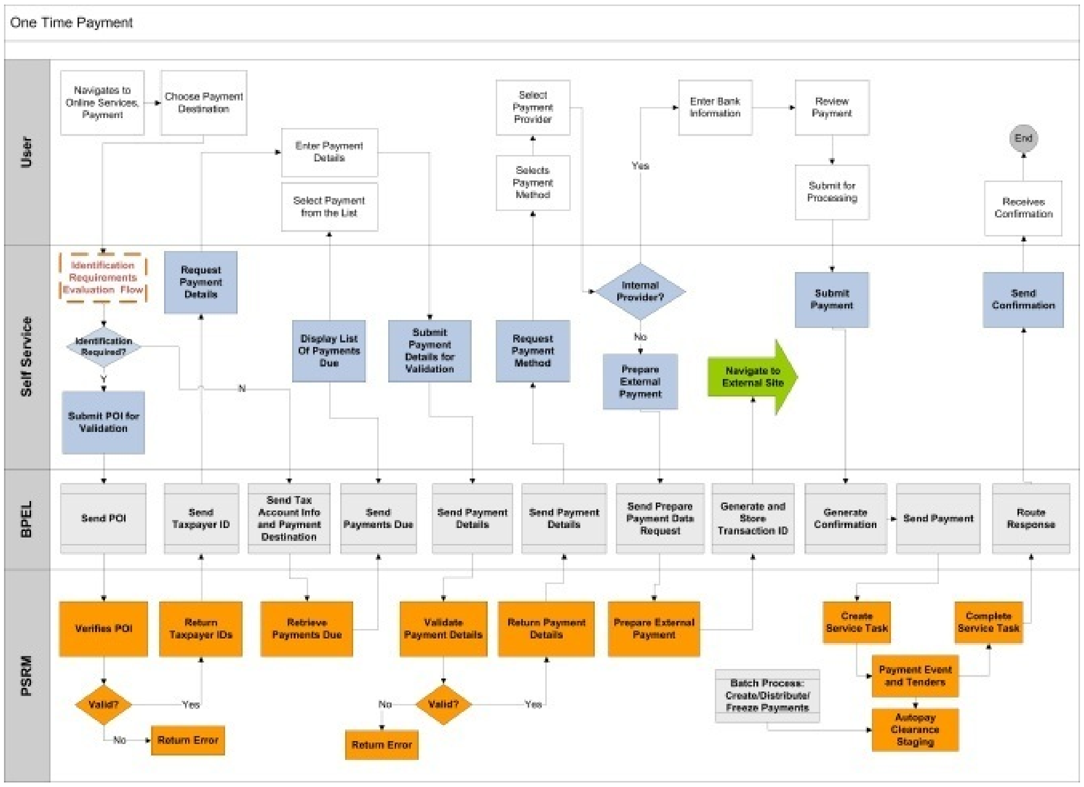
- The service request fields mapping is used when creating the target service task to correctly populate the request data list from the list of fields passed in on the web service. Define the mappings appropriate for this service request as needed.

- Add request-specific configurations where needed.

FASTPATH: Additional Configuration is required in the self service product and in the integration layer for new service requests. Refer to the self service product documentation for more information.

Online Payment

In this release, the integration supports payment by a taxpayer using a bank or credit card number. The following diagram illustrates an expected process flow.



The integration has the following steps:

- Initial processing works as follows:
  - **Casual user**. The self service application captures information about the taxpayer and sends a web service request to this product to validate the taxpayer (proof of identity, or POI).
  - **Enrolled user**. The identification is not performed.

  FASTPATH: Refer to the *User Identity Verification* topic for more information.

- Once identified, the casual user indicates what the payment is for. This is referred to as the target of the payment or the payment destination or the payment details. For example, is this a payment for a pay plan scheduled payment? If so, the taxpayer must provide the pay plan identifier. Once the taxpayer indicates the payment target details, another web service request is sent to this product to verify the details.
- When the enrolled user is browsing a specific tax account it becomes possible to retrieve payments that are currently due and present user with the pre-populated list so the user will choose what to pay rather than enter payment details manually.
- Next, the taxpayer indicates the method of payment:

- If the method is via a credit card and the implementation uses an external payment vendor, the self service application sends a web service request to this product to retrieve any additional information about the taxpayer that may need to be sent to the external vendor (such as the taxpayer's address) and to determine if the external vendor's fee should be passed on to the taxpayer. At that point the taxpayer is taken to the external vendor's website to further process the payment details. Once that step is complete a final web service request is sent to this product to process the payment.
- If the method is via a bank account or if the implementation does not use an external payment vendor for credit card payments, the self service application sends a web service to this product to create the payment. The standard Auto Pay logic in the product is used to process the payment.

- If the external vendor supports payment reconciliation, the subsequent reconciliation report is processed by the product.

The topics below describe more information about the provided functionality, followed by configuration tasks.

Identifying the Taxpayer

In this release the integration supports payments by an 'unregistered' taxpayer. The taxpayer does not need to log in using a user name and password in order to submit a payment. In addition, the product supports a person paying on behalf of another taxpayer.

The taxpayer must provide enough information to be identified by the system prior to proceeding. A special service task business object is provided for identifying a taxpayer for one-time payments (**C1-TaxpayerIdentifySSTask**).

FASTPATH:
Refer to *Configuration Tasks for Online Payment* for more information.

Payment Destination

When creating a payment, the product supports using distribution rules to determine where the payment should be targeted. The integration provides out of the box support for the following payment destinations.

- **Pay a filing period**. This method allows the taxpayer to provide their taxpayer ID, a tax type and a filing period to direct the payment to.
- **Pay a pay plan**. This method allows the taxpayer to provide a reference to a pay plan to pay one or more of the scheduled payments.
- **Pay a collection notice**. This method assumes that a collection notice has been issued from a customer contact in the system and that the customer contact ID has been provided to the taxpayer as an identifier.
- **Pay a Form (DLN Only)**. This method assumes that a tax or business registration form was processed by the system and the Document Locator Number has been provided to the taxpayer.
- **Pay an Obligation**. This method allows taxpayer to provide the exact ID of the obligation and pay obligation's balance.
- **Pay a Tax Role Balance**. This method allows taxpayer to provide the exact tax role ID and pay an outstanding tax role balance.
- **Pay a Tax Form**. This method allows the taxpayer to provide a DLN of the form that is not necessarily exists in the system yet. The assumption is that this method would be used for payments immediately following the on-line form submission for the scenario where the form submission web service request is queued by the integration while the payment request is processed immediately.

The business object provided by the product for processing one time payments has been designed to support different payment destinations. To do this, the business object must support receiving different types of information (for example, pay plan ID or collection notice ID) and use plug-ins that are specific to each payment destination that knows the type of data expected, how to use that data to identify the correct account and obligation(s) and process the payment successfully. At a high level it works as follows:

- The web service requests sent by the self service application to validate the payment target and to process the payment include an indication of the payment destination and a field/value pair collection to hold the values that represent the 'target' for the payment.
- The self service master configuration record includes a mapping between the payment destination and a self service task type.
- The service task type for each unique payment destination references a "processor" service, which can be a service script or a business service. This service should be specific for a given payment destination and expects the data appropriate for it. For example, the process to "pay a pay plan" expects the service task's destination details to specify a pay plan ID.
- The service task type includes a field mapping section. This mapping indicates which field value in the destination details list to populate in which target XPath in the "processor" service.
- The web service called to validate the destination details provided by the taxpayer invokes the "processor" to ensure that the data can be found.
- When processing the payment, the service task that processes payments includes an algorithm that uses the configuration on the task type to call the "processor" service, passing the details of the payment target.

FASTPATH: Refer to *Configuration Tasks for Online Payment* for information about configuring the system to support these payment destinations. Refer to *Implementing New Payment Destinations* for information about defining additional payment destinations for your implementation.

Account Verification Service

The processor is used to verify if the input account is valid for the entity identified by the current access information. If the account is invalid it returns an error.

When creating a new custom Processor, use any of the base business services or service scripts provided as an example of the type of functionality needed. All base components provided begin with **C1-OTPVAc%**. The processor may be implemented as a service script or a business service.

The custom service, whether it be a service script or a business service should include the data area **C1-SelfServiceKeys** along with an **<accountId>** element

Retrieving Payments Due

The web service used to retrieve the list of outstanding payments is processed by the XAI Inbound Service TSRetrievePaymentsDue, which invokes the service script **C1-RtPymtDue**.

The web service request contains:

- Access info: access type and access keys – Identifies the enrollment unit, for example Tax Role, Account or others
- Payment destination defines what type of payment is expected; for example, collection notice, obligation, etc.

The service script determines the service task type based on the payment destination (via master configuration). It then calls the retrieve payments due processor. The processor should return a collection of the prospective payments. Each entry contains:

- Payment amount due.
- Payment destination fields collection.
- Parameters collection used by self-service system to display the formatted info about this payment. Parameters are injected into the message defined in the self-service Payment Destination admin configuration.

The parameters for the obligation info display are obligation type and payment due date.

NOTE: Integration Mapping should be completed for proper display of obligation information.

Validating the Payment Destination

The web service used to validate the payment destination information entered by the taxpayer is processed by the XAI Inbound Service **TSPrepareExtPaymentData**, which invokes the service script **C1-PrExPyDta** with an action of **VALIDATEONLY**. The service script determines the service task type based on the payment destination (via master configuration). It then calls the payment destination details processor with an action of "validate only". The processor should return an error if the information provided by the taxpayer does not identify an appropriate related object for the payment destination.

If the enrolled user is making a payment, the service script calls an additional processor that checks if the account determined during the destination details validation is relevant for the tax account in context (access type+ access keys).

Creating the Appropriate Service Task

The web service request to create a payment is processed by the XAI Inbound Service **TSOneTimePayment**, which invokes the service script **C1-PyUnrgUsr**. The service script is responsible for creating a Service Task of the correct service task type based on the payment destination.

Processing Auto Pay Payments In the System

When a taxpayer provides bank account information for payment in self service, the payment detail is sent to this system and the integration with the customer's bank is handled through standard auto pay processing. This method may also be used to process credit cards. However, it's expected that an integration with an external vendor will most often be used for that. Refer to the section below for more information on external payment vendors.

Requiring an Auto Payment Agreement

Many tax authorities require taxpayers to sign an automatic payment agreement prior to allowing them to submit tax payments online. The product supports checking for a reference to an ACH agreement that is defined as a characteristic linked to the taxpayer's record.

Payment Processing Lifecycle

The one-time payment service task provided by the base product includes the following basic lifecycle states:

- **Pending**. When the web service request is processed, a service task is created in pending status. Initial validation is performed, including verifying that the information provided for the payment destination and determining the appropriate account to associate with the payment. If any issues are found, the service task is not stored and an error is returned. Once the record is validated, no further processing occurs at this time. When the service task monitor is run all pending records are further processed.
- **In Progress**. When transitioning to this status, the payment is processed. The data related to the payment destination is mapped to appropriate payment event distribution details and the payment is created. If any errors are encountered, the service task transitions to **Error**. If the payment was processed through an external vendor that requires reconciliation, it transitions to **Pending Reconciliation**. Otherwise, it transitions to **Complete**.
- **Error**. Any errors detected in payment processing or during reconciliation processing causes the service task to transition to this state and creates a To Do entry. A user must review and resolve the problem.
- **Error Resolved**. When a user resolves the error reported the service task can be transitioned to this state. Note that the assumption is that the user has manually corrected the error. This state does not cause any logic to occur.
- **Pending Reconciliation**. When a payment is made through an external vendor that requires reconciliation, the service task transitions to this state and waits for the reconciliation process. Refer to the external payment vendor section below for more information about reconciliation. Note that this state is configured with time out logic. The payment vendor can be configured with a number of days to wait for the reconciliation report. If that number of days passes, a To Do entry is created using an appropriate To Do type defined for the payment vendor.

- **Complete**. Service tasks transition to this state when the payment is fully processed.

External Payment Vendor

Many websites integrate with an external payment vender to process credit cards real-time. In this case, the following functionality is provided:

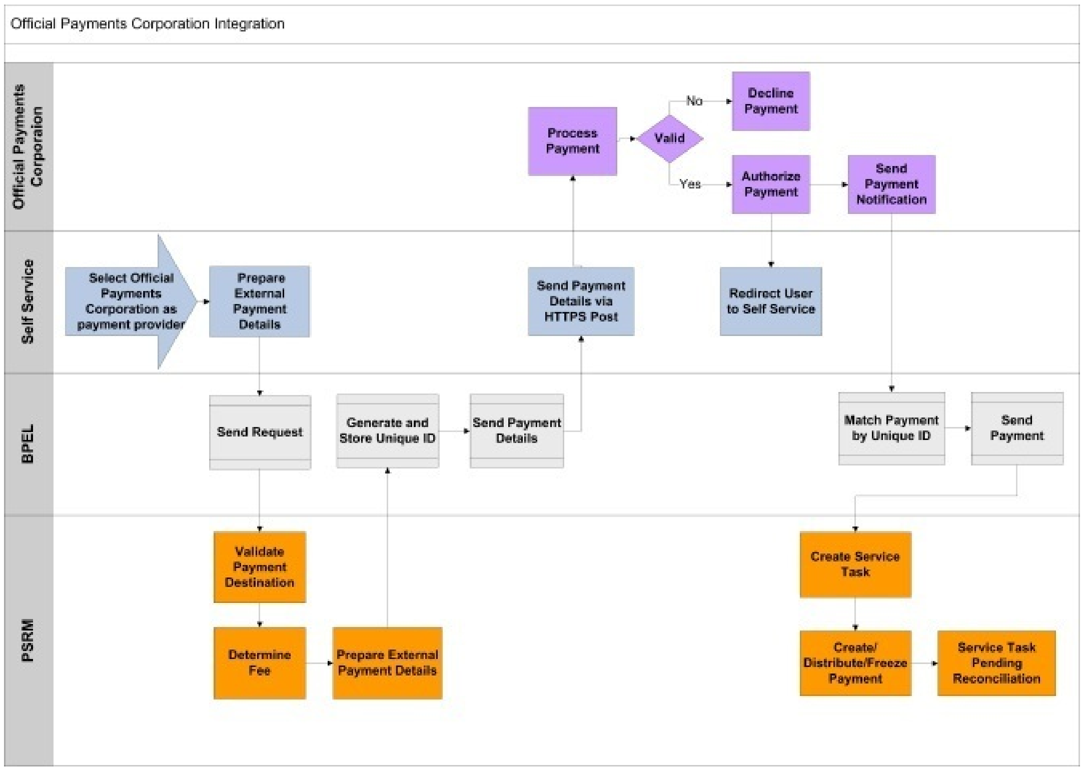
- Preparing External Payment Data. After the standard steps of identifying the taxpayer and validating the payment destination, when a taxpayer indicates payment using a credit card, the self service system sends a web service request to this product to determine whether a fee is applicable and to return additional payment data.
- Payment processing. Once the credit card information is processed by the external vendor, a web service request with the payment information is sent to this system. This causes an appropriate service task to be created to process the payment and apply it to the taxpayer's balance appropriately.
- Reconciliation. External vendors may supply reconciliation information for all the payments that were processed in a given time period. The product supports the ability to receive the reconciliation information, compare it against our records and highlight any discrepancies.

NOTE: Integration with Official Payments Corporation (OPC). The product provides functionality to integrate with OPC as an external vendor. The logic provided in the base product is based on that vendor's requirements.

Refer to the self service documentation for more information about configuring the system to work with external vendors.

The following topics provide additional detail about the integration functionality provided.

The following diagram illustrates the prepare external payment data and payment processing components for the integration with OPC.


Official Payments Corporation Integration
Official Payments Corporaion
Self Service
BPEL
PSRM
Process Payment
Valid
No
Yes
Decline Payment
Authorize Payment
Send Payment Notification
Select Official Payments Corporation as payment provider
Prepare External Payment Details
Send Payment Details via HTTPS Post
Redirect User to Self Service
Send Request
Generate and Store Unique ID
Send Payment Details
Match Payment by Unique ID
Send Payment
Validate Payment Destination
Determine Fee
Prepare External Payment Details
Create Service Task
Create/ Distribute/Freeze Payment
Service Task Pending Reconciliation

Prepare External Payment Data

Once the taxpayer has indicated that payment will be made by credit card, the self service application sends a web service request to determine the fee requirement and to retrieve detail about the taxpayer to provide to the external vendor. These are processed by the XAI Inbound Service **TSPrepareExtPaymentData**, with an action of **PREPARE**, which invokes the service script **C1-PrExPyDta**. This script does the following:

- It uses the payment destination to determine the appropriate service task type (via the master configuration) and uses this information to determine the fee requirements. External vendors often charge fees for using their service. The tax authority may to choose to pass the fee onto the taxpayer or to absorb the cost of the fee. The product supplies configuration to allow the tax authority to define for each service task type whether taxpayers should be charged the fee based on the person's person type.

NOTE: Refer to the Service Task Types section of *Configuration Tasks for Online Payments* for more information.

- It retrieves additional information about the taxpayer, if required by the external vendor. It looks for a Prepare Data service script linked to the external vendor's record. If one is defined it is called. Refer to *Configuration Tasks for Payment Vendors* for more information.

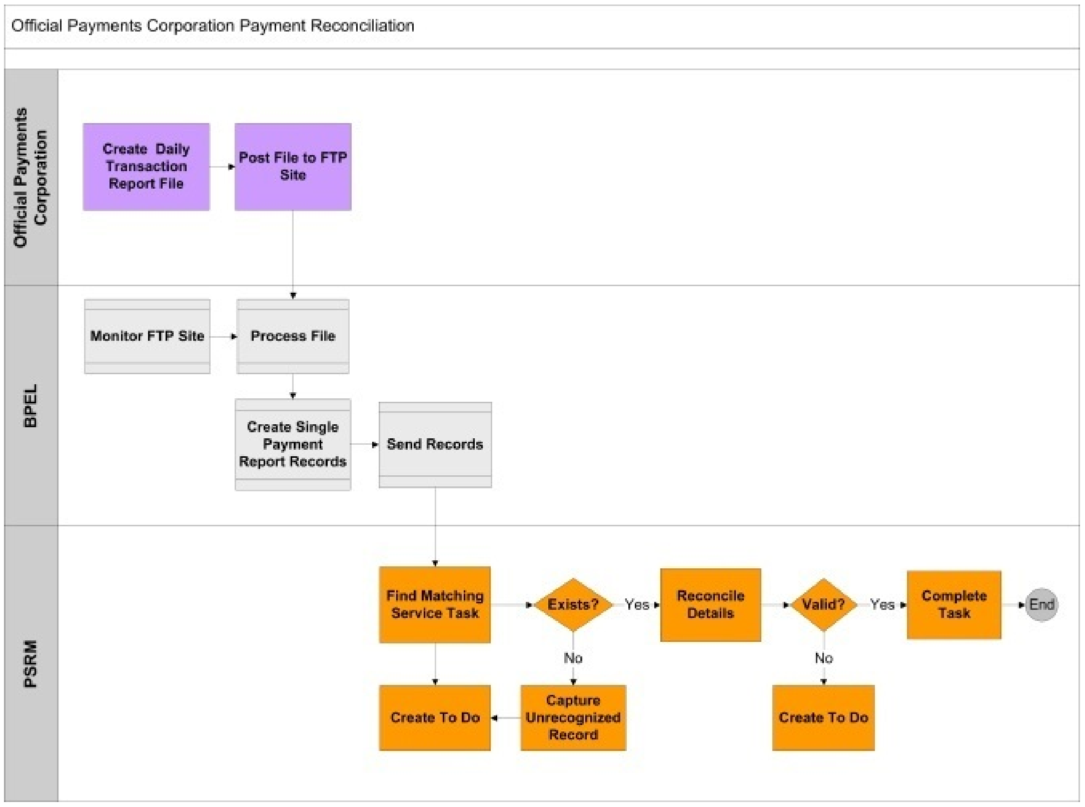
Payment Processing

For the most part, the payment processing for these payments is analogous to the processing done for bank payments. The tender type used should be one that is not configured to integrate bank details. And an external reference to the payment vendor's record should be recorded with the service task using a characteristic. This allows the reconciliation process to identify the record.

The product supports capturing additional information as defined by the requirements for the external vendor. The data is stored in a "raw" element on the service task called externalPaymentData. The structure of the information captured must be defined using the External Payment Data Area linked to the payment vendor lookup.

Reconciliation

The following diagram illustrates the reconciliation component for the integration with OPC.


Official Payments Corporation Payment Reconciliation
Official Payments Corporation
Create Daily Transaction Report File
Post File to FTP Site
BPEL
Monitor FTP Site
Process File
Create Single Payment Report Records
Send Records
PSRM
Find Matching Service Task
Exists?
Yes
No
Reconcile Details
Valid?
Yes
No
Complete Task
End
Create To Do
Capture Unrecognized Record
Create To Do

Once the service task successfully creates the payment, it checks whether reconciliation with the payment vendor is applicable and if so, it transitions to the **Pending Reconciliation** state to wait for the reconciliation details.

When the reconciliation details are processed, individual web service calls are sent to the system for each payment. These are processed by the XAI Inbound Service **TSProcessExtPayReportRecord**, which invokes the service script **C1-PrcPyRpRc**. This script uses the external reference to find an existing payment service task in the **Pending Reconciliation** state with this reference linked as a characteristic.

- If one is found, it is updated with a snapshot of the payment report from the vendor. The payment report data is stored on the service task in a "raw" element called externalPaymentReport. The structure of the information captured must be defined using the External Payment Report Data Area linked to the payment vendor lookup. In addition, it is marked as ready to reconcile. The next time the reconciliation monitor runs the service task's reconciliation algorithm is executed. The algorithm is configured to call logic that is specific for the payment vendor. It calls the Payment Reconciliation Script defined on the payment vendor lookup. If there are any issues it transitions to **Error**, otherwise it transitions to **Complete**.
- If one is not found, the system creates a new service task record using a special task type configured for logging an unrecognized external payment in the payment report. The service task BO provided by the product for this scenario **C1-UnrecognizedExtPymtRptTsk** provides a simple lifecycle that creates the record in the **Pending** state. The next time the service task deferred monitor runs, the record creates a To Do entry for a user to research and resolve the issue and sets this record to **Complete**.

FASTPATH:
Refer to *Configuration Tasks for Payment Vendors* for more information.

Configuration Tasks for Online Payments

The following sections list the configuration tasks needed to implement the online payment functionality.

Standard Payment and Automatic Payment Options

When the one-time payment service task processes the payments it relies on standard payment logic in the product. Refer to the payment configuration documentation for configuration options required for general payment processing as well as configuration options for automatic payment processing.

FASTPATH: Refer to the Defining Financial Transaction Options > Managing Payment Setup and Automatic Payment Options in the administration guide for more information.

Obligation Type for Excess Credit

The base product payment distribution rule algorithms that create payments support the ability to direct excess credit to a special obligation rather than crediting the obligation used to levy the taxes. If your implementation uses the base product algorithms, define an obligation type to use when directing excess credit to a special obligation for the account.

Suspense Obligation

In some cases when an account cannot be found for directing a payment, a payment is created in suspense, meaning that it is created for a suspense obligation. If your implementation does not already have a suspense obligation set up, create one.

Match Types

Define a match type for Obligation. This is used by the Pay a Pay Plan distribution rule.

Algorithms

Create a Distribution Rule - Create Payment algorithm for each supported payment destination.

- **Pay a Filing Period**. If your implementation supports the ability to direct a payment to a filing period, define an algorithm for the algorithm type **C1-PAYFRM**. In the parameters define the division / obligation type for the excess credit obligation along with the suspense obligation. Finally, define characteristic types for each of the payment details that are required by the algorithm. Note that the product supplies characteristic types that may be used for each one.
  - C1-DLN for document locator number. Note that this is not expected to be supplied by a taxpayer for web self service payments, but the parameter is required for the algorithm.
  - C1-TXPID for taxpayer ID
  - C1-TAXTY for tax type
  - C1-FLNGP for filing period
- **Pay a Collection Notice**. If your implementation supports the ability to direct a payment to a collection notice, define an algorithm for the algorithm type **C1-DR-PAYCC**. In the parameters define the division / obligation type for the excess credit obligation. In addition, define the characteristic type that the overdue process uses to log the creation of the customer contact. This allows the algorithm to determine the related overdue process to find the covered obligations.
- **Pay a Pay Plan**. If your implementation supports the ability to direct a payment to a pay plan, define an algorithm for the algorithm type **C1-DR-PAYPP**. In the parameters define the division / obligation type for the excess credit obligation. In addition, define the match type for referencing the pay plan's obligation.
- **Pay a Tax Role Balance**. If your implementation supports the ability to direct a payment to a tax role balance, define an algorithm for the algorithm type C1-DR-PAYTXR. In the parameters define the division / obligation type for the excess credit obligation.
- **Pay an Obligation**. If your implementation supports the ability to direct a payment to an obligation, you may use an algorithm C1-DR-PAYOBL delivered with the product.

Distribution Rules

Create a distribution rule for each supported payment destination.

- **Pay a Filing Period**. If your implementation supports the ability to direct a payment to a filing period, multiple distribution rules must be created, one for Taxpayer ID, one for Tax Type and one for Filing Period. Each should reference the appropriate characteristic type. Refer to the create payment algorithm above for details of the characteristic types to use. Note the following points related to configuring these distribution rules:
  - The payment distribution processing expects that the create payment algorithm is only linked to the last distribution rule.
  - The payment distribution processing expects that the Determine Tender Account is linked to all the distribution rules.
  - The payment details processor business service used by the payment service task may expect the definition of the distribution rules to be in a certain order. Refer to the business service description for **C1-PaymentDestinationTxTyFilPd** for more information.
- **Pay a Collection Notice**. If your implementation supports the ability to direct a payment to a collection notice, create a distribution rule referencing the characteristic type **C1-CUSCN** Customer Contact. It should also reference the Create Payment algorithm created above for paying a collection notice.
- **Pay a Pay Plan**. If your implementation supports the ability to direct a payment to a pay plan, create a distribution rule referencing the characteristic type **C1-PAYPL** Pay Plan. It should also reference the Create Payment algorithm created above for paying a pay plan.

- **Pay a Form (DLN Only)**. If your implementation supports the ability to direct a payment to a form using DLN only, create a distribution rule referencing the characteristic type C1-DLN GDocument Locator. It should also reference the Create Payment Algorithm created above for paying a filing period.
- **Pay an Obligation**. If your implementation supports the ability to direct a payment to an obligation, create a distribution rule referencing the characteristic type C1-OBLIG GObligation. It should also reference the Create Payment algorithm created above for paying an obligation.
- **Pay a Tax Role Balance**. If your implementation supports the ability to direct a payment to a tax role balance, create a distribution rule referencing the characteristic type C1-TAXRL Tax Role. It should also reference the Create Payment Algorithm created above for paying a tax role balance.

Business Object

Decide whether or not an email confirmation should be sent to the taxpayer once the payment is processed. If so, add an Enter plug-in to the In Progress state after the payment creation algorithm or perhaps the Complete state to generate the confirmation email.

FASTPATH: Refer to the algorithm section of the *General Configuration Tasks* for more information about configuring the confirmation algorithm.

Service Task Types

Taxpayer Identification Task Type

Define a task type to use for taxpayer identification with the business object **Taxpayer Service Request Self Service Task Type**.

- The Related Transaction BO should be set to **C1-TaxpayerIdentifySSTask**.
- Service Task Class should be set to **Taxpayer Identification**.
- To Do Type should be set to a standard error To Do type to use when transitioning to error. The base product To Do Type **Self Service Task Issues (C1-SSTTD)** is provided for this purpose. Note that the algorithm plugged in on the error state for the base business object checks that there is not already a To Do for the record before creating one of this type. This allows algorithms that detect an error to also create a To Do with a specific type if desired.
- To Do Role should be configured to an appropriate default To Do role for the above To Do type. This is optional. If no value is provided the default role for the To Do type is used.
- Configure the Confirmation Header Message Category and Message Number and the Error Header Message Category and Message Number to use for communication to the self service user.

  FASTPATH: Refer to the Messages section of the *General Configuration Tasks* for more information.

- The service request fields mapping is used when creating the target service task to correctly populate the requestField list from the list of fields passed in on the web service request XML. For the base transaction business object, the following fields are expected:

Sequence	Transaction BO XPath
1	requestData/legalName
2	requestData/idType
3	requestData/personIdNumber
4	requestData/emailAdd
5	requestData/addressLine1
6	requestData/addressLine2

7	requestData/city
8	requestData/county
9	requestData/state
10	requestData/postal
11	requestData/onBehalfOfFlag
12	requestData/secondaryLegalName
13	requestData/secondaryIdType
14	requestData/secondaryPersonIdNumber
15	requestData/secondaryEmailAdd

FASTPATH: Service Task Type Mapping. The service task type defined here must be configured in the service task domain value mapping in the SOA Composer application. Refer to the self service product documentation for details.

Payment Destination Task Types

Create a service task type for each payment destination with the business object **One-Time Payment Self Service Task Type**. Each service task configured should include the following common configuration.

- The Related Transaction BO should be set to **C1-StandardOneTimePaySSTask**.
- Service Task Class should be set to **Self Service Payment**.
- To Do Type should be set to a standard error To Do type to use when transitioning to error. The base product To Do Type **Self Service Task Issues (C1-SSTTD)** is provided for this purpose. Note that the algorithm plugged in on the error state for the base business object checks that there is not already a To Do for the record before creating one of this type. This allows algorithms that detect an error to also create a To Do with a specific type if desired.
- To Do Role should be configured to an appropriate default To Do role for the above To Do type. This is optional. If no value is provided the default role for the To Do type is used.
- Configure the Confirmation Header Message Category and Message Number and the Error Header Message Category and Message Number to use for communication to the self service user.

  FASTPATH: Refer to the Messages section of the *General Configuration Tasks* for more information.

The following configuration differs based on the payment destination:

- Payment Distribution Rules. Link the appropriate distribution rule(s) created above for the payment destination represented by this service task type.
- Link an appropriate Processor business service or service script that knows how to receive the detail related to the payment destination and create the payment event distribution details correctly. The base product provides one Processor business service for each supported payment destination. Using the search look for business services that begin with **C1-PaymentDestination%**.
- Specify if taxpayer should exist in the system in order to process the payment. Typically this should be set to **Y**es ("Y") as the taxpayer record is necessary for account and other payment details verification. An example of the exception from this rule is a payment for a form by DLN. When payment is submitted immediately following online form filing the system should be able to accept payment from the self-service user who is a first-time filer. In this scenario the taxpayer record is created during form posting and that may actually happen after payment submission.
- Field mappings. For each payment destination detail captured on the service task, indicate the target XPath value in the above processor.

NOTE: Special note on payment destination Tax Form (C1OF): This payment destination represents a payment for a form that was submitted online but has not necessarily reached the system at time of payment. For a service task type for this payment destination use processor ***C1-PaymentDestOnlineFormFiling*** and the distribution rule **Pay a Form (DLN only)** created above.

Configure the appropriate Processors for access type-specific logic. For each access type:

- **Account Verification**. This processor verifies whether account determined based on payment destination details is valid for the service request access information. This verification is optional and should be performed according to the business requirements. Account verification is needed if the payment has to be made within the context of the current access entity.

  Consider the following example: a self-service user is currently browsing a tax role and attempts to pay a collection notice. Configure account verification for access type **Tax Role** if you want user to be able to pay only for those collection notices associated with the tax role in context. Using the search look for business services or service scripts that begin with **C1-OTPVAc%**.

- **Payments Due**. This processor retrieves all pending items belonging to a payment destination within the context of the current access entity. For example, for payment destination Pay an Obligation and access type **Tax Role** it may retrieve all obligations with an outstanding balance. Using the search, look up business services or service scripts that begin with **C1-RetPaymentsDue%**.

  The examples of payments due retrievers provided out of the box:

  - ***Pay an Obligation*** and the access type ***Tax Role***. It retrieves all the obligations with an outstanding balance linked to the tax role identified by access key 2). If your implementation supports this payment destination, specify **C1-RetPymtsDueObligationTxRole**.
  - ***Pay a Tax Role Balance*** and the access type ***Tax Role***. It retrieves a single entry for an outstanding balance for the tax role identified by access key 2). If your implementation supports this payment destination, specify **C1-RetPymtsDueTaxRoleBalance**.

Finally, if external vendors are supported, indicate whether or not fees are appropriate based on the taxpayer type.

FASTPATH: Refer to *Implementing A Fee Requirement Script* for information.

NOTE:
The product supports two distinct methods of paying a form by DLN:

- Payment immediately following the online form filing, supporting first-time filers. For this task type, specify that the taxpayer does not have to exist in the system.
- Payment is made by a registered taxpayer for an existing form.

Master Configuration

The self service master configuration record includes several settings required for payment related processing.

- Char Type for Overdue Process Log. If your implementation supports the Pay a Collection Notice payment destination, define the characteristic type that is used to link the customer contact to the overdue process. The base product provides a value that may be used.
- Refer to the ACH Agreement Validation section below for information about the Char Type for ACH Agreement Verification.
- Tender Types. Indicate the tender types that are valid for paying online.
- Supported Payment Destinations. For each supported payment destination (as defined in the **C1_PAYMENT_TARGET_FLG** lookup), indicate the corresponding Service Task Type.

- Define the customer contact types that represent the collection notices that a taxpayer may pay online using the Pay a Collection Notice payment destination.

ACH Agreement Validation

If your implementation requires a pre-signed agreement from the taxpayer when submitting a payment through their bank account, define a characteristic type on the taxpayer, the following configuration is needed:

- Create an appropriate characteristic type for **ACH Agreement**. Define a valid characteristic entity of **Person**.
- Update the master configuration to indicate this characteristic type.
- Update the business object **C1-StandardOneTimePaySSTask** to plug in the validation algorithm to check for the existence of this characteristic. Navigate to the **Lifecycle** tab and expand the configuration for the **Pending** state. Add an entry in the Algorithms collection with a System Event of **Enter**, an appropriate sequence and the algorithm **C1-CHKACHAGR**.

NOTE: The product does not provide any functionality for populating the characteristic on the taxpayer record. This functionality is custom and must be provided by your implementation.

Integration Mapping

Obligation types used by the product should have corresponding values defined in the self service product.

Your implementation may decide to use exactly the same values in both products or configure the mapping using domain value mapping in the SOA Composer application

FASTPATH: Refer to the self service product documentation for details.

Configuration Tasks for Payment Vendors

The following sections list the additional configuration tasks needed to support an external payment vendor.

External Payment Report Task Type

Create a service task type to use when a payment in the reconciliation report for an external vendor is not found.

- The service task type BO should be **C1-StandardSelfServiceTaskType**
- The related transaction BO should be set to **C1-UnrecognizedExtPymtRptTsk**.
- To Do Type is not applicable. The service task BO has an algorithm that creates a To Do based using the unrecognized payment To Do Type on the payment vendor lookup.
- To Do Role is not applicable
- Confirmation Header Message Category and Message Number and Error Header Message Category and Message Number are not applicable.

External Vendor Extendable Lookup

Create a payment vendor extendable lookup record for the external vendor that is used by the self service application.

- If additional information is required by the external vendor to process the payment (such as the taxpayer's address), indicate an appropriate **Prepare Payment Data Script** that populates the data. The script should use the data area **C1-ExternalPaymentDetails** as its schema. The data area defines a paymentDetails list, which is a collection of field name and value pairs, to capture the additional information.

NOTE: Integration with Official Payments. The product provides a service script **C1-BldExPyDt** that builds the additional data required for integration with Official Payments.

- If additional information is included in the payment processed by the external vendor, indicate the **External Payment Data Area** that describes the structure of the data. This information is captured in the service task created to process the payment in the raw element externalPaymentData.

NOTE: Integration with Official Payments. The product provides the data area **C1-OPCExtPaymentData** that supports the additional data sent with payments made via integration with Official Payments.

If the vendor supports a reconciliation report, check the **Reconciliation Required** checkbox and fill in the remaining detail.

- Indicate the **External Payment Report Task Type** created above.
- Define the **External Payment Report Data Area** that describes the structure of the data supplied with the reconciliation report.

NOTE: Integration with Official Payments. The product provides the data area **C1-OPCExtPaymentReportData** that defines the data provided in the reconciliation report from Official Payments.

- Indicate the **Number Of Days To Wait For Timeout** appropriate for your implementation.
- Define the **Payment Reconciliation Script** that includes the vendor specific logic invoked by the reconciliation algorithm. The script should reference the business object **C1-StandardOneTimePaySSTask** as its schema.

NOTE: Integration with Official Payments. The product provides a service script **C1-PymtRecon** that provides the additional payment reconciliation logic for integration with Official Payments.

Additional Topics

The following sections provide some additional information related to online payment functionality.

Implementing New Payment Destinations

Your implementation may wish to support additional payment options for your taxpayers. For example, perhaps a certain tax type is assigned an external identifier on its tax role and your taxpayers are able to use that identifier to "pay a given tax type". For the purposes of this section, the assumption is that the external identifier is unique across all tax roles in the system. The following information outlines the steps required in this product to support an additional payment destination.

NOTE: Additional steps are required in the self service product. Refer to the self service product documentation for more information.

Lookup

Define a new value in the **C1_PAYMENT_TARGET_FLG** lookup to represent the new payment destination.

Create Payment Algorithm

Each payment destination requires a create payment algorithm that understands how to direct the payment to the appropriate obligation(s). In our example of paying a filing period via external id, the create payment algorithm must use a provided

External ID to determine an appropriate Tax Role. From that tax role, the algorithm must determine the obligation(s) for the tax role that have outstanding debt and direct the payment towards the obligations appropriately.

The algorithm must be coded and implemented using an Algorithm Type and related Algorithm. The base product Create Payment algorithm types may be used as examples.

Characteristic Type

If there is not already a base characteristic type for each distribution detail value that must be provided for the payment distribution to work, one must be created. In this example, an Ad-hoc characteristic type to capture the External ID must be defined.

Distribution Rule

An appropriate distribution rule for each distribution detail required by the new payment destination must be created. In our example a distribution rule for "pay a given tax type" is required configuring the above created characteristic type and create payment algorithm.

NOTE: If the payment destination requires multiple pieces of information, such as an ID plus a filing period, then multiple distribution rules are required, each referring the appropriate characteristic type. Only the last distribution rule should reference the Create Payment algorithm.

Payment Processor Service

The processor service is used for the following purposes:

- Validation. When the payment service task is first created, the processor is called to validate the details. The processor should use the payment destination details to verify that an appropriate record is found and that the appropriate account for that record can be determined. For our example, the processor should take the input External ID and find one and only one tax role with that ID and determine the tax role's account. The processor should provide appropriate errors if the data cannot be found.
- Validate Only. This is a variation of validation and is used for checking payment details when using an external vendor. The only difference between this and the validation logic is that the account ids are not retrieved for this logic.
- Build. When getting ready to process the payment, the service task BO first uses this processor to build the appropriate collection of Payment Distribution Details expected by the creation of a payment. The distribution details built by the processor are captured in the service task in the <paymentDistributionList> collection. A subsequent algorithm uses this information to create the payment with this distribution detail.

When creating a new custom Processor, use any of the base business services provided as an example of the type of functionality needed. All base business services provided begin with **C1-PaymentDestination%**. The processor may also be implemented as a service script, if preferred.

The custom service, whether it be a service script or a business service should include the data area **C1-PayDestProcessorCommon** along with a destination details group that includes the list of payment details provided by the taxpayer. These elements may be defined with specific identifiers. The service task type will contain mapping to populate the elements from the input information.

Service Task Type

Each additional payment destination requires its own service task type. Refer to *Configuration Tasks for Online Payments* for more information.

Account Verification Service

The processor is used to verify if the input account is valid for the entity identified by the current access information. If the account is invalid it returns an error. When creating a new custom Processor, use any of the base business services or service scripts provided as an example of the type of functionality needed. All base components provided begin with C1-OTPVAc%. The processor may be implemented as a service script or a business service. The custom service, whether it be a service script or a business service should include the data area C1-SelfServiceKeys along with an <accountId> element.

Retrieve Payment Due Service

When creating a new custom Processor, use any of the base business services or service scripts provided as an example of the type of functionality needed. All base components provided begin with C1-RetPymtsDue%. The processor may also be implemented as a service script, if preferred.

The custom service, whether it be a service script or a business service should include the data areas C1-SelfServiceKeys and C1-RetrievePaymentsDueCommon.

Service Task Type

Each additional payment destination requires its own service task type. See *Configuration Tasks for Online Payments* for more information.

Implementing a Fee Requirement Script

When integrating with an external payment vendor, the product provides the ability to indicate for each service task type (i.e., payment destination) if the fee should be passed onto the taxpayer or not based on taxpayer type. Your implementation may conditionally pass on the fee to certain kinds of taxpayers. For example, maybe when a taxpayer is paying a filing period, the fee is passed on if the taxpayer is not paying the filing period on time. The product provides a service script (**C1-FeeReqFPr**) that includes this logic.

If your implementation has rules for when a fee should be passed onto the taxpayer based on specific conditions, a service script with the appropriate rules must be implemented. The above base script should be used as a sample, including the input and output that is expected.

Account Information

The self service product user is viewing the tax account data and eventually pays an outstanding balance. Account information portal displays account summary, various alerts, and also filing and payment history.

This functionality follows an inquiry request handling pattern where the service task type captures various configurations and instruction but the request does not result in the service task creation. Otherwise the integration is using the inquiry request flow.

NOTE: Refer to the *Message Processing Overview* for more information about integration flow patterns.

The topics in this section describe more information about the provided functionality, followed by configuration tasks.

Account Summary

The tax account summary contains the most important facts your implementation may wish to bring to taxpayer's attention. The summary is supposed to provide at a glance overview of the account status.

The structure of the summary is a follows:

- **Title** - A short description of the tax account.
- **Details** - Essentials about tax account status, for example a start and end date, the date and the amount of the last payment etc.
- **Location** - An address associated with the account.

- **Current Balance** - The outstanding balance including penalty and interest forecasted as of the current date.

In the self service product the title and the details of the summary are composed using message text with the substitution parameters.

The product is expected to supply the actual data items (dates, numbers, types); this information is used by the self service product as substitution parameters for the summary messages.

The self service product sends an inquiry request containing the tax account identifiers (access type + access keys) and a line of business. The web service is processed by the XAI Inbound Services TSGetTaxAccountSummary, which invokes the service script **C1-GetAcctSm**.

The service script reads the Master Configuration, determines the service task type holding account information-related configurations and invokes an account summary retriever logic associated with request's access type.

Account Summary Retriever

The account summary retriever is expected to deliver the following information:

- Summary Title parameters list.
- Summary Details parameters list.
- Taxpayer Name (required, used internally by the self service product).
- Tax Type (optional, used internally by the self service product).
- Current balance amount and currency (optional, used internally by the self service product).
- Address data.

Retriever logic may also completely override summary title and details text and populate it on the response.

The account summary retriever provided with the base product is implemented for the access type ***Tax Role***. It uses summary instructions defined on the service task type and returns the following details:

- Summary title parameters *tax type* and a *taxpayer name*, summary details parameters: tax role's *startdate*, *outstanding balance*, last *payment's date* and *amount*.
- Account's mailing address.
- Tax role's outstanding balance forecasted as of the current date.

NOTE: Review the business service **C1-GetTaxRoleAccountSummary** as an example of a tax account summary retriever. Refer to *Service Task Types* for more details about summary instructions configuration.

Alerts

Alerts are meant to inform the taxpayer about various urgent matters related to the tax account and in some cases to take immediate action to resolve the issue. Alert may also be used to communicate tax legislation changes potentially affecting the taxpayer or remind about important tax-related dates.

The integration provides out of the box support for the following alerts:

- **Open collections.** This alert indicates that the tax account is associated with open collections.
- **Overdue balance exists**. This alert indicates the existence of an overdue balance for the tax account.
- **Stop Filer Alert.** This alert indicates that return is due for one of the filing periods.
- **Taxpayer Info Incomplete** This alert indicates that taxpayer record doesn't have a valid email address

The topics below describe more information about the provided functionality, followed by configuration tasks.

Retrieving Alerts

The self service product sends an inquiry request containing the tax account identifiers (access type+access keys) and a line of business. The web service is processed by the XAI Inbound Services TSGetTaxAccountAlerts, which invokes the service script C1-GetAlerts.

The service script reads the Master Configuration, determines the service task type holding alert-related configurations for the input line of business, derives applicable alert types, and invokes alert retrievers.

Alert retriever is expected to deliver the following information:

- Alert parameters list.
- Document location (optional).
- Payment Amount.

Alert parameters are used by the self service product as substitution parameters for the alert message. The document location (if provided) is displayed as a link.

Retriever logic may also completely override alert text and populate it on the response.

In the self service product many aspects of the alert are configurable, including the target navigation URL. For example, an alert may read "New child benefit claim forms are available. File now" and the link "File now" would redirect the taxpayer to the online form. The navigation URL may also be overridden by the retriever.

NOTE: See *Tax Account Access Information* for tax account-related inquiry flows.

Alert Type

Self service Alert Type is implemented by the product via extendable lookup. A single lookup value is representing an alert type and references alert retrievers.

The product supports two categories of alerts:

- **Access Based** alerts are alerts whose logic evaluates tax account-related issues. Alerts of this type define retrievers for each supported access type. It is implemented using the extendable lookup business object **C1-AccessTypeAlertLookup**
- **General** alerts are alerts whose logic evaluates system-wide issues. Alerts of this type define a single retriever It is implemented using the extendable lookup business object **C1-GeneralAlertLookup**

Applicable Alerts List

The tax authority has an option to designate certain alerts to a specific audience within the self service user base. The product supports the ability to configure a list of alert types applicable for a line of business, including:

- Tax account-related alerts applicable to a line of business, e.g., alerts concerning corporate taxes.
- Generic alerts applicable for a line of business, e.g., alerts concerning business registration requirements.
- Generic alerts applicable for all taxpayers.

The list is implemented using service task type business objects **C1-AccessAlertsTaskType** and **C1-GeneralAlertsTaskType**. The objects are included in the base product.

Implementing a New Alert Type

Your implementation may wish to introduce additional generic or tax account-specific alerts. The following information outlines the steps required in this product to support a new alert.

NOTE: A new alert should be configured in the self service product. Refer to the self service product documentation for more information.

Alert Retriever

Design and implement the business logic for alert. If the reason for issuing this alert is related to outstanding payment(s), consider retrieving the amount due. Use alert parameters to communicate dates, amounts and other important information.

When creating a new custom retriever, use any of the base business services provided as an example of the type of functionality needed.

The retriever may also be implemented as a service script, if preferred.

The custom service, whether it be a service script or a business service, should include the data area **C1-TaxpayerAccountAlertCommon**.

Extendable Lookup

Define a new value in the extendable lookup to represent the new alert type.

- For tax account-related alerts, use **C1-AccessTypeAlertLookup**.
- For general alerts, use **C1-GeneralAlertLookup**.

Service Task Type

Add the new alert type to the service task type(s) that represent the applicable alerts list.

Payment History

The tax account payment history displays the list of payments. The self service product provides an ability to filter the list by a date range.

For each payment on the list the product displays payment amount, date, payment method, status and the confirmation number (where applicable).

The self service product sends an inquiry request containing the tax account identifiers (access type + access keys) and a line of business. The web service is processed by the XAI Inbound Services **TSGetPaymentHistory**, which invokes the service script **C1-GetPymtHs**.

The service script reads the Master Configuration, determines the service task type holding account information-related configurations and invokes the payment history retriever logic associated with request's access type.

The payment history retriever business service **C1-GetTaxRolePaymentHistory** provided with the base product is implemented for the access type ***Tax Role.***

Filing History

The tax account filing history displays the list of submitted forms. The self service product provides an ability to filter the list by a date range.

For each filing period the following details are retrieved:

- Filing period start and end date
- Form type and received date - not available for missing filing periods
- Filing's due date
- Filing status
- Document locator number
- Confirmation ID
- Form's printable document location
- Amount due (hidden on the self service screen)

The self service product sends an inquiry request containing the tax account identifiers (access type + access keys) and a line of business. The web service is processed by the XAI Inbound Services TSGetPaymentHistory, which invokes the service script **C1-GetPymtHs**.

The service script reads the Master Configuration, determines the service task type holding account information-related configurations and invokes the filing history retriever logic associated with request's access type.

Filing History Retriever

The filing history retriever business service **C1-GetTaxRoleFilingHistory** provided with the product is implemented for the access type ***Tax Role***.

This retriever utilizes the following instructions defined on the service task type for account information:

- Filing types that should be excluded from the response
- Form's characteristic type for the printable form document location
- Indicates if the missing filing periods should be included in the list

The filing status is represented in the base product by the values of the lookup SS_FILING_STATUS_FLG

Configuration Tasks for Account Information

The following sections list the configuration tasks needed to implement the account information functionality.

NOTE: For information about configuration tasks for new alert types, see *Implementing a New Alert Type*.

Characteristic Types

Printable Form Document Location

If your implementation supports the ability to link a printable document (PDF or other format) to the form and these documents are stored in the location accessible for the public, create a characteristic type of class **File Location**. Specify **Form** as a characteristic entity.

Service Task Types

Account Information Task Type

Create service task types for the **account information** configurations with the business object Account Information Task Type. Configure one service task type for each line of business your implementation wish to support.

- Select the line of business from a drop-down list.
- The related transaction BO should be set to C1-StandardInquiryTask.
- Service task class should be set to Standard Inquiry.
- To Do Type should be set to a standard error To Do type to use when transitioning to error. The base product To Do Type Self Service Task Issues (C1-SSTTD) is provided for this purpose.
- To Do Role should be configured to an appropriate default To Do role for the above To Do type. This is optional. If no value is provided the default role for the To Do type is used.
- Configure the Confirmation Header Message Category and Message Number and the Error Header Message Category and Message Number to use for communication to the self service user.

  NOTE: Refer to the Messages section of the *General Configuration Tasks* for more information.

- **Summary Instructions.** If your implementation uses the tax account summary retrievers provided by the base product, define the instructions to use when retrieving the tax account summary:
  - Specify a default name type to be used as a taxpayer name on the summary.
  - Indicate if the address should be included in the summary by default. This configuration could be different for ***Business*** and ***Individual*** line of businesses. For example, an individual is likely to have a single mailing address associated with all tax roles and accounts and there is no need to display the address redundantly in multiple places.
  - List special instructions for the tax types supported by your implementation:
    - Use checkbox to indicate if the primary taxpayer's data should be used for the summary.
    - If the above is unchecked, specify either account-person relationship type or tax role-person relationship type to use in order to find a taxpayer whose information should be used.
    - Specify the taxpayer's name type that should be displayed.
    - Indicate if the summary should include the address. For example, consider the situation where Individual line of business is encompassing both individual income and property taxes. For the individual income, the address bears little significance but for the real property tax it makes sense to display the address associated with the tax roles.
- **Filing History Instructions.** If your implementation uses the tax account filing history retrievers provided by the base product, define the instructions to use when retrieving the tax account fling history:
  - Specify a char type created above as a Char Type for Printable Form Location.
  - Indicate if missing filing periods should be included in the list.
  - List filing types to exclude from the filing history.
- **Payment History Instructions.** If your implementation uses the tax account payment history retrievers provided by the base product, define the instructions to use when retrieving the tax account payment history:
  - Indicate if cancelled payments should be included in the list.
- **Access Type Processors.** For each supported access type list the processors responsible for the tax account information retrieval.
  - Account Summary Retriever.
  - Filing History Retriever.
  - Payment History Retriever.

    NOTE: The base product provides the business services **C1-GetTaxRoleAccountSummary**, **C1-GetTaxRolePaymentHistory**, and **C1-GetTaxRoleFilingHistory**. See the *Account Summary*, *Filing History*, and *Payment History* topics for more details.

Applicable Alerts Task Type

Create service task types for the **general** and **tax account-related alert lists** with the business objects **General Alerts** and **Access Based Alerts**, respectively. Use this configuration to group the alerts according to your business requirements.

- Select the line of business from the drop-down list.
- The related transaction BO should be set to C1-StandardInquiryTask.
- Service task class should be set to Standard Inquiry.
- To Do Type should be set to a standard error To Do type to use when transitioning to error. The base product To Do Type Self Service Task Issues (C1-SSTTD) is provided for this purpose.
- To Do Role should be configured to an appropriate default To Do role for the above To Do type. This is optional. If no value is provided the default role for the To Do type is used.
- Configure the Confirmation Header Message Category and Message Number and the Error Header Message Category and Message Number to use for communication to the self service user.

NOTE: See the Messages section of the *General Configuration Tasks* topic for more information.

- **Alert Types**. List all alert types applicable for the line of business selected above.

Master Configuration

The self service master configuration record includes several settings required for the account information inquiry processing:

- Account Information. For each supported line of business (as defined in the LINE_OF_BUSINESS_FLG lookup) indicate the corresponding Service Task Type for Account Information.
- Alerts.
  - Specify a Service Task Type that defines default generic alert types list. If no line of business is included in the alert inquiry web service request, this list will be used.
  - For each supported line of business (as defined in the LINE_OF_BUSINESS_FLG lookup) indicate the corresponding Service Task Types for tax account-related and generic alerts.

Integration Mapping

Alert Types. All supported alert types must be configured in the alert type domain value mapping in the SOA Composer application.

Tender Types. Payment captured in the system and shown on the payment history may be coming from various sources, not limited to the self service product. In order to display all payments properly all Tender Types defined in the system must be configured in the tender type domain value mapping in the SOA Composer application.

Payment Status. Payment statuses used in the system must be configured in the payment status domain value mapping in the SOA Composer application.

Refer to the self service product documentation for details.

Taxpayer Information

The self service product user is viewing taxpayer information and allowed to make certain updates online. The taxpayer in this case is a taxpayer linked to the tax account in context, which means that "user" is not always equal to a "taxpayer"; for example consider the situation where the business owner is viewing the business account and exploring related business taxpayer record as opposed to the same person viewing his/her own personal income tax account and related individual's information.

Taxpayer portal includes:

- The *taxpayer summary*
- The *contact information*
- The *correspondence information*

This functionality uses the service task types to captures various configurations and instruction but using it may also result in the service task creation.

NOTE: See *Message Processing Overview* for more information about integration flow patterns.

The topics below describe more information about the provided functionality, followed by configuration tasks.

Taxpayer Summary

The taxpayer summary contains the most important facts your implementation may wish to bring to user's attention. The structure of the summary is a follows:

- **Title** - a short description of the taxpayer
- **Details** - essential details.
- **Primary Contact -** the name and an email address of a primary contact

In the self service product the title and the details of the summary are composed using message text with the substitution parameters.

The base product is expected to supply the actual data items - dates, numbers, types; this information is used by the self service product as substitution parameters for the summary messages.

The self service product sends an inquiry request containing the tax account identifiers (access type + access keys) and a line of business. The web service is processed by the XAI Inbound Services **TSGetTaxpayerSummary**, which invokes the service script **C1-GetTxpSum**.

The service script reads the Master Configuration, determines the service task type holding taxpayer information-related configurations and invokes the taxpayer summary retriever logic associated with request's access type.

Taxpayer Summary Retriever

The account summary retriever is expected to deliver the following information:

- Summary Title parameters list
- Summary Details parameters list
- Taxpayer Name (required, used internally by the self service product)
- Taxpayer Type (optional, used internally by the self service product)
- Primary Contact details:
  - Contact Type
  - Contact Name
  - Email address

Retriever logic may also completely override summary title and details text and populate it on the response.

The taxpayer summary retriever provided with the base product is implemented for the access type ***Tax Role***. It uses summary instructions defined on the service task type to return the following details:

- Summary title parameters *taxpayer name* and *ID number*, summary details parameters: *taxpayer type*
- Primary Contact details, where applicable.

Review the business service **C1-RetrieveTaxpayerInfoSummary** for an example of a taxpayer summary retriever. Refer to *Service Task Types* for more details about summary instructions configuration.

Contact Information

Contact information includes a list of taxpayer's phone numbers and an email address stored in the system. The self service user may choose to add and/or edit the information.

The self service product sends a request containing the tax account identifiers (access type + access keys) and a line of business. It may also contain updated contact info. The web service is processed by the XAI Inbound Services **TSGetTaxpayerContactInformation**, which invokes the service script **C1-GetTxpCon**.

The service script reads the Master Configuration and determines the service task type holding taxpayer information-related configurations.

The contact info instruction on the service task type indicates whether it should be the taxpayer itself or the person linked to the taxpayer via person-to-person relationships.

The script proceeds according to the input action:

- For action **READ** it invokes the contact info retriever logic associated with request's access type
- For action **UPDATE** it creates the service task referenced on the service task type. Service task algorithm(s) perform the actual update of the taxpayer information.

NOTE: Refer to the algorithm **C1-UPDTXPINF** for more details about taxpayer contact info update logic.

Contact Info Retriever

The taxpayer contact info retriever provided with the base product is implemented for the access type ***Tax Role***. It uses contact info source instructions defined on the service task type to return the following details:

- List of taxpayer's phone numbers, including extensions.
- Email address.

The instruction indicates whether the source of the contact info is the input taxpayer itself or the person linked to the taxpayer via person-to-person relationships.

Review the business service **C1-RetrieveTaxpayerContactInfo** for an example of a taxpayer contact info retriever. See *Service Task Types* for more details about contact information source instructions configuration.

Correspondence Information

The correspondence information contains list of addresses associated with the taxpayer. The self service user may choose to add and/or edit the specific address.

The portal displays address type and formatted address info for each address instance.

The base product is expected to supply the actual address fields and the formatting is performed in the self service product.

The self service product sends a request containing the tax account identifiers (access type + access keys) and a line of business. The web service is processed by the XAI Inbound Services **TSGetTaxpayerCorrespondenceInformation**, which invokes the service script **C1-GetTxpCor**.

The service script reads the Master Configuration, determines the service task type holding taxpayer correspondence-related configurations and invokes the address retriever logic associated with request's access type

The correspondence info retriever provided with the base product is implemented for the access type ***Tax Role***.

Review the business service **C1-GetTaxpayerCorrespondInfo** for an example of a taxpayer address information retriever. Refer to *Service Task Types* for more details about related configurations.

Update an Address

The self service product allows taxpayer to update an existing address record.

The integration applies the following steps:

- The self service user modifies existing address information or adds a new address.
- The self service product sends a request containing the tax account identifiers (access type + access keys), line of business, and the address data. The web service is processed by the inbound web service **TSAddressMaintenance**, which invokes the service script **C1-MaintAddr**.
- The service script reads the Master Configuration and determines the service task type holding taxpayer correspondence-related configurations for the input line of business.
- It then creates the service task referenced on the service task type. Service task algorithm(s) perform the actual update of the address.

NOTE: Refer to the algorithm **C1-MNTTXPADR** for more details about taxpayer correspondence information update logic.

Configuration Tasks for Taxpayer Information

The following sections list the configuration tasks needed to implement the taxpayer information functionality.

Service Task Types

Taxpayer Information Task Type

Create service task types for the **taxpayer information** configurations with the business object Taxpayer Information Task Type. Configure one service task type for each line of business your implementation wishes to support, as follows:.

- Select the line of business from the drop-down list.
- The related transaction BO should be set to **C1-UpdateTaxpayerInfoTask**.
- The service task class should be set to **Service Request**.
- **To Do Type** should be set to a standard error To Do type to use when transitioning to error. The base product provides **To Do Type Self Service Task Issues** (C1-SSTTD) for this purpose.
- **To Do Role** should be configured to an appropriate default To Do role for the above To Do type. This is optional. If no value is provided, the default role for the To Do type is used.
- Configure the **Confirmation Header Message Category and Message Number** and the **Error Header Message Category and Message Number** to set up communication with the self service user.

NOTE: See the Messages section of the *General Configuration Tasks* for more information.

- **Summary Instructions.** If your implementation uses the taxpayer summary retrievers provided by the base product, define the instructions to use when retrieving the taxpayer summary:
  - Specify a default name type to be used as a taxpayer name on the summary
  - List special instructions for the taxpayer types supported by your implementation. This configuration could be different for ***Business*** and ***Individual*** line of businesses. For example, individual taxpayer may not need/have an additional person to be listed as primary contact, while businesses are likely to have a designated person for this purpose:
    - Use the checkbox to indicate if the primary taxpayer's data should be used as the primary contact.
    - If the above is unchecked, specify either person-to-person relationship type to use in order to find a taxpayer whose information should be used.
    - Specify taxpayer's name type that should be displayed for the primary contact.
- **Contact Info Source Instructions.** If your implementation uses the taxpayer contact info retrievers provided by the base product and the algorithms linked to the base business object, define the instructions below. Note that the taxpayer whose contact information is displayed and updated is not necessarily the same taxpayer whose email was used for the primary contact. Consider the following situation: the taxpayer "in context" is a business. The source of the primary contact for the business could be a customer service department (person) linked to the main business (person). However the source of the contact info should be a business (person) itself, with phone numbers of business's departments.
  - List special instructions for the taxpayer types supported by your implementation:
    - Use the checkbox to indicate if the primary taxpayer's data should be displayed and updated online.
    - If the above is unchecked, specify either person-to-person relationship type to use in order to find a taxpayer whose information should be used
- **Access Type Processors.** For each supported access type list the processors responsible for the tax account information retrieval.
  - Taxpayer Summary Retriever.
  - Contact Info Retriever.

  NOTE: The product provides out of the box the business services **C1-RetrieveTaxpayerInfoSummary** and **C1-RetrieveTaxpayerContactInfo**.

Taxpayer Correspondence Information Task Type

Create service task types for the **correspondence information** configurations with the business object Taxpayer Addresses Task Type. Configure one service task type for each line of business your implementation wish to support.

- Select the line of business from the drop-down list.
- The related transaction BO should be set to C1-GetTaxpayerCorrespondInfo.
- Service task class should be set to Service Request.
- To Do Type should be set to a standard error To Do type to use when transitioning to error. The base product To Do Type Self Service Task Issues (C1-SSTTD) is provided for this purpose.
- To Do Role should be configured to an appropriate default To Do role for the above To Do type. This is optional. If no value is provided the default role for the To Do type is used.
- Configure the Confirmation Header Message Category and Message Number and the Error Header Message Category and Message Number to use for communication to the self service user.

NOTE: See the Messages section of the *General Configuration Tasks* for more information.

- **Access Type Processors.** For each supported access type list the processors responsible for the tax account information retrieval.

  Correspondence Info Retriever

  NOTE: The base product provides the business services **C1-GetTaxpayerCorrespondInfo**.

Master Configuration

The self service master configuration record includes settings required for the account information inquiry processing:

- **Taxpayer Information**. For each supported line of business (as defined in the LINE_OF_BUSINESS_FLG lookup) indicate the corresponding Service Task Type for Taxpayer Information.
- **Correspondence Information**. For each supported line of business (as defined in the LINE_OF_BUSINESS_FLG lookup) indicate the corresponding Service Task Type for Correspondence Information.

Integration Mapping

Phone Types. All Phone Types defined in the system must be configured in the phone type domain value mapping in the SOA Composer application.

Alternatively, your implementation may choose to use exactly same values in both products.

Online Form Processing

In this release, the integration supports on-line form filing. The following diagram illustrates an expected process flow:

The integration has multiple steps.

Initial processing works as follows:

- **Casual user.** The self service application captures information about the taxpayer and sends a web service request to this product to validate the taxpayer (proof of identity - POI).
- **Enrolled user.** The identification is not performed.

  NOTE: See *User Identity Verification* for more information.

The interpretation of the identification outcome includes a provision for the first-time filers. The option is configured in the self service product. If first-time filers are allowed, the identification request may return neither taxpayer ID(s) nor an identification error.

Next the online form is rendered and the taxpayer begins to fill in the information. User may press action buttons to request the data to be copied from the previous return and validate the incomplete form multiple times. Every action is triggering a web service call and the base product is processing the incoming web service request. See *Form Actions* for more details.

Lastly, the user is pressing Ready To File button and the form data is validated one last time. If this action produced no errors, the user is offered an option to review and print the form and then submit it. The integration generates the DLN and also a confirmation ID.

As a result of a successful online form submission the form is created and linked to the form submission service task. The DLN is captured on the service task via characteristic.

The topics below describe more information about the provided functionality, followed by configuration tasks.

Form Data Model

Form definitions are in a sense shared between self service application and the base product. Both systems rely on the metadata when rendering forms on screen and applying various business rules on the form data.

The solution provides the ability to import form type definitions into the self service product.

The self service product operates with generic form data model. The requests originated from the self service product contain the form data in this format:

```
<formData>
<id/>
<name/>
<section type="list">
<id/>
<name/>
<sequence/>
<line type="list">
<field type="group">
<id/>
<name/>
<dataType/>
<value/>
</field>
</line>
<table type="list">
<id/>
<name/>
<tableRow type="list">
<sequence/>
<field type="list">
<id/>
<name/>
<dataType/>
<value/>
</field>
</tableRow>
</table>
</section>
</formData>
```

The form processing integration flow assumes that for the form of a certain type the names of the elements on the generic form data model (sections, lines, tables etc) exactly match the business names of the elements on the form type definitions. This assumption allows an automated transformation of the generic form data XML into form business object XML based on the form type definition.

The base product provided with the business services **C1-TransformFormToGenericWeb** and **C1-TransformGenericWebToForm** for the form data conversion.

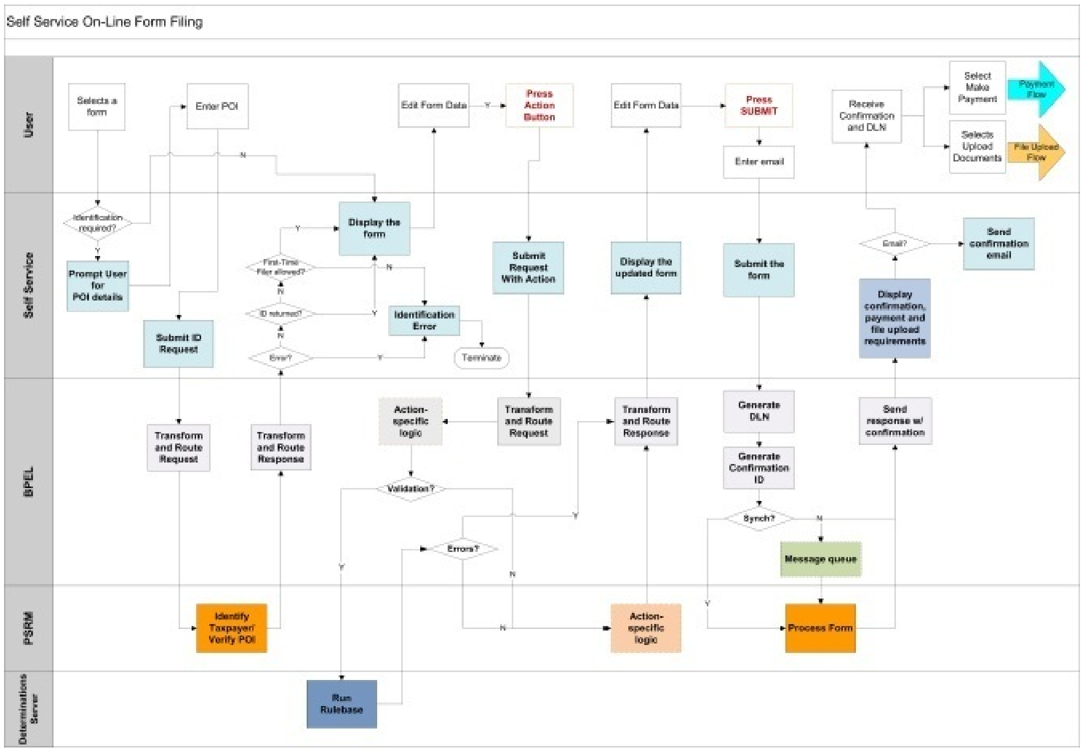
Fastpath: Refer to the *Import Form Definition* for more information

Form Process Request

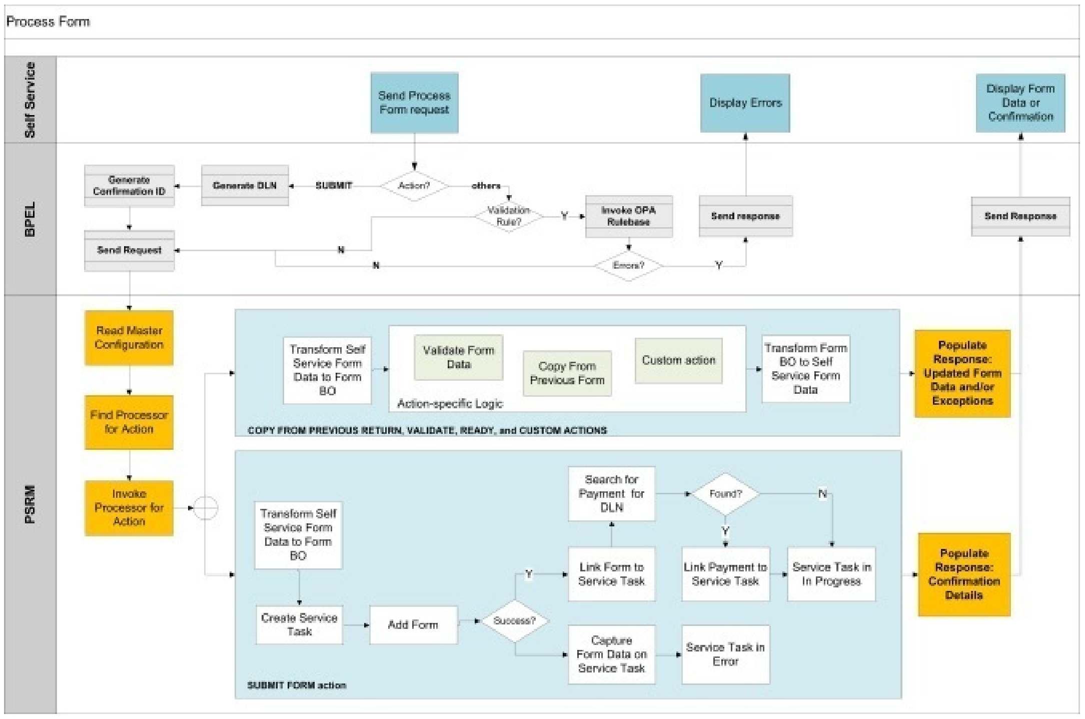
The web services handling online business registration and tax form filing are processed by the XAI Inbound Services **TSProcessRegistrationForm** and **TSProcessTaxForm**, respectively. Both services invoke the service script **C1-RSISvcReq**. The appropriate service task type is derived from the Master Configuration based on the information received from the self service system.

The handling of a specific form action is delegated to the processor components (service scripts or business services) specified on the Master Configuration.

The following diagram illustrates the online form filing flow:



Form Actions



Process Form
Self Service
BPEL
PSRM
Send Process Form request
Display Errors
Display Form Data or Confirmation
Generate Confirmation ID
Generate DLN
SUBMIT
Action?
others
Validation Rule?
Invoke OPA Rulebase
Send response
Send Response
Send Request
Errors?
Read Master Configuration
Find Processor for Action
Invoke Processor for Action
Transform Self Service Form Data to Form BO
Validate Form Data
Copy From Previous Form
Custom action
Action-specific Logic
Transform Form BO to Self Service Form Data
COPY FROM PREVIOUS RETURN, VALIDATE, READY, and CUSTOM ACTIONS
Populate Response: Updated Form Data and/or Exceptions
Transform Self Service Form Data to Form BO
Search for Payment for DLN
Found?
Link Form to Service Task
Link Payment to Service Task
Service Task in In Progress
Create Service Task
Add Form
Success?
Capture Form Data on Service Task
Service Task in Error
SUBMIT FORM action
Populate Response: Confirmation Details

Figure 2: Handling Form Actions in PSRM

The following form actions are supported by the base product.

Copy From Previous Return is expected to return an updated form data or a message indicating that no previous return was found. The product supports an alternative optional scenario for this action. If no previous return is found, the form may be populated with basic taxpayer details, e.g., primary ID, name, and mailing address. This is done by implementing the action **Get Taxpayer Info**.

NOTE: The action **Get Taxpayer Info** is *not* called directly from the self service application.

Check Form Data. This action triggers the form data validation and the response is expected to contain the corrected form data and the list of exceptions. The solution provided in the base product performs the validation using the **C1-ApplyWSSValidation** business service.

Ready to File. This action is expected to perform the final round of the form data validations. The outcome is expected to be similar: updated form data and a list of exceptions, if any. If this action returns no errors/exceptions, the self service product allows the user to submit the form.

Create Form. This action is responsible for actual form creation. The service task created as a result of a form submission plays a slightly different role than other service tasks. The form is created "outside" of the task; the form action processor **C1-CreateForm** creates the form and links it to the service task as a related object.

If the form submission is followed by supporting documents, upload, and/or payments, these entities are also linked to the service task. Thus, the service task serves as the tracking device for the various activities related the form submission. The lifecycle of the service task business object includes a monitor algorithm that completes the service task when the form is successfully posted.

NOTE: All form action processing starts with transforming the self service generic form data XML into base form business object XML and ends with transforming the form business object XML into a self service generic form data XML. The business logic is performed on the form business object.

Notes on Form Validation

The business service **C1-ApplyWSSValidation** invokes the form rules linked to the WSS Validation rule event. When designing the rules, consider the following circumstances:

- Calculate whatever is possible to calculate and minimize manual input.
- Auto-correct is the preferable rule outcome. Create an exception only if the self service user could possibly fix the issue.
- The form may be only partially filled when the user invokes the validation.
- The validation is likely to be invoked multiple times.
- Don't include internal IDs and other sensitive information in exception messages.
- By default, the self service product displays the expanded message text for each rule exception. Make sure these messages are suitable for the self service user; explain the issue in a user-friendly manner and avoid technical details.

Configuration Tasks for Online Form Processing

The following sections list the configuration tasks needed to implement the on-line form processing functionality.

Form Sub-Type

The form sub types defined in the base product - ***Business Registration Form*** and ***Tax Form*** - must be configured in form category domain value mapping in the SOA Composer application. Refer to the self service product documentation for details.

Form Rule Group

The self service user is certainly less competent than the agency user and would likely make various mistakes and miscalculate the numbers. Even though the on-line form is equipped with multiple help and guidance tools and the form data is validated using Oracle Policy Automation-based validation engine, the chances are that the self service user would repeat the *Check My Form* action multiple times. When designing the form rule group(s) associated with WSS Validate (C1WI) rule event try to maximize the auto-correction and to minimize the number of possible exceptions.

Form Source

You may wish to create a dedicated form source for the forms submitted through the self service portal

Service Task Types

Taxpayer Identification Task Type

Define a task type to use for taxpayer identification with the business object Taxpayer Service Request Self Service Task Type.

- The **Related Transaction** BO should be set to **C1-TaxpyrIdentFormFilingSSTask**.

  NOTE: The state algorithm **C1-IDTXFORMF** implements the first-time filer scenario.

- **Service Task Class** should be set to **Taxpayer Identification**.
- **To Do Type** should be set to a standard error To Do type to use when transitioning to error. The base product To Do Type Self Service Task Issues (**C1-SSTTD**) is provided for this purpose. Note that the algorithm plugged in on the error state for the base business object checks that there is not already a To Do for the record before creating one of this type. This allows algorithms that detect an error to also create a To Do with a specific type, if desired.
- **To Do Role** should be configured to an appropriate default To Do role for the above To Do type. This is optional. If no value is provided, the default role for the To Do type is used.
- Configure the Confirmation Header Message Category and Message Number and the Error Header Message Category and Message Number to use for communication to the self service user.

  NOTE: Refer to the **Messages** section of the General Configuration Tasks for more information.

- The service request fields mapping is used when creating the target service task to correctly populate the **requestField** list from the list of fields passed in on the web service request XML. For the base transaction business object, the following fields are expected:

Sequence	Transaction BO XPath
1	requestData/legalName
2	requestData/idType
3	requestData/personIdNumber

- Configure the Service Request Data Instructions. Specify if the identification request task should be persistent and whether the web service response should be echoing back the input data.

NOTE: Refer to the Messages section of the General Configuration Tasks for more information.

NOTE: Service Task Type Mapping. The service task type defined here must be configured in the service task domain value mapping in the SOA Composer application. Refer to the self service product documentation for details.

Form Submission Task Types

Define a task type to use for each Form Sub Type (***business registration*** and ***tax***) with the business object **Form Submission Self Service Task Type**.

- The related transaction BO should be set to **C1-RegFormSubmissionSSTask** and **C1-TaxFormSubmissionSSTask**, respectively.
- The service task class should be set to *Standard Self Service*.
- **To Do Type** should be set to a standard error To Do type to use when transitioning to error. The base product provides **To Do Type Self Service Task Issues** (C1-SSTTD) for this purpose.
- **To Do Role** should be configured to an appropriate default To Do role for the above To Do type. This is optional. If no value is provided, the default role for the To Do type is used.
- Configure the **Confirmation Header Message Category and Message Number**, as well as the **Error Header Message Category and Message Number**, to enable communication to the self service user.

NOTE: See the *Messages* section of the *General Configuration Tasks* topic for more information.

Master Configuration

The self service master configuration record includes several settings required for the form definition import processing.

- Set Char Type for DLN to **C1-DLNSS**.
- Specify the Form Source created above.
- Configure the service task types to use when processing for each form sub-type. Create entries for Business Registration Form and Tax Form sub-types and specify service task types created above.
- Configure the processing components for all form actions supported by the base product. The business services mentioned below are provided with the product. The implementation may use them as-is or implement alternative action processors.
  - Action **Copy From Previous Return** - specify a business service **C1-CopyFromPreviousForm**.
  - Action **Ready to File** - specify a business service **C1-ValidateFormData**.
  - Action **Create Form** - specify a business service **C1-CreateForm**.
  - Action **Check Form Data** - specify a business service **C1-ValidateFormData**.
  - Action **Get Taxpayer Information** – specify a business service **C1-GetTaxpayerInfoForm** if your implementation is to support the special behavior for the **Copy From Previous Return** action, where basic taxpayer information is populated on the form if no previous return is found.

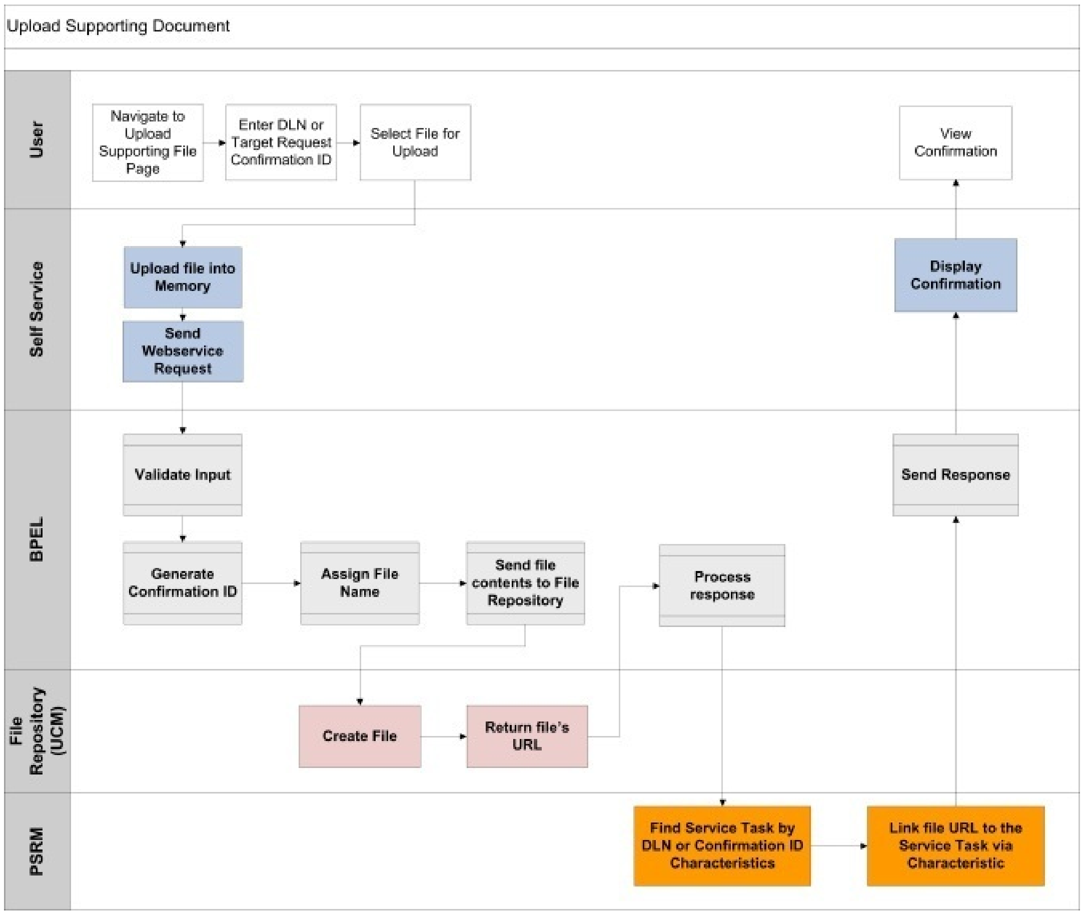
Implementing a Custom Form Action

Your implementation may wish to support additional actions for the online filing process. The self service product allows enabling a form action on specific form. For example, consider the following requirement/implementation scenario:

- **Requirement:** Improve the self service user's experience by splitting **Check My Form** into two actions: *Calculate Totals*, whose purpose is to update the numbers on the form, and *Check my Form* whose purpose is purely data verification.
- **Possible implementation:**
  - Create a new **Form Rule Event** named *Calculate* and implement form rules for calculations.
  - Implement the form action processor to call a rule event created above **C1-ApplyWSSValidation**. For the base product, from XML conversion to and from the self service generic XML, use the business services **C1-TransformFormToGenericWeb** and **C1-TransformGenericWebToForm**.
  - Add a new form action using lookup **C1_SELF_SVC_FORM_ACTION_FLG**.
  - Configure the new action on the **Self Service Master Configuration**.
  - Set up a new action in the self service product and enable it on the form definition(s).

Upload Supporting Documents

The following diagram illustrates the Upload Supporting Documents process flow:



The integration supports an ability to upload supporting documentation for a form or a service request with the following steps:

- The file is uploaded on the self service portal and processed by the SOA Composite.
- File contents are sent to the document repository and the result file location is populated on the request.

- A confirmation ID for the file upload event is generated and populated on the request. The request is forwarded to the base product.

The request is processed by the XAI Inbound Services **TSUploadSupportingDocument**, which invokes the service script **C1-UplSupFle**.

The service script is searching for the service task either by a target confirmation ID or a DLN. The logic is adding a characteristic to the service task referencing the input supporting file URL and adds a new detail to the service task confirmation info containing the file upload confirmation ID.

Configuration Tasks for Supporting Documents Upload

The following sections list the configuration tasks needed to implement the upload supporting files for form functionality.

Master Configuration

The self service master configuration record includes settings required for the form definition import processing.

Specify characteristic type C1-SUPDC as a Char Type for File Location.

Import Form Definition

The integration supports the import of form type definitions into the self service product. The ability to import the definition allows seamless integration between the product and the self service application.

The import process has the following steps:

- The implementer enters a form sub-type and other search criteria. This triggers the web service call.
- The request is processed by the XAI Inbound Service **TSRetrieveActiveFormTypes**, which invokes the service script **C1-RetAcFrTy**. The service script reads the Master Configuration and invokes the defined processor. The list of form types available for import is determined by the product and returned back to the self service portal.
- The user is prompted to select one or more form types. This triggers another web service call for the service **TSRetrieveFormTypeDefinitions**, and the form definitions are retrieved and captured as a self service product's form definition.

The imported information for a single form type includes:

- **Form Sections:** Business name, display sequence, occurrence, label.
- **Form Single Lines:** Business name, display sequence, line number, label, data type, referenced lookup (if applicable), data precision, and scale.
- **Form Group Lines:** Business name, display sequence, line number, label.
- **Referenced lookups:** A list of values and descriptions for each lookup referenced by a form line.

Notes:

- Form Group Line is interpreted as a table in self service product. On the import XML it is represented by the tag <table>.
- Form Single Line within a group line is interpreted as a table column.
- All language-base information is returned for all languages supported by the base product.

Configuration Tasks for Import Form Definition

The following sections list the configuration tasks needed to implement the form definition import functionality.

Master Configuration

The self service master configuration record includes settings required for the form definition import processing.

Specify the business service **C1-RetrieveActiveFormTypes** as Active Form Types Retriever.

Maintaining Self Service Tasks

Use the Service Task transaction to view and maintain service tasks. Navigate using **Main Menu** > **Self Service** > **Service Task.**

Self Service Task Query

Use the query portal to search for a service task. Once a service task is selected, you are brought to the maintenance portal to view and maintain the selected record.

Self Service Task Portal

This page appears when viewing the detail of a specific self service task.

Self Service Task - Main

This portal appears when a Self Service Task has been selected from the Self Service Task Query portal.

The topics in this section describe the base-package zones that appear on this portal.

Self Service Task

The Self Service Task zone contains display-only information about the selected Self Service Task, including the following:

- Audit information for the task, including the web user ID and name, the email address and IP address.
- The list of related objects associated with the task
- Any error information
- Confirmation ID and message and any confirmation details
- Other information that is specific for the type of task. Refer to the in-line help text for more information about the fields.

Self Service Task - Log

Navigate to the Log tab to view the logs for a self service task.

Chapter 24

Oracle Policy Automation Integration

Oracle Public Sector Revenue Management provides tools to facilitate the integration with Oracle Policy Automation platform (OPA).

Integration Overview

Oracle Policy Automation's is designed to handle the implementation of complex policies and business logic. The integration allows invoking OPA rulebase via webservice and retrieving and capturing the determination result, including step-by-step decision report on how the OPA reached the conclusion.

The product supports multiple integration patterns. When selecting an integration method that is appropriate for your business requirements, consider the following scenarios:

- A complex policy, a benefits calculator, an eligibility rule or other determination is implemented as OPA rulebase. This rulebase is deployed on the determinations sever that is accessible either publicly or via organization’s intranet. Your requirement is to perform similar validation or determination. In this scenario, use the *Direct Webservice Integration* method to invoke an independently-designed rulebase from within the product.
- Your implementation designers determined that certain functional requirement is suitable to be implemented with OPA and should be executed during business object lifecycle or as part of another business process. Or, you’ve been designing a Form and realized that most of the Form validation rules can (or better be) implemented using OPA. For these scenarios, use the *Shared Data Model Integration* method to create the rulebase with pre-generated data model and seamlessly incorporate it into the product.

The following sections describe the technical information needed by your implementers to be able to invoke OPA rules from within the system.

Configuring the Direct Webservice Integration

The topics in this section describe configuration required in the product for direct webservice integration with OPA and steps needed to use one of the sample OPA Rulebases provided by the product.

Verify the Rulebase Web Service Adapter Options

The direct webservice integration with OPA uses *web service adapters* to define the connection information to the OPA server and the specific mapping required between the product and the individual OPA rulebases. The product provides a Rulebase Web Service Adapter BO (**F1-OpaWebSvc**). Each OPA Rulebase requires an individual web service adapter that uses this business object.

Before creating any web service adapter records, verify that the Rulebase Web Service Adapter BO has been properly configured for the installation. Each installation must configure two BO options with specific server locations related to the integration. Review the BO options to ensure they are configured. If not, the installation team can provide the appropriate values:

- **Oracle Web Determinations Server Location**. This is the URL of the Oracle Web Determinations server. It should be in the format **http://[server-name]:[portNumber]/determinations-server/soap**. For example **http://opa-server-name:1000/determinations-server/soap**. This is the server location that is part of any WSDL URL used to create an appropriate web service adapter for a specific rulebase.
- **Oracle Policy Automation Rulebase Location**. This is the server directory that includes a copy of the deployment files of the OPA rulebases supplied with the product. This is needed by the business service that generates the mapping data areas when creating a web service adapter for a rulebase.

Once the configuration of the Rulebase Web Service Adapter BO is verified, your implementers may now create specific web service adapters for each specific rulebase used by your implementation.

Configuration Required for a Sample OPA Rulebase

For any sample OPA rulebase provided by the product, most components are supplied by the product, but final steps are required by each implementation. To illustrate the details, we'll use as an example the timeliness validation algorithm for creating an appeal.

- The product supplies the rulebase. In this scenario it is called **Appeal_Validation**. As part of the installation of the product, the deployment files are installed on the OPA server that the product will integrate with and also copied to the rulebase location server directory described above.
- The product supplies an appropriate algorithm type that invokes the call to OPA. The algorithm type is responsible for gathering the data required as input to the OPA rulebase and includes appropriate logic based on the outcome returned by the rulebase. For example, the call to OPA may return a decision report. The algorithm may be designed to store the decision report and link the decision report ID to the appropriate entity for viewing later. In this scenario the product supplies a BO Enter plug-in for the Appeal business object's **Validated** state. It is called *C1-AP-CHKATL* . The product supplies the algorithm type. However, the algorithm requires the Web Service Adapter name as input so the product does not supply the algorithm nor is it plugged into the appeal business object.

The following points explain the additional configuration steps required by an implementer to implement a sample rulebase.

1. Determine the WSDL URL for the rulebase. The WSDL URL is composed as follows **[Oracle Web Determinations Server Location]/[rulebase name]/specific?wsdl**. Using the URL example above and this rulebase the WSDL URL would be **http://opa-server-name:1000/determinations-server/soap/Appeal_Validation/specific?wsdl**.

   TIP:
   To verify that you have composed the WSDL URL correctly, paste the composed URL into an internet browser and verify that you can access the WSDL.

2. Define a web service adapter for the rulebase using the rulebase WSDL. Refer to *Understanding Web Service Adapters* for details about fully creating and generating the web service adapter.

3. Create an algorithm for the appropriate algorithm type provided by the product for this particular OPA integration. The algorithm requires the web service adapter name. Depending on the specific algorithm type, there may be additional parameters required.
4. Plug the algorithm into the appropriate plug-in spot as defined by the specific integration point.

FASTPATH:
For this scenario, refer to *Configuring the Appeal Timeliness Rule* for specifics about defining the algorithm correctly and plugging it into the appropriate plug-in spot.

Defining a New Rule

The first step is to design the business rule. In addition to designing the logic of the OPA rulebase, your designers must consider where and when the business rule should be invoked by the product as well as what logic should occur in the product based on the output from the call to OPA.

The topics in this section describe more detail to consider when implementing a new business rule using an OPA rulebase.

Develop the Algorithm to Invoke the Rule

An important step in considering an integration point with OPA is to consider when it should be invoked within the product. In other words, what plug-in spot? For example:

- Is the business rule validation that must be invoked any time a certain object changes? Then perhaps this should be implemented as a BO validation algorithm for an appropriate business object.
- Is the business rule validation that determines if an object may transition to a different status? Then perhaps this should be implemented as an enter plug-in algorithm on an appropriate state for a business object.

As part of determining the appropriate plug-in spot, the designer should become familiar with the expected responsibilities of the plug-in spot and the data it receives when invoked.

Next, the designer should determine if the product already supplies an algorithm type for the desired plug-in spot that invokes OPA. For example, perhaps the product supplies an out of the box integration with OPA and the algorithm logic (the data it retrieves to send to OPA and the logic that occurs after returning from OPA) is appropriate for your business rule. However, the detailed logic performed by the product's sample rulebase does not match the business rule needed by the implementation. If so, your implementation may only require a new OPA rulebase to plug into this algorithm type.

The following points highlight more detail needed for designing a new algorithm to invoke an OPA rulebase:

- Work with the business rule designer to determine what information needs to be provided to OPA. The algorithm is responsible for gathering all the information required by OPA, if it is not already supplied to the algorithm.
- To invoke OPA, the product provides a web service dispatcher business service (**F1-InvokeWebService**). The business service requires the web service adapter name, the name of the operation to invoke (typically **Assess** for OPA integration) and the request and response data areas related to the operation. The algorithm must determine the appropriate web service adapter and must populate the request data area with the information required for the OPA call.

  The product algorithms have typically been designed to define the web service adapter name as an input parameter. The algorithm then includes logic to retrieve the request and response data areas defined on the web service adapter. This allows for algorithm type re-use. It means that the same algorithm type may be used for different business rules implemented with different OPA rulebases, assuming that the other logic performed by the algorithm applies to the different rules.
- Determine what logic should occur upon receiving the rulebase response. See the Rulebase Outcome section below for more information.

Rulebase Outcome

The type of data returned by a rulebase is configured in the request by populating the Outcome Style element for each output value (referred to as a "goal" in OPA terminology).

- An outcome style of **value-only** tells OPA to simply return the value of the goal (output) . This is the default if nothing is specified in the request.
- An outcome style of **decision-report** tells OPA to return a detailed report of why a particular result is given in addition to the value of the goal.

NOTE:
The above values are defined in the Outcome Style extendable lookup (**F1-OutcomeStyleLookup**).

Based on the business requirements, the algorithm must determine what should occur based on the results.

If the business requirement dictates that the decision report should be stored, the product provides a business service Create Decision Report (**C1-CreateDecisionReport**). This business service stores the decision report details in a special "attachment" table and returns the ID. Based on the business rules, the ID may be captured and stored with the object that requested the call to OPA. The following points provide more detail on storing a decision report.

- The following information should be provided to the business service to create the decision report:
- Populate the BO with **F1-DecRpt**
- Populate the MO and prime key values to appropriate values based on the object that invoked the OPA rule. For example, if the OPA rulebase was used to validate an appeal, the MO is set to **C1-APPEAL** and the PK value is set to the appeal id. This allows cross referencing on the decision report to the object that created it.
- Populate the Decision Report Data with the appropriate decision report node returned by the call to OPA.
- The attachment id is returned. To store a link to the decision with the object that created it, the product recommendation is to create a Log Entry, if the maintenance object supports logs. The characteristic type for the log should be set to **C1-DCRPT** and the characteristic value should be set to the attachment ID.

NOTE:
The characteristic type may require additional configuration to include the log in question as one of the valid characteristic entities.

Refer to *Viewing Decision Report* for information about viewing the decision report.

Deploying the Rulebase

Use Oracle Policy Modeling to develop the rulebase. Once the rulebase is tested and it is compiled and built, the deployment file is created.

- This rulebase should be deployed on the Oracle Web Determinations server. Note that in most implementations a separate group such as a release services group is responsible for this type of task.
- As described in *Configuring the Integration*, besides deploying the rulebase on the server, the web service adapter logic requires the rulebase to be copied to the server directory defined in the BO option Oracle Policy Automation Rulebase Location.

Final Steps

The remaining steps for the integration involve creating the web service adapter and fully configuring and plugging in the algorithm.

FASTPATH:
Refer to *Configuring the Integration* for more information.

Configuring the Shared Data Model Integration

The topics in this section describe configuration required in the product for shared data model integration with OPA and provide high-level guidelines for OPA rulebases participating in this integration method.

The idea of this integration method is that an object in the product and the rulebase may share the data structure. Firstly, the entity is designed and defined in the main system. Then the entity's data structure is described using a special xml format and stored in a file that can be imported into OPA rulebase as a data model. The similarity of the models allows seamless exchange of the information between the product and the OPA rulebase. The webservice interaction between the product and the OPA determinations server is completely encapsulated.

Generated Rulebase Data Model

The data model has to be generated for each individual OPA rulebase invoked using this integration method. The generated file is stored as an owned attachment linked to the source object, for example Form Type or Data Area. The product provides an OPA Rulebase Data Model BO (**C1-OPARulebaseDataModel**) to capture the data model xml. The product also supplies the OPA Rulebase Data Model (attachment) search.

TIP:
Multiple rulebases can be built on the same data model. For tracking purposes capture the rulebase name on the Generated Data Model attachment.

NOTE:
The product does not support an automatic synchronization between the source entity (such as Form Type or Data Area) and the generated rulebase data model. Manually propagate the minor modifications of the source entity to the rulebase properties. Regenerate the data model if you significantly modified the Form Type structure or the data area's schema.

Setup OPA Integration Master Configuration

The shared data model integration uses master configuration to define the connection information to the OPA server, setup parameters for the data model generator and store default configurations for the OPA-based form rules. The product provides an OPA Integration Configuration BO (**C1-SelfServiceMasterConfig**).

Setup OPA Integration Configuration prior to an attempt to generate OPA rulebase data model for either a form type or a data area. For configurations related to the specific OPA installation, request the appropriate values from the installation team.

The the end-point URL for the rulebase webservice invocation is constructed dynamically at run-time. The URL is composed as: **[port:host]/[determitation server deployment location]/[webservice wsdl reference]/[name of the rulebase invoked]**,for example **http://myhost:0000/determinations-server0000/assess/soap/generic/MyRuleBase**.

The master configuration defines parts of the end-point URL pertaining to the OPA sever participating in the integration:

- **Determinations Server Location**. This is the deployment location of the Oracle Determinations server. It should be in the format **http://[server-name]:[portNumber]/[determinations server]**. For example, **http://myopaserver:0000/determinations-server**. By default, the **determinations-server** directory is created during OPA installation. However, OPA does not imposes a mandatory directory structure, leaving it at the installation team's discretion.
- **Web Determinations Location**. This is the deployment location of the Oracle Web Determinations. It should be in the format **http://[server-name]:[portNumber]/[web determinations location]**. For example, **http://myopaserver:0000/**

web-determinations. This definition can be used if you implement requirement to trigger OPA web interview from the hyperlink placed anywhere on the portal zone.

- **Rulebase WSDL Reference**. This is the partial path of the ***Assess*** generic wsdl used for the integration. Combined with the value of the **Oracle Determinations Server** and the rulebase name it produces the full end-point URL for the rulebase invocation. This attribute may refer to either a version-neutral or a version-specific wsdl. Specify **/assess/soap/generic/** unless your business requirements dictate executing the historic version of OPA rulebases. An example of a version-specific reference: **/assess/soap/generic/10.4/**. Note that this definition is applicable to all run-time interactions with OPA.
- **Rulebase List WSDL Reference.** This is the partial path pointing to the determination server's wsdl. Combined with the value of the **Oracle Determinations Server** attribute, it produces the full URL of the wsdl. This wsdl described the ***odsServer*** service that the system invokes to retrieve list of rulebases deployed on the Oracle Determination Server
- **Installed OPA Version**. This is the exact version of OPA instans participating in the integration. The precise version number is necessary to ensure the compatibility of the generated data model with the current installation. It should be in a format of **N.N.N.N**, for example **10.4.4.21**

At design time the following definitions are used by the rulebase data model generator services **C1-DataAreaGenerateDataModel** and **C1-GenerateOPARulebaseModel:**

- **System Name** This is an alphanumeric code that identifies the current system. It is used internally by the data model generator and indicates the product that is triggering the generation process. Specify: **PSRM**.
- **Attachment Business Object**. This is a business object that the system uses to store the generated data model. The product provides the business object **C1-OPARulebaseDataModel** for this purpose.

The following definitions are used at run-time for the webservice request:

- **SOAP Action**. This is the definition of the SOAP action for the OPA webservice. It is version-specific and should include the 2–parts OPA version number. For ***Assess*** webservice specify: **http://oracle.com/determinations/server/10.4/rulebase/types/Assess**.

NOTE:
It is important to keep the master configuration in synch with the actual OPA installation. Verify the configuration every time a newer version of OPA is installed.

Configuration Required for Form Validation using OPA Rulebase

Consider implementing the OPA-based Form Rule if you intention is to validate the entire form at once and implement multiple business rules, verifications and calculations. In this scenario most infrastructure components are supplied by the product and only few configuration steps and business logic implementation in OPA are required:

- Generate an OPA rulebase data model for the Form Type. This option is available only for active Form Types.
- Save the data model file locally with extension **xsrc**.
- Create a new OPA rulebase and add the data model file as properties file. Develop, test and deploy the rulebase.

FASTPATH:
Refer to *Defining a New Rule with Pre-generated Model* for specifics about creating the form validation rulebase.

- Add a Form Rule to the source Form Type and associate it with the rulebase created above.. The product supplies a Form Rule BO (**C1-OpaValidation**) complete with an apply rule algorithm type that performs the call to OPA. The algorithm is responsible for gathering the form data required as input to the OPA rulebase, followed by webservice invocation and includes appropriate logic based on the outcome returned by the rulebase. For example, the Form Rule configuration may indicate that call to OPA should return a decision report. The algorithm is designed to store the decision report and link the decision report ID to the Form Log for viewing later. The product supplies both the algorithm type *C1-OPAVLAPRL* and the algorithm that is plugged into the form rule business object.

FASTPATH:
Refer to *Using OPA Validation for the Form Rule* for specifics about configuring the OPA-based Form Rule.

Configuration Required for Data Area Validation using OPA Rulebase

The product provides the infrastructure allowing to process a data area using OPA rulebase. Most components are supplied by the product, but final steps are required by each implementation. To illustrate the details, we'll assume that there is a requirement to implement an eligibility rule plug-in using OPA.

Remember that the rulebase has no direct access to the product database and may not be able to retrieve additional information. Hence the preliminary processing logic supposed to gather all the information required to determine the eligibility. This logic could be implemented using Java or scripting. Next, the data should be passed to the OPA rulebase that in turn will determine the eligibility.

Here is what’s provided by the product:

- The product supports the ability to generate rulebase data model that is structurally similar to the data area’s schema.
- The product supplies a context-sensitive dashboard zone that contains an action button that is triggering the generator. The zone also displays the information about previously generated data models. The information is displayed as a hyperlink that points to the OPA Rulebase Data Model search. Another dashboard zone provides a complimentary ability to view the plug-in spot schema.
- The product supplies a business service **C1-DataAreaInvokeOPA** to facilitate the call to OPA. It is responsible for transforming the data into the format compatible with OPA webservice request and includes logic based on the outcome returned by the rulebase. Either updated xml or an error message are returned. Business service input allows to specify if call to OPA should return a decision report. You can specify that decision report is either stored immediately as an attachment or returned to the caller as a raw xml.

The following points explain the configuration steps required by an implementer to implement a data area validation via rulebase.

- Define the input for the rulebase and create a new data area that will serve as an API for the OPA call.
- Generate the rulebase data model and save it locally with extension **xsrc**.
- Create a new OPA rulebase and add the data model file as properties file. Develop, test and deploy the rulebase.
- Implement the call to the business service **C1-DataAreaInvokeOPA**, sending in the data area and the rulebase name.

FASTPATH:
Refer to *Defining a New Rule with Pre-generated Model* for specifics about creating a rulebase with pre-generated data model.

Defining a Rule with Pre-generated Model

The topics in this section describe more detail to consider when implementing a new business logic using an OPA rulebase with pre-generated data model.

Understanding the Generated Data Model

The generated data model reflects the structure of the source entity. Each entity name corresponds to the xml node name and may be prefixed with the succession of parent node names and node type “qualifiers”, such as *section*, *line*, *table*, *list*, *group* or *tablerow*. This is done in order to ensure the uniqueness of the entity names within the generated model.

Each entity that represents the actual data-bearing elements, has two value-holder attributes, one for an original input value and one for the value determined as a result of the rulebase execution. The input value is named **xxx_submitted_value**, the output is named **xxx_determined_value.**

Develop the OPA Rule

In OPA development environment create a new project, select **Properties** folder on the project explorer three and use the option **Add Existing File** to add the generated data model file as a properties file. Perform an initial sanity check: compile the rulebase and verify that the generated data model contains no errors.

Set up the initial values for the attributes that will be referenced by the validation logic. To validate or pre-calculate the values you may need to perform various mathematical and logical operations with the values. Note that OPA does not automatically sets either blank values for text or zeroes for numeric attributes. Instead, it defaults all of them to a the special value **unknown**. This value is not equivalent to a null and cannot be evaluated or compared with other regular values, hence you may need to perform an explicit initialization of the attributes whose values are not provided. This example illustrates the initialization of a numeric attribute in the OPA rule:

Total Gross Sales' Determined Value	
0	The total gross sales' submitted value is unknown.
The total gross sales' submitted value	otherwise

Implement form data validations and calculations. The ultimate goal is to determine whether the data is valid, to correct the invalid values and/or provide the calculation results. The rulebase may also raise form exceptions. For that purpose, you should populate the entity’s attribute named **xxx_error_message** with the actual error text.

Certain limitations apply:

- **Do not alter public names of the entities and attributes.** It will break the integration. You can modify the generated entity's text to make the entity more usable and intuitive. For example you may turn the generated "lineItems section’s firstPrepayment" into "the first prepayment". This simplifies the rule writing process and makes you rule much more readable..
- **Do not remove entities and/or value-holder attributes from the generated model**. The integration logic populates the webservice request with all the values available in the system. The corresponding entity and/or attribute must be defined on the receiving side.
- **Data model changes should be made in both systems**. If the business requirements dictate changes to the data model, similar changes should be made to the model’s source object’s structure.
- **The output error message size should not exceed 250 characters**. This is an internal integration limitation.

Rulebase Outcome

The rulebase outcome comprise multiple determined values, validation error(s) and error indicators. Make sure your rulebase’s output meets the expectations of the corresponding integration component:

- A form rule algorithm *C1-OPAVLAPRL* expects the **error_severity** and **rule_outcome** indicators to be populated with numeric values. These values are mapped to the standard form rule parameters **Exception Class** and **Rule Action** and further interpreted by the product’s form rule engine. The error messages returned by OPA are captured as form exceptions.
- A business service **C1-DataAreaInvokeOPA** expects that OPA returns a single error message and treats it as a regular application error. .

The product integration supports retrieving a single decision report associated with the root data model entity. This entity has a boolean attribute named **xxx_contains_error** that should be set to **true** or **false** by the rulebase.

If requested, the integration component stores the decision report and links it to the “owner” entity via log..

Refer to *Viewing Decision Report* for information about viewing the decision report.

Deploying the Rulebase

Use Oracle Policy Modeling to develop the rulebase. Once the rulebase is tested and it is compiled and built, the deployment file is created.

- This rulebase should be deployed on the Oracle Web Determinations server. Note that in most implementations a separate group such as a release services group is responsible for this type of task.

Final Steps

The remaining steps for the integration involve configuring the components that invoke the rulebase.

FASTPATH:
Refer to *Configuring the Shared Data Model Integration* for more information.

Viewing the Decision Report

The product provides examples of storing a resulting decision report after a call to an OPA rulebase. For example, the **Appeal - Check Timeliness** algorithm stores the resulting decision report and provides a link to the decision report via the appeal log.

FASTPATH:
Refer to *C1-AP-CHKATL* for more information.

As described in *direct webservice integration* and *shared data model integration* chapters, a stored decision report can be linked to the calling object using a log entry and its characteristics. The section indicates an appropriate foreign key characteristic type to use.

A user views the decision report by navigating to the log and clicking the hyperlink for the stored decision report (attachment) id. Assuming that the appropriate characteristics type was used, clicking the hyperlink causes the decision report to display in a separate window.

NOTE:
The decision report (attachment) can be referenced via foreign key anywhere on the business object’s schema. In this case it appears on UI as a regular hyperlink.

Chapter 25

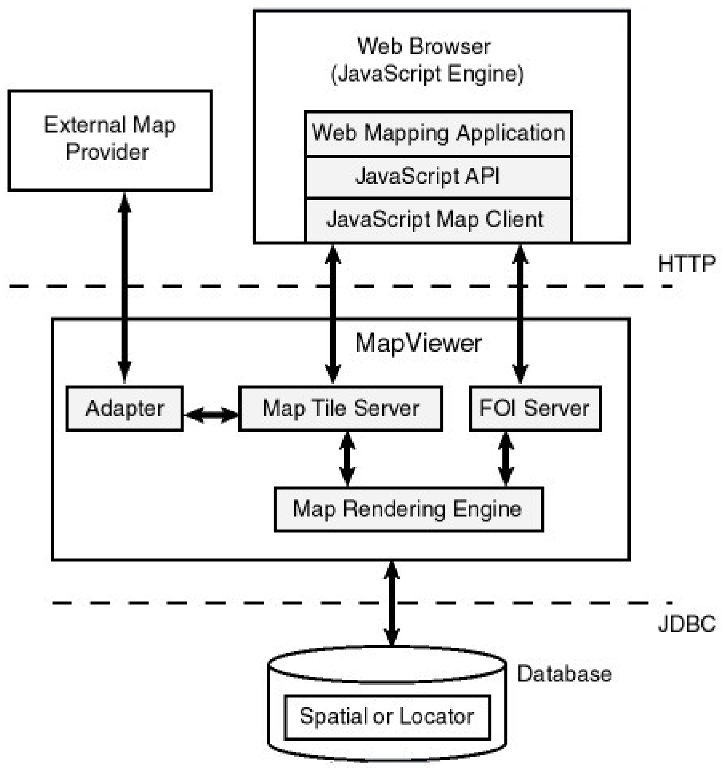
Oracle MapViewer Integration

Oracle Public Sector Revenue Management provides tools to facilitate the integration with Oracle Fusion Middleware MapViewer (Oracle MapViewer) and supports the ability to display geographic maps on the portal..

The topics in this section describe the technical information needed by your implementers to configure the integration.

About the Oracle MapViewer

Oracle MapViewer supports the ability to incorporate highly interactive maps and spatial analysis into business applications.



The system uses the Oracle MapViewer JavaScript API to retrieve the map and render it on the page.

It invokes JavaScript map client that is fetching the map image tiles from the map tile server, then displays the map in the Web browser. The application may also invoke the JavaScript map client to fetch dynamic spatial features from the feature of interest (FOI) server and display them on top of the map tiles.

Tiles can be fetched either directly from the Oracle MapViewer map rendering engine or from an external Web map services provider.

Oracle MapViewer configurations including topology, map layers and themes are configured using Oracle MapViewer Management Console.

FASTPATH: For more details refer to the Oracle Fusion Middleware MapViewer documentation.

Integrating Maps

The product integrates with Oracle MapViewer as follows:

- The integrated Oracle MapViewer is deployed on the network. It is connected to a map data source. The actual map data is not included in the base product and has to be obtained (and possibly licensed) separately. Any map data source compatible with Oracle MapViewer may be used.
- The reference to the Oracle MapViewer is configured in the system through a single, dedicated master configuration.
- The map is rendered on the UI Map within Explicit Map Portal Zone. UI Map's HTML includes JavaScript code that uses the Oracle MapViewer API to invoke various functions and retrieve and display the map.

For information on featured integration patterns see *Basic Map Entities*.

Basic Map Entities

Multiple entities in the system are associated with geographic location, directly or by proxy.

Map-enabled entities are those in which the data includes the latitude/longitude directly or in which the latitude/longitude can be derived using other information captured on the Business Object.

For these entities the following is supported:

- Maintenance portal includes a geographic map that shows location(s) associated with the entity, e.g., a person's address.

  NOTE: The map's zoom level is adjusted automatically to include all locations in the initial rendering.

- Additional information about an object residing in this location, such as property picture, asset’s appraised value etc. is displayed.
- Search results include View Map option.

Map Zone on Maintenance Portal

The availability of the map is controlled by the Business Object option **Geo Map Derivation Script**. This option references a service script that retrieves list of geo locations associated with the entity.

The zone on the maintenance portal is visible only when the script returns one or more locations.

NOTE: The product geo map derivation service scripts follow a naming convention of script code `C1-GeoMap%`. These scripts share a common schema structure:

```
<schema>
    <includeDA name="C1-GeoMapInfo"/>
</schema>
```

Map on the Search/Query Results

The search results for the map-enabled entity includes View Map option. The derivation of the location information is performed by Business Object’s Geo map Derivation Script. The map is shown on the popup window.

Customizing the Integration

The following components of the integration can be customized:

- **Geo Map Derivation Script**. It returns the whole set of information used to retrieve and render the map. It includes the list of locations to display and also the general map definitions, such as initial zoom levels, pointer images, map layer and even the Oracle MapViewer installation. Custom geo Map derivation Script logic can completely override the entire map.
- **Zone Visibility Script**. The base product script evaluates whether the geo location information is available in order to decide if the map should be displayed. The implementation may configure alternative conditions for map visibility.
- **Pointer Image**. You can customize/replace the graphic delivered with the base product and/or associate special graphic with a location. For example, Address Type can be associated with a special pointer image.

Configuring the Integration

To enable the integration you need to set up the MapViewer Configuration master configuration as follows. This setup is used at runtime when the system renders the map.

1. Navigate to **Admin** > **Master Configuration** and select the **MapViewer Configuration** business object from the list.
2. Define a reference to the Oracle MapViewer installation connected to the current environment:
   - **Base URL** defines the location of the Oracle MapViewer installation.
   - **Java Script Path** defines the location of MapViewer JavaScript library. The path should be specified relative to the base URL (see above).
   - **Spatial Reference ID** uniquely identifies the spatial coordinate system. Refer to *https://en.wikipedia.org/wiki/Spatial_reference_system* and *https://en.wikipedia.org/wiki/SRID* for more information.
3. Define the map source and initial rendering parameters:
   - **Default Window Style** refers to one of the styles available from Oracle MapViewer API.
   - **Map Data Source** refers to one of the map data sources installed and configured in Oracle MapViewer. It is the source for the embedded maps.
   - **Map Tile Layer** defines the default tile layer displayed on the map. Single Map Data Source may include multiple tile layers whose content may wary. The tile layer is the ultimate link between the UI component and the MapViewer.

     NOTE: The master configuration defines a **default** map tile layer used for *Basic Map Entities*. The implementation can integrate other tile layers in order to address specific business requirements.

   - **Default Pointer Image** references an image used as a location pointer on the map.
   - **Default Initial Zoom**, **Default Maximum Zoom**, and **Default Minimum Zoom** define the initial parameters of the rendering.
4. Additional configurations include:
   - **Default Info Callout Title** defines the metadata field whose description is used as the title of the info callout displayed when user double-clicks on a location pointer image.
   - **Default Historic Address Label** defines the metadata field whose description is used as a label for the historical addresses.
   - **Labels** for the map zoom levels. These labels are displayed on the map's zoom control widget. Example: Street, City, Country.

Enabling a Map on a Maintenance Portal

Availability of the latitude and longitude is imperative for displaying the location on the map.

Map-enabled entities are those in which the data includes the latitude/longitude directly or in which the data can be derived using other information captured on the Business Object.

To enable a map on a maintenance portal:

1. Derive geographic map information by creating a service script that retrieves one or more locations associated with the object.

   The common schema for geo map derivation service scripts is:

   ```
   <schema>
       <includeDA name="C1-GeoMapInfo"/>
   </schema>
   ```

   For each location retrieve:

   - **Latitude** and **Longitude**.
   - **Tooltip** text is the text that is shown in the callout when the user hovers over the pointer image.
   - **Additional information contents** and **title** is the information displayed in the popup when user clicks on the pointer image and the title for this popup. The information can be returned as plain text or HTML.
   - **Pointer Image Location** is the alternative image for the pointer image. Populate it if your implementation wishes to override the default graphic, or when a certain location(s) is associated with a special image.

   There is also an option to override the displayed map completely. The output of the script may include an alternative map window style, map theme, map layer, default initial zoom and even point to an alternative Oracle MapViewer installation.

2. Define a business object option by adding a new option, **Geo Map Derivation Script**, to the business object. Specify the service script created above.

3. Add a portal zone by adding an **Explicit Map Zone** to the maintenance portal.

   - **Display Object**. Specify `ss='C1-GeoMapDer'`. This common service script derives object-specific **Geo Map Derivation** script and invokes it to retrieve the map data.
   - **UI Map**. Specify map='C1-GeoMap'. This map is designed to display a set of geo locations derived by a **Geo Map Derivation** script.
   - **Zone Visibility Service Script**. The base product includes visibility script that evaluates an object by maintenance object+primary keys combination, invokes Geo Map Derivation script and returns true if there is at leat one location to display. An example of this parameter configuration for **Asset**:

     ```
     ss='C1-GMShwZn'
     input=[mo='C1-ASSET' pkval1=ASSET_ID]
     output=shouldShowZone
     ```

   - **XML Parameters 1-15**. Use these parameters to pass maintenance object and primary keys for the entity in context to the service script. An example of XML Parameters 1 and 2 for Asset:
     XML Parameter 1:

     ```
     name=ASSET_ID
     targetPath=input/pkValue1
     poprule=R
     ```

     XML Parameter 2:

     ```
     name=MAINT_OBJ_CD
     targetPath=input/mo
     spec='C1-ASSET'
     ```

   Refer to Zone `C1-ASSETMAP` as an example.

4. Configure the main query/search zone. The **Map** column with the **View** link does not appear on the search result automatically, so the zone configuration must be amended as follows:

   - Add logic to check for the existence of the geo location information (e.g., for the existence of at least one address linked to an asset whose latitude and longitude are populated. Return the existence indicator as an SQL column.

- Make sure the SQL results include the entity identifiers **Maintenance Object** and **Primary Keys**.
- Configure an explorer column to invoke the BPA script `C1-GeoMapOP`, passing entity information (**Maintenance Object** and **Primary Keys**) as an input. The script determines the business objects, invokes the geo map derivation script, and opens a UI map with the data returned by the script.

  An example of the column configuration for the **Asset Search**:

  ```
  source=SQLCOL
  sqlcol=LABEL_LONG
  label=ADDRESS_MAP_LBL
  bpa='C1-GeoMapOP'
  tempstorage=[mo='C1-ASSET' pkValue1=ASSET_ID]
  ```

Refer to zone `C1-ASSETQ1` as an example.

Chapter 26

CTI-IVR Integration

Oracle Public Sector Revenue Management 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 Public Sector Revenue Management
- 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.

Launching The System From an External Application

There following sections describe possible options to launch the system from an external system.

Launching Control Central Using an ActiveX Navigator

Oracle Public Sector Revenue Management provides an ActiveX component **CDxCTI.CDxNavigator** that external applications, such an IVR application, can use to launch Control Central for a given account number or phone number.

NOTE:
Enable ActiveX. In order to use the navigator, you must *configure your browser to enable ActiveX*.

The CDxNavigator object exposes two methods:

- **ContralCentralByAccountId** invokes Control Central for a given account ID.
- **ControlCentralByPhone** invokes Control Central for a given phone number.

Method: ControlCentralByAccount

Input:

- **Server URL:** The Oracle Public Sector Revenue Management 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 Public Sector Revenue Management 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 Public Sector Revenue Management.
- 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 Public Sector Revenue Management server: /ci/cti/CTISample.HTM.

- Start an Internet Explorer session.
- Navigate to URL above.
- Select an account ID or phone number and click **Navigate**.

NOTE:
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 Public Sector Revenue Management.

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).

FASTPATH:
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.

Overview of Automated Dialer

In order to initiate a call to a taxpayer from within the system, a context menu item **Go To Automated Dialer** is available on the Person context menu. To call a taxpayer 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**.

NOTE:
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

Micorsoft Phone Dialer Configuration

If your implementation chooses to use the functionality provided with the system and integrate with Microsoft Phone Dialer, you must copy the JSP page ext_cti_dialer.jsp from the /cm_templates directory found under the web application root directory on your Oracle Enterprise Taxation and Policy Management server to the /cm directory.

NOTE:
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 Public Sector Revenue Management 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 **Admin, 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**.

FASTPATH:
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 BPA script **C1-GetNxtClr** is provided and can be used to request the next call waiting in an inbound queue managed by a CTI application. Your implementation may choose to configure a menu entry for this BPA script or may recommend that appropriate users configure this BPA script as a "favorite" script and enable the Favorite Scripts dashboard zone.

When the BPA is invoked, 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 Enterprise Taxation and Policy Management 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

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");
```

Chapter 27

Configuration Migration Assistant Addendum

The Configuration Migration Assistant (CMA) provides customers with a flexible, extensible facility for migrating their configuration data from one environment to another e.g., from a production environment to a test environment. Data is exported from the source system to a file. The file can then be checked in to a version control system for reuse, or can be immediately imported into the target system and applied.

This addendum describes how the Configuration Migration Assistant can be used with Oracle Public Sector Revenue Management.

Read and review the *Configuration Migration Assistant* chapter in the Oracle Utilities Application Framework Administration Guide before attempting to use this functionality.

At a high level, migrating data involves the following steps:

- Configuration:
  - Identify or create *Migration Plans* that each define a maintenance object to migrate and includes sub instructions to identify related objects. The product supplies migration plans for most administrative maintenance objects that may be used. Or implementations may develop their own.
  - Identify or create an appropriate *Migration Request*, which defines the migration plans to include in a given export. Specific selection criteria may be used when configuring the migration plans to limit the data that is included in the request.
- Export the data set. Use the *Migration Data Set Export* object to facilitate the export a specified migration request.
- The resulting file written to the export directory needs to be copied to the appropriate import directory.
- Import the data in the target region.
  - Use a *Migration Data Set Import* object to facilitate importing the data.
  - Review the changes detected by the import and compare process.
  - Apply approved changes.

The topics in this section describe specific information about migration plans and migration requests provided by the product.

Overview

To export data, there must be a migration request that references one or more migration plans. The migration plan is a set of instructions that defines the structure of an object to be exported. It may also contain instructions for foreign keys related to the object.

The definition of the migration plan for a given object and the instructions to include depend a little bit on the type of migration request that it will be included in. It also depends on whether or not the foreign key is a physical column in the object or if it is captured in the BO Data Area (CLOB) or as a flattened characteristic. The instructions for foreign keys serve two purposes:

- **Copy Related Data**. The instructions serves as an indication to the CMA tool that the related object represented by that foreign key should also be copied. For example, a migration plan for Tax Type may have an instruction for its Revenue Calendar. If the Tax Type migration plan is included in a migration request but the Revenue Calendar migration pan is not, the revenue calendars related to the migrated tax types will also be migrated. This is true whether the foreign key is linked as a real column or defined in the BO Data Area for one of the business objects for the migration plan's maintenance object.
- **Group Related Data**. For a migration request that includes a list of migration plans for different objects, the instructions help the CMA tool to group together objects in a logical 'transaction' for importing. For example, imagine the migration plan for Person Type contains an instruction for its default Identifier Type. If a migration request includes the migration plans for both Person Type and Identifier Type, the CMA tool will export all Person Types and Identifier Types. In addition because of the instruction on Person Type indicating that there is a relationship between person type and identifier type, the CMA tool will ensure that any Identifier Type defined on a Person Type will be grouped for import with its related person type. The result is that at import time the Identifier Type is added before the Person Type preventing any foreign key validation issues.

NOTE: The CMA tool does not require instructions for "real" columns to be able to group related data. This is because the tool can use the table / FK constraints to determine the dependencies. So for the examples above, if it is known that the Tax Type and Revenue Calendar migration plans will always be included in the same Migration Request and that the selection criteria for each plan will be "select all records", it is not necessary to define a sub-instruction in the Tax Type migration plan referencing Revenue Calendar. In this case, the CMA tool will select all tax types and all revenue calendars. Then it will use the constraints to understand relationships and will group any revenue calendar that is referenced by a tax type in the same transaction as the tax type.

Configuration migration assistant may be used for two broad types of data migration:

- **Wholesale migrations**. This is where an implementation is trying to "copy all" administrative data from one environment to another. For example, if trying to set up a test region that mirrors production in order to recreate a bug. Another example is to "promote" configuration changes from a test region to production after testing the new configuration. In these types of migrations, the migration plans for objects that include foreign key data in the BO Data Area, it is beneficial to have sub-instructions for all foreign keys. This will allow CMA to correctly group related data.
- **Piecemeal migrations**. This type of migration is more targeted and would only copy a subset of maintenance objects or a subset of data within a given set of maintenance objects. Some examples:
  - A set of new form types for the coming year are defined and their data should be copied to a test region.
  - The new dates for all the revenue calendars have been added for the coming year and should be copied to production.
  - A new calculation control is introduced with the changes for the coming year's property tax calculation rules and should be copied to a test region.

  For these types of migrations, it is beneficial to only include sub-instructions for the foreign keys that need to be included. For example, in the case of a new form type, if it is for an existing obligation type, there is no need to copy the obligation type and all the data that is related to that. Ideally, the migration should only copy the data that is expected to be new or changed as a result of the new form type.

Migration Plans

As described in the previous section, the drivers for which instructions to include in a migration plan differ a bit based on whether the migration plan will be included in a wholesale or a piecemeal migration request. The decision for the base product migration plans was to not provide multiple migration plans for various objects (one for wholesale migrations and one for piecemeal migrations). Rather, the base product provided migration plans supports a reasonable piecemeal migration. In general, sub-instructions are not provided for the foreign keys where there is a reasonable assumption that the data already exists in the target region. For example, the migration plan for any record that has a foreign key to Tax Type will not include a sub-instruction for tax type given that tax type is considered a type of “parent” configuration object that would be copied on its own to a region and not as part of the migration of a lower level entity.

NOTE: Potential issues in wholesale migrations. Because not all foreign key references that are part of base business objects in the BO Data Area are defined in the migration plans, it means that CMA will not know about the relationship between the record and its foreign key and will not know to group the data together. This could cause some “foreign key not found” errors on the import side. Users importing the data will then need to manage resolving the order of insertion of the new records in the target region.

To see the migration plans provided with the base product, navigate to **Admin Menu** > **Migration Plan** and view the data that is provided there.

The following points highlight some information about the base administrative objects and their migration plans.

- A small number of administrative maintenance objects include a Log table. Those MOs have been configured to not migrate the log entries (using the **Non-Migrated Table** MO option).
- The Form Change Reason maintenance objects has been configured to not migrate its related form types (using the **Non-Migrated Table** MO option).
- The following objects do not have their own migration plans but rather are included as a sub-instruction in a related object’s migration plan:
  - Adjustment Type Extension (part of the adjustment type migration plan)
  - Form Section (part of the form type migration plan)
  - Form Line (part of the form type migration plan)
  - Letter Template (part of the customer contact type migration plan)
  - Revenue Period (part of the revenue calendar migration plan)
- Tender source includes an optional reference to a suspense Obligation ID. CMA cannot copy master and transaction data. As a result, the base migration plan includes an import algorithm to not copy the Obligation ID from the source when adding or updating a tender source in the target region. Note that when the tender source already exists in the target region and has a reference to a valid Obligation ID from the target region, any updates to the tender source via CMA will not change the Obligation ID.
- Master configuration, service task type and extendable lookup are Framework owned objects, however special migration plans are provided in the base product to define foreign key references in the BO Data Area for some of the base business objects.
- Migration plans were not provided for any functionality that is considered “legacy” functionality and therefore not recommended to use going forward for new installations. For example, administrative data related to classic billing, legacy address, case type or DB processes are not provided.
- A migration plan is not provided for Activity Type. If an implementation uses this object and wishes to include it in a migration, a custom migration plan is needed.

- A migration plan is not provided for Collection Agency. This object comprises of a code, description and Person ID (to represent the agency). Because CMA cannot copy master or transaction data, and Person ID is a required field, this object cannot be copied. An implementation must manually create collection agency records in the target region.

Refer to the detailed descriptions of the base migration plans for more information.

NOTE: Base package migration plans cannot be altered. To extend an existing base package migration plan, the base package plan must be duplicated and then altered as needed. The duplicate migration plan can then be added to a custom migration request.

Migration Requests

The product provides out of the box migration requests. To see the migration requests provided with the base product, navigate to **Admin Menu** > **Migration Request** and view the data that is provided there. The following sections provide general information about the base provided migration requests.

Wholesale Migration

In order to do a full “copy all administrative data” migration, several migration requests are needed. The separate migration requests are used to minimize the dependencies that would be found when trying to copy all data in one request. The following are the recommended migration requests to export / import:

1. **F1–Languages**. Use this migration request to copy languages. Note that this is not necessary if the environments use only the ENG (English) language because this language is included as part of the product installation. It may also be simpler to go to the target region and define the languages manually assuming there are not very many languages supported. Note however that all languages defined in the source region must be defined in the target region if administrative data that will be copied includes language rows for that language.
2. **F1–SchemaAdmin**. This migration request copies the objects that are part of the building blocks of a schema.
3. **C1-FWOwnedAndCoreBaseAdmin**. This migration plan copies the remaining framework owned migration objects not already copied in previous migration requests along with base owned migration plans that are considered “core” and would have a lot of other objects that reference them. Note the following with respect to this migration request:
   - There are several configuration tool objects where only records with a CM owner flag are included in the migration. However for other configuration tool objects, all records are included to cater for the fact that the objects may have customizable columns or CM owned collection entries. The CMA tool will know to only update the CMable portions of base owned objects.
   - The migration request includes Rate Schedule, but it does not include RQ Rule. As a result, it specifically doesn’t include any rate schedules that reference an RQ rule.
4. **C1-BaseAdmin**. This migration request includes the remaining administrative tables. Note the following with respect to this migration request:
   - The Form Types copied as part of the request are only those in the status of **PENDING**, **GENERATED** or **ACTIVE**.
   - The base owned Master Configuration migration plan is included here. However, it specifically does not copy the SEPA Master Configuration record because that record references master data such as a Person and Address. If your implementation uses the SEPA Master Configuration, it must be defined manually in the target region.
   - The entries for Rate Version and Rate Components specifically do not include records related to a rate schedule that references an RQ rule.

Piecemeal Migration

The base product provides a migration request to help orchestrate a targeted copy of Form Data. The **C1–FormData** migration request includes all administrative objects related to managing form data. An implementation may use the migration request as is to copy all form data from one region to another. Otherwise, it can use this migration request as a template for creating custom migration requests that may include specific selection criteria.

Also note that the **C1–FormType** migration plan has sophisticated instructions so that it may be used in a special migration request to copy one or more form types and their related data. The product has not provided a migration request for this migration plan because the selection criteria would differ for each migration. To create a specific migration request for one or more form types, follow these steps:

1. Navigate using

   Admin Menu > **Migration Request** in Add mode.
2. Define an appropriate Migration Request code and Description.
3. Define the **C1–FormType** migration plan.

   Use the Selection Type to define a **Specific Key** when attempting to migrate one or more form types and their data by listing the form type codes. Or use a Selection Type of **SQL Statement** if there are multiple form types to select and the selection should be by something other than the code (such as all the form types for a given tax type or all form types effective on or after a given date).
4. Save the record and proceed to the export and import steps.

Chapter 28

The Conversion Process

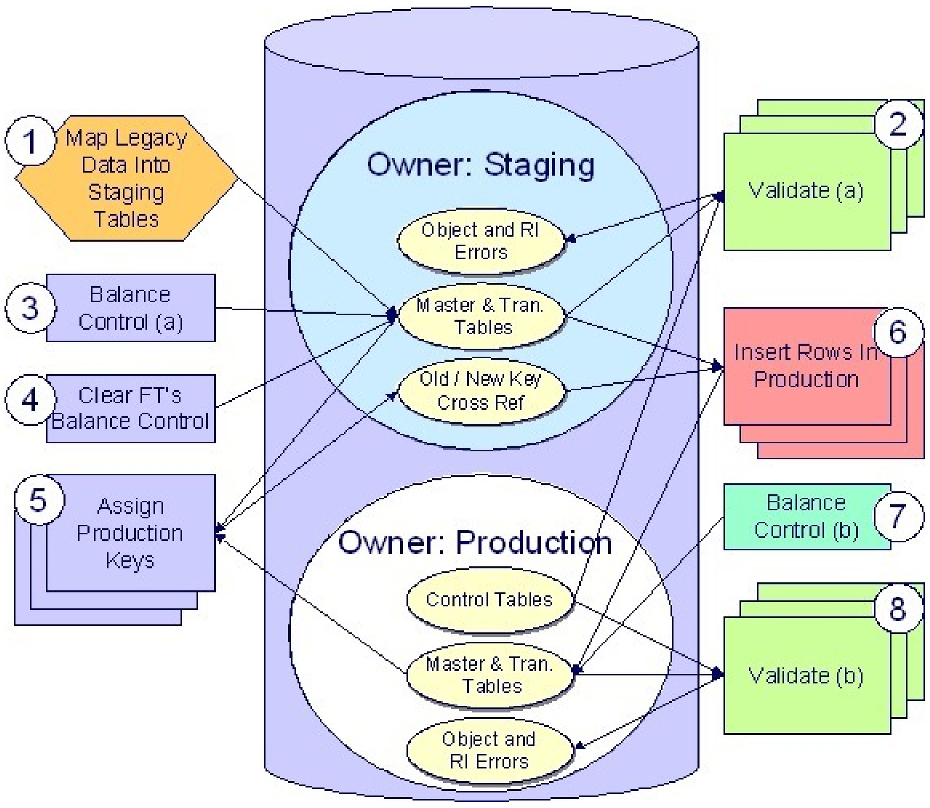
This document describes the Oracle Public Sector Revenue Management conversion process.

Introduction

When you're ready to convert data from your legacy system into Oracle Public Sector Revenue Management, you will have analyzed your requirements according to your business and organizational needs and set up the control tables accordingly.

FASTPATH: 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 and tax role) and transaction data (e.g., adjustment, tax forms) 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.

  CAUTION: 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, location, 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.

  NOTE: 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.

- **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 Process Steps

The following sections provide more details about the steps in the conversion process.

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.

NOTE: **Recommendation.** You can use any method you prefer to load Oracle Public Sector Revenue Management 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 they 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 account in a ConfigLab environment and we don't want its key to be allocated to an account added in production.

- Key tables only have two columns: 1) The key of the object and 2) 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 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_TAX_ROLE 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 number. 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, etc.) and a limited number of transaction data objects. 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 topics in this section describe how to execute the 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 the same objects you may also be converting historic records. You may not want to perform validation on completed records. To support this, some background processes provided for transaction data allow you to limit the validation to records in a given 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 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.

CAUTION: **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.

Referential Integrity Validation Programs

It's important to understand that only master data objects (e.g., persons, accounts, assets, locations, 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.

NOTE: **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.

CAUTION: 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.

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.

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:

- Gather Schema statistics using the DBMS_STATS PL/SQL package after data has been inserted into the staging tables. Collecting statistics at the schema level is more efficient than object by object. Be sure to specify the ‘degree’ parameter to enforce parallel statistics collection.
- *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. Monitor system resources (CPU, RAM, IO) to fine tune how many jobs can be run simultaneously.
- Remember that the *object validation programs* can be run in "validate every n^{th} row". We recommend running these programs using a relatively large value for this parameter until the pervasive problems have been rectified.

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.

NOTE: 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.

NOTE: Submitting this process. You submit this process in the staging database. For a detailed description of this batch control and its parameters, refer to the Batch Control page in the application or view this batch control in the application viewer.

Allocate Production Keys

The topics in this section describe the background processes used to assign production keys to the staging data.

The Big Picture of Key Assignment

It's important to understand that the system does not overwrite the prime-keys or foreign 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_TAX_ROLE 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.

NOTE: 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).

NOTE: Iterative conversions. Rather than perform a "big bang" conversion (one where all taxpayers are populated at once), some implementations have the opportunity to go live on subsets of their taxpayer 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 taxpayers, you can be assured of getting clean keys. Note that iterative conversions of tax types for existing taxpayers is not currently supported.

NOTE: Optional Foreign Keys. One responsibility of the key allocation programs is to generate a single "zero" entry in its key table to represent a blank foreign key. This step is required by the subsequent Insert programs so that they may correctly insert new rows where optional foreign keys are blank. For example, the CI_FT table has a foreign key to BILL_ID. Conversion for classic billing is no longer supported. However, the key generation program for Bills must still be run in order to generate the entry in the CK_BILL table for the "zero" row. This is needed by the FT Insertion program. The details of which Key Generation programs are required for successful run of a given insert program are detailed in each staging table section.

NOTE: 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.

> **NOTE: 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.

> **NOTE: 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*.

- Gather table statistics after each key generation program runs and before running key generation programs for the next level. Table 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.
- Key generation programs are single threaded and use set-based SQL. Consider running key generation jobs from the same level (see *Program Dependencies*) in parallel. Monitor system resources (CPU, RAM, IO) to fine tune how many jobs can be run simultaneously
- Optimal use of the **Mode** parameter under *Submitting Key Assignment Programs*.
  - Before any key assignments, drop both the “old key” CX_ID index and the “new key” CI_ID index on the CK_<table_name> tables. Make sure you save the index DDLs before dropping the indexes. You can use the script provided in *Appendix D* to save the index DDLs. The script creates another script called ***create_ck_index.sql***, which can be used to recreate the indexes in a subsequent step.
  - Run all *key assignment tiers*, submitting each job with MODE = "I".
  - Recreate the CX_ID and CI_ID indexes on the CK_<table_name>, using the ***create_ck_index.sql*** script that was created previously. This script uses parallelism and no logging options to speed up the index creation. After the indexes are created, this script will change the parallelism to 1 and will enable the logging. 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.
  - Do not run key generation jobs from different tiers simultaneously. It is OK to run key generation programs from the same tier in parallel. See *Program Dependencies* for a description of the tiers.

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.

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").
  - Alter all indexes except primary key and unique indexes on the production tables being inserted into to be unusable.
  - Gather statisticson the old key/new key table(s) using the DBMS_STATS PL/SQL package.
- After the insertion programs have populated production data,

- Rebuild the indexes on the production tables.
- Gather statistics using the DBMS_STATS PL/SQL package after insertion programs have completed and the indexes have been rebuilt.
- Validate the production schema to ensure that all schema changes made during conversion have been reversed properly.

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.

NOTE: **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.

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 location 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 table and foreign key error combination.

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.

NOTE: Most foreign keys on a table are obvious based on the field name. For example, if the output shows Table Name **CI_ADJ**, Count **100**, Foreign Key Field Name 1 **ADJ_TYPE_CD**, and Foreign Key Value 1 **SALES_INTEREST**, you know that there are 100 records in the Adjustment table referencing an invalid Adjustment Type Code of **SALES_INTEREST**. However, some foreign keys are multi-part keys and in viewing the Foreign Key names, the table they refer to may not be obvious. If it is not clear what table a foreign key is referring to, navigate to the Table page for the table reported in the error and view the Constraints tab. All the foreign keys being checked are listed there with their respective table names.

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 tables record is related. For example, the CI_ACCT_CHAR record identified by its displayed primary keys should be related to an Account record with the Account 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.

CAUTION: Recommendation. We recommend you read this document on a browser (or using Word under windows) so you can take advantage of the *Color Coding*.

NOTE: 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.

CAUTION: 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.

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:

- For maintenance objects that require *object validation*, an 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 separate *referential integrity batch validation* program exists for each table that requires one. 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 etc.) that must be created before you can convert transaction data.

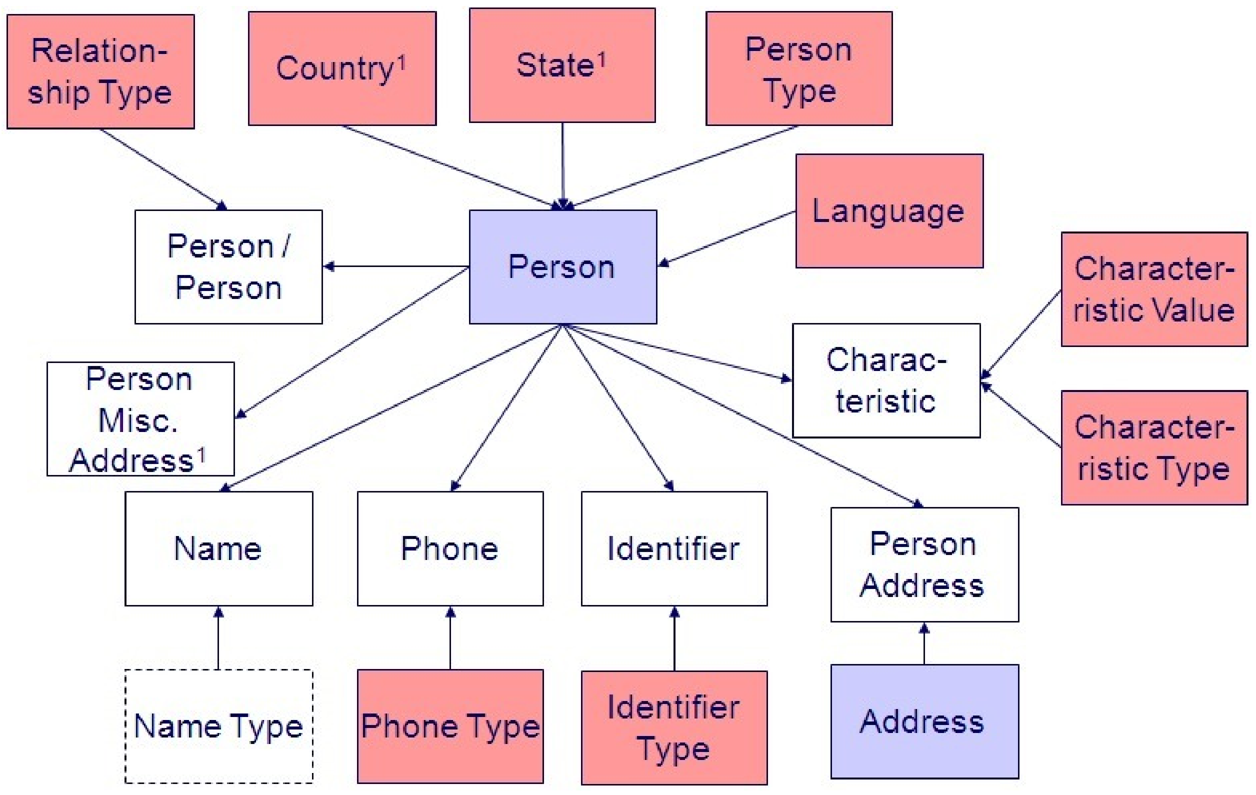
NOTE: Key Assignment Dictates The Order Of Conversion. The following topics are listed in the order in which the objects should be converted in order to maintain referential integrity.

Person

This section describes the person object.

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

Address	*CI_PER_ADDR*	No. The key is PER_ID plus a sequence number.	CIPVPADV	CIPVPADI
Miscellaneous Address[1]	*CI_PER_ADDR_SEAS*	No. The key is PER_ID plus a sequence number.	CIPVPSAV	CIPVPSAI

NOTE: **Note 1.** Country, State and the Person Miscellaneous Addresses are only supported for legacy addresses. Refer to *Address Support* for more information.

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.

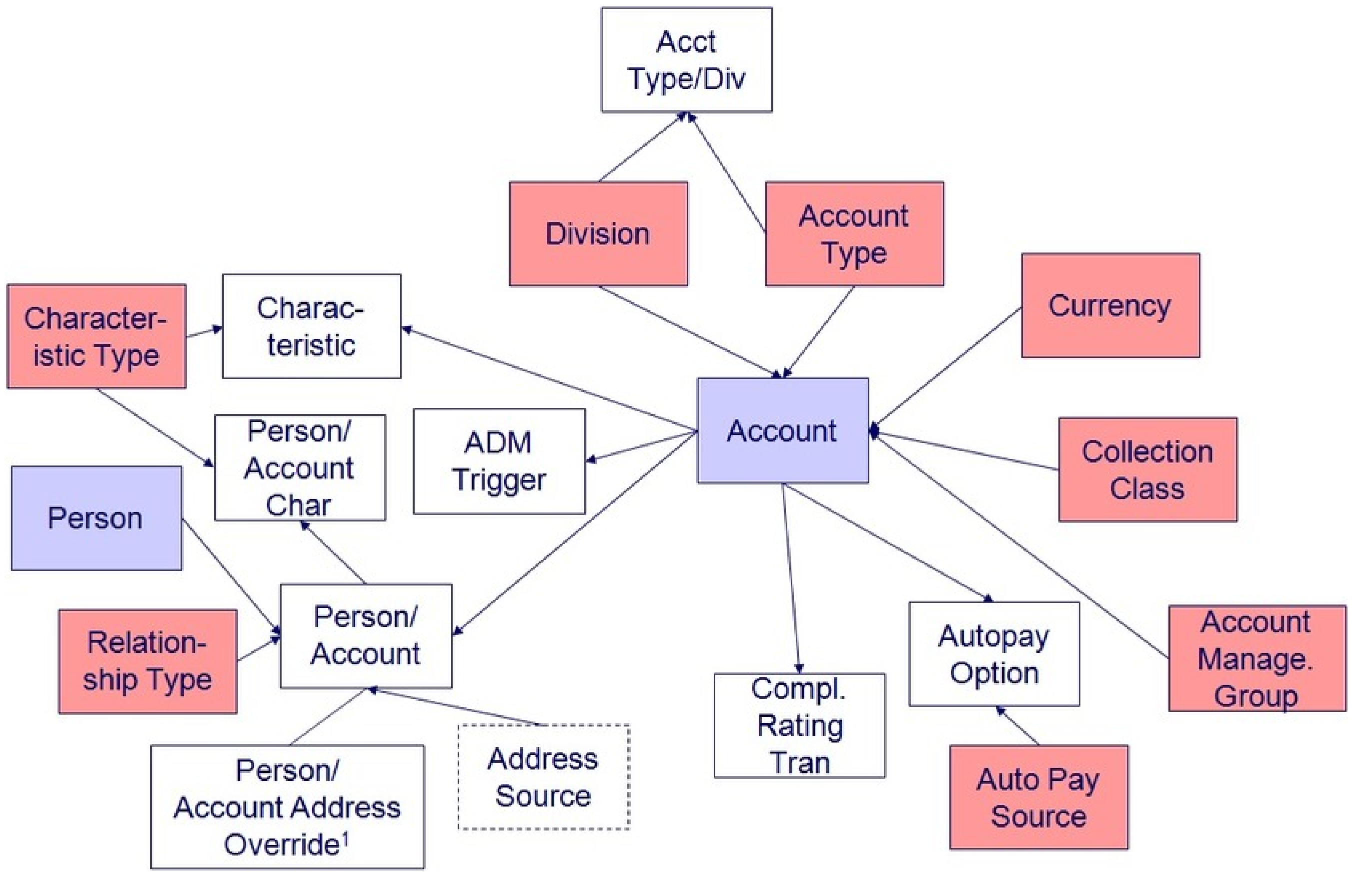
This maintenance object includes a character large object field that your organization may be using to capture implementation specific data as defined by your business objects. For records of this type, the process to insert the records to the staging table is responsible for populating the data in this CLOB as per the record's business object schema.

Account

This section describes support for converting the account object.

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 See Optional FK Note below.
Auto pay 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 Char	*CI_ACCT_PER_CHAR*	No. The key is account ID plus person ID plus an		CIPVAPCV		CIPVAPCI

		edate and a char type.			
Person/Account Address Override[1]	*CI_PER_ADDR_OVRD*	No. The key is Account ID plus Person ID	CIPVPAOV		CIPVPAOI
Compliance Rating Transaction	*CI_CR_RAT_HIST*	Yes *CI_CR_RAT_HIST_K*	CIPVCRTV	CIPVCRRK *Has dependencies*	CIPVCRTI
ADM Trigger	*CI_ADM_RVW_SCH*	No. The key is account ID plus date	CIPVARSV		CIPVARSI

NOTE: **Note 1.** Mailing Location and the Person / Account Override Addresses are only supported for legacy addresses. Refer to *Address Support* for more information.

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 taxpayer. Please see column notes for the account / person table for inter-field validation in respect of the various switches (e.g., if main taxpayer 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 *overdue 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 compliance rating, you should create compliance rating transactions. The values you create need to be consistent with the base and threshold compliance rating on the installation record. Refer to the account user documentation for more information.

This maintenance object includes a character large object field that your organization may be using to capture implementation specific data as defined by your business objects. For records of this type, the process to insert the records to the staging table is responsible for populating the data in this CLOB as per the record's business object schema.

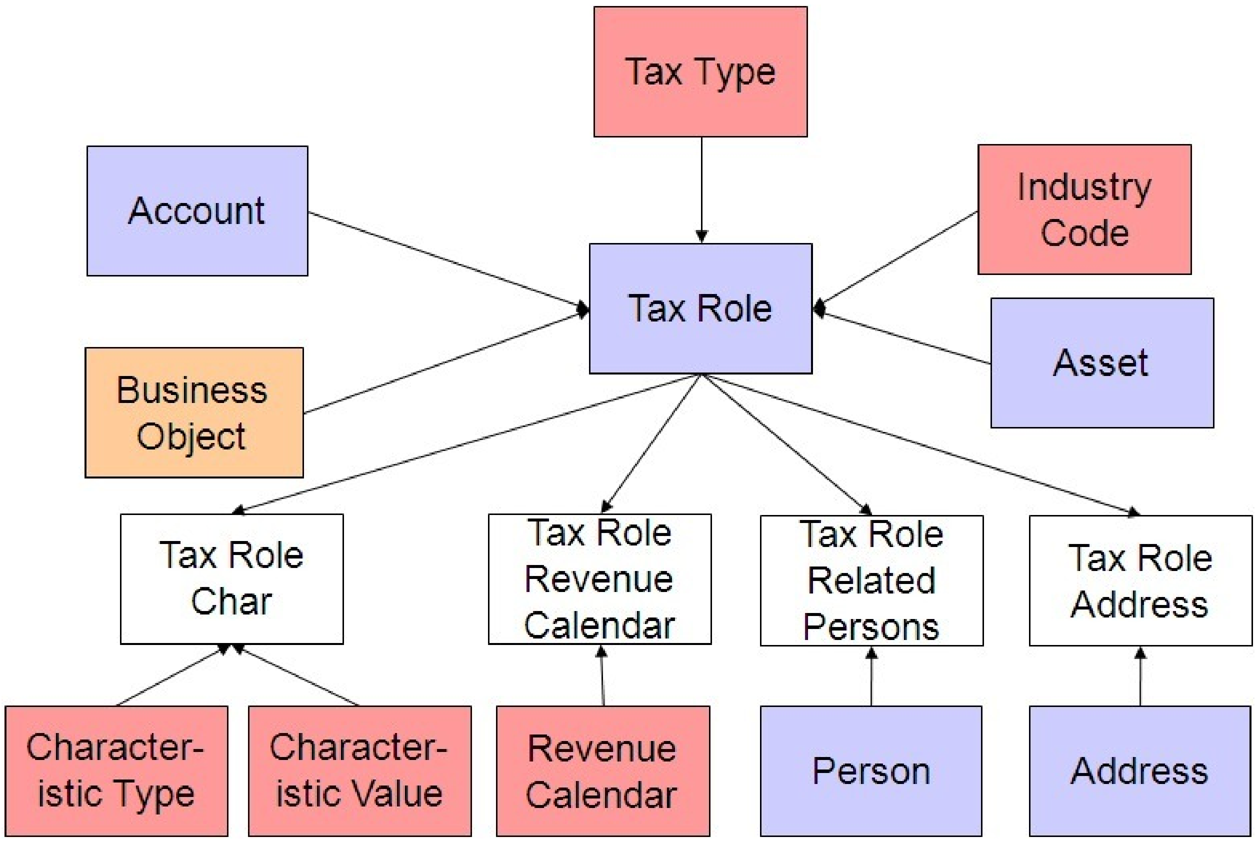
Optional FK Note

Account has an optional foreign key MAILING_PREM_ID to the Location (CI_PREM) table. Conversion for locations is no longer supported. However, the key generation program for Location (CIPVPRMK) must still be run. Refer to *The Big Picture of Key Assignment* for more information.

Tax Role

Tax Role Data Model

The following data model illustrates the tax role object.



Tax Role 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***
Tax Role	*CI_TAX_ROLE*	Yes. *CI_TAX_ROLE_K*	VAL-TAXR	CIPVTXRV	CIPVTXRK *Has dependencies*	CIPVTXRI
Tax Role Revenue Calendar	*CI_TAX_ROLE_CAL*	No. The key is Tax Role ID and Effective Date.		CIPVTRCV		CIPVTRCI
Tax Role Address	*CI_TAX_ROLE_ADDR*	No. The key is Tax Role ID and Sequence Number.		CIPVTRAV		CIPVTRAI
Tax Role Related Persons	*CI_TAX_ROLE_PER*	No. The key is Tax Role ID, Relationship Type, Start Date and Person ID.		CIPVTXPV		CIPVTXPI
Tax Role Characteristic	*CI_TAX_ROLE_CHAR*	No. The key is Tax Role ID, Characteristic		CIPVTXCV		CIPVTXCI

		Type, and Effective Date.				

Tax Role Suggestions

This maintenance object includes a character large object field that your organization may be using to capture implementation specific data as defined by your business objects. For records of this type, the process to insert the records to the staging table is responsible for populating the data in this CLOB as per the record's business object schema.

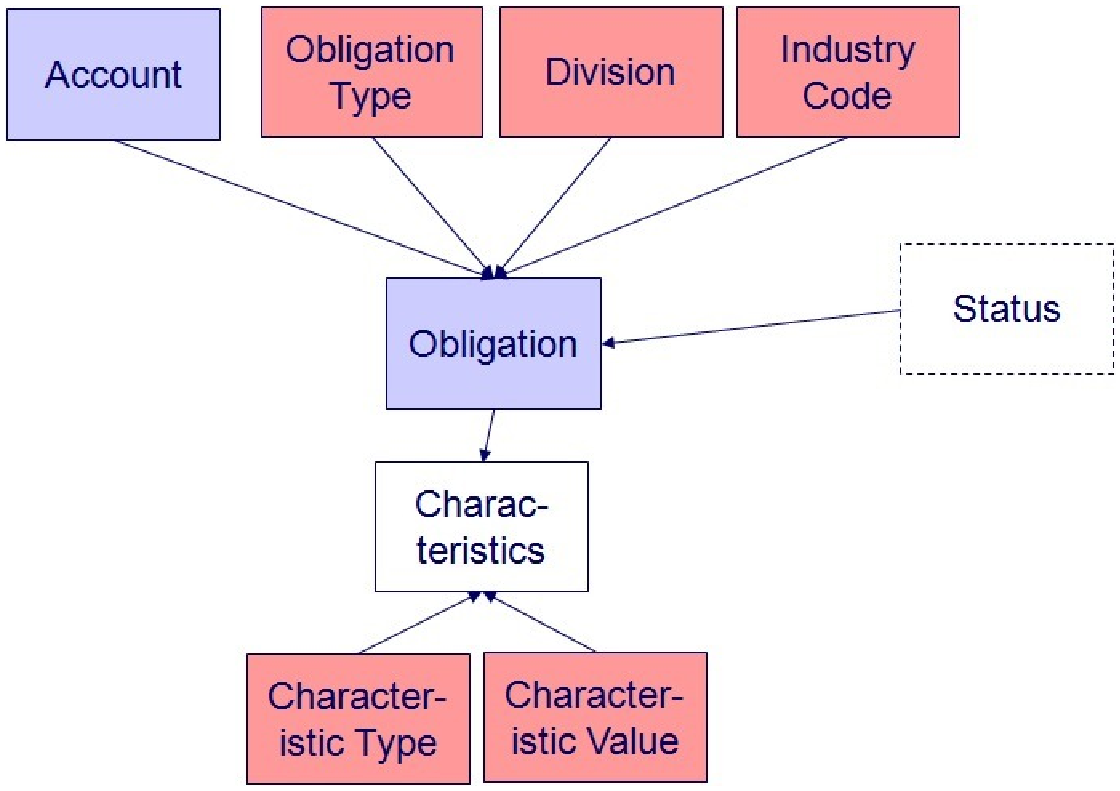
Optional FK Note

Tax Role has an optional foreign key ASSET_ID to the Asset (CI_ASSET) table. Implementations that are not planning to convert Assets must still run the key generation process for Assets. Refer to *The Big Picture of Key Assignment* for more information.

Obligation

Obligation Data Model

The following data model illustrates the obligation object.



Obligation 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*

Obligation	*CI_SA*	Yes *CI_SA_K*	VAL-OBLG		CIPVSVAK *Has dependencies*	CIPVSVAI See Optional FK Note below.
Characteristic	*CI_SA_CHAR*	No. The key is obligation ID plus an edate and a char type.		CIPVSACV		CIPVSACI

Obligation Suggestions

This maintenance object includes a character large object field that your organization may be using to capture implementation specific data as defined by your business objects. For records of this type, the process to insert the records to the staging table is responsible for populating the data in this CLOB as per the record's business object schema.

Optional FK Note

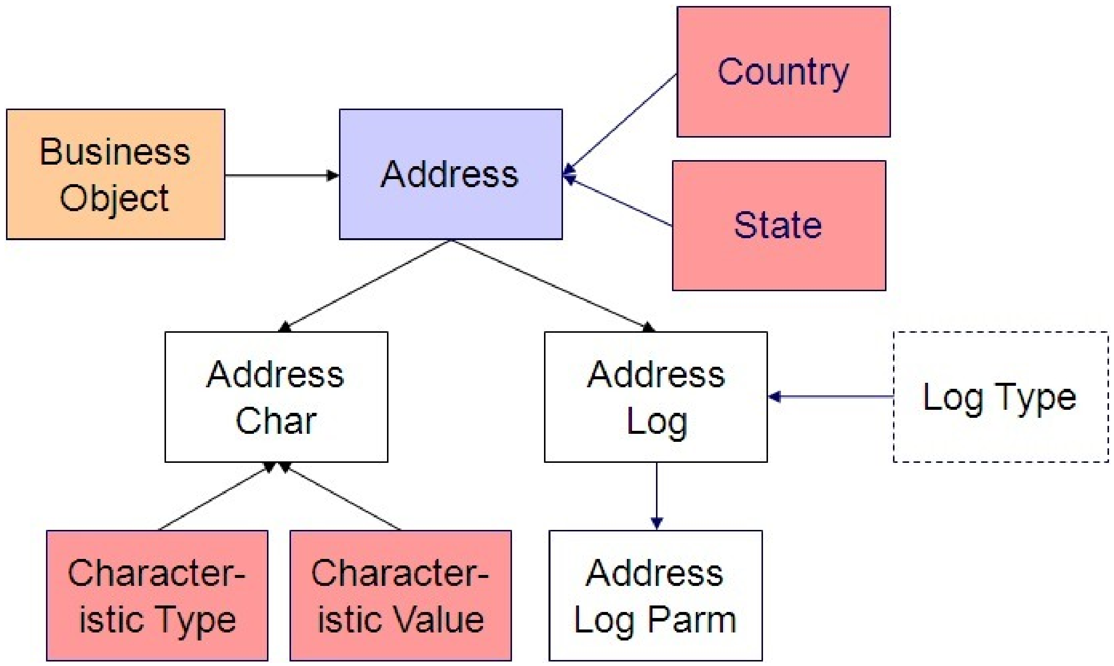
Account has an optional foreign key CHAR_PREM_ID to the Location (CI_PREM) table. Conversion for Locations is no longer supported. However, the key generation program for Location (CIPVPRMK) must still be run. Refer to *The Big Picture of Key Assignment* for more information.

Account has an optional foreign key TAX_ROLE_ID to the Tax Role (CI_TAX_ROLE) table. If your implementation is planning to convert obligations, but not tax roles, the key generation program for Tax Role must still be run. Refer to *The Big Picture of Key Assignment* for more information.

Address

Address Data Model

The following data model illustrates the Address object.



Address 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*
Address	*C1_ADDRESS*	Yes. *C1_ADDRESS_K*	VAL-AIN	CIPVAINV	CIPVAINK	CIPVAINI
Address Characteristic	*C1_ADDRESS_CHAR*	No. The key is Address ID, Characteristic Type, and Sequence.		CIPVAICV		CIPVAICI
Address Log	*C1_ADDRESS_LOG*	No. The key is Address ID and Sequence.		CIPVAILV		CIPVAILI
Address Log Parameter	*C1_ADDRESS_LOG_PARM*	No. The key is Address ID, Sequence, and Message Parameter Sequence.		CIPVAIMV		CIPVAIMI

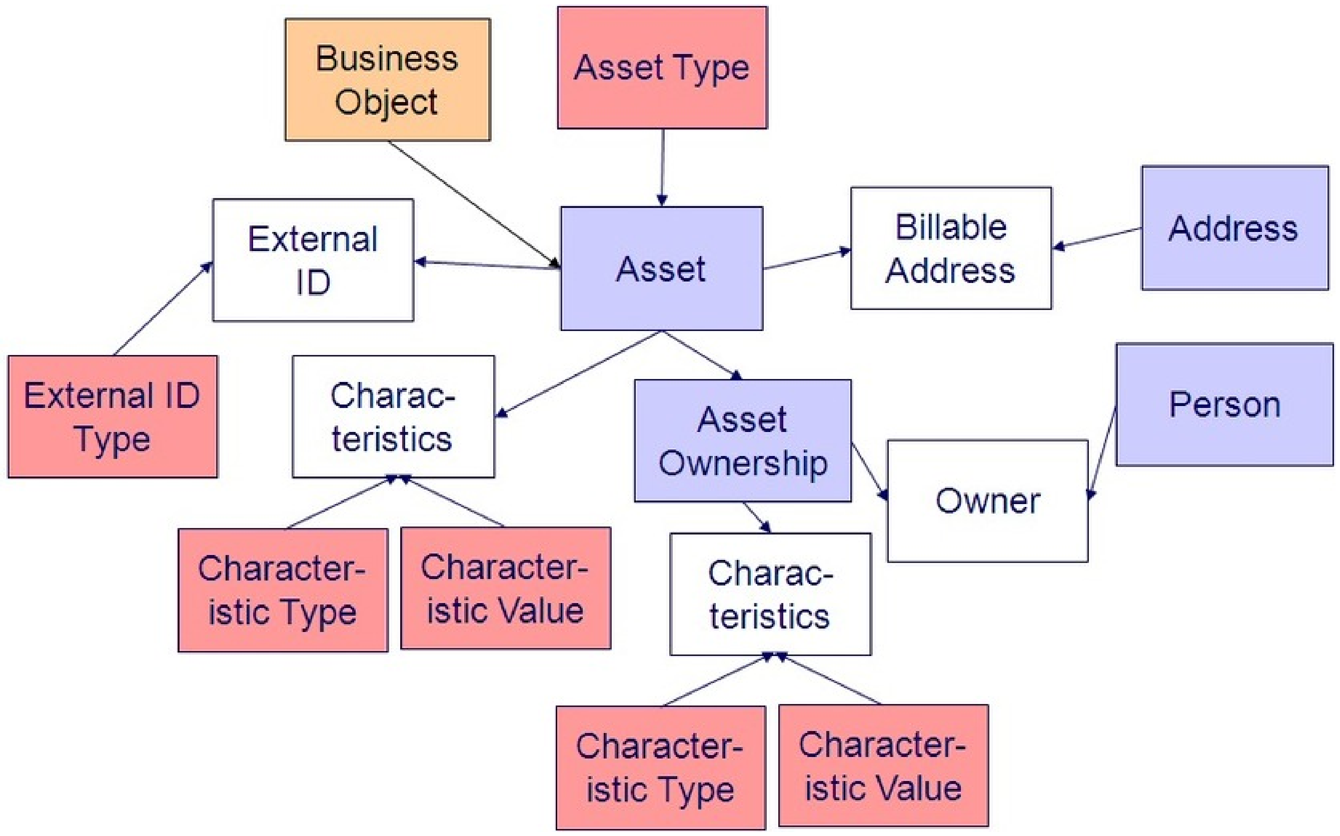
Address Suggestions

This maintenance object includes a character large object field that your organization may be using to capture implementation specific data as defined by your business objects. For records of this type, the process to insert the records to the staging table is responsible for populating the data in this CLOB as per the record's business object schema.

Asset

Asset Data Model

The following data model illustrates the asset object.



Asset 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*
Asset	*CI_ASSET*	Yes. *CI_ASSET_K*	VAL-ASST	CIPVASTV	CIPVASTK	CIPVASTI
Asset Billing Address	*CI_ASSET_BILLING_ADDR*	No. The key is Asset ID, Sequence Number.		CIPVASBV		CIPVASBI
Asset External ID	*CI_ASSET_EXT_ID*	No. The key is Asset ID, Sequence Number.		CIPVASEV		CIPVASEI
Asset Related Persons	*CI_ASSET_PER*	No. The key is Asset ID, Sequence Number.		CIPVASRV		CIPVASRI
Asset Characteristic	*CI_ASSET_CHAR*	No. The key is Asset ID, Characteristic Type, and		CIPVASCV		CIPVASCI

		Sequence Number.				
Asset Log	*CI_ASSET_LOG*	No. The key is Asset ID, Sequence Number.		CIPVASLV		CIPVASLI
Asset Log Parameter	*CI_ASSET_LOG_PARM*	No. The key is Asset ID, Sequence, and Message Parameter Sequence.		CIPVASPV		CIPVASPI
Asset Ownership	*C1_ASSET_OWN*	Yes. *CI_ASSET_OWN_K*	VAL-AOWN	CIPVAOWV	CIPVAOWK *Has dependencies*	CIPVAOWI
Asset Owners	*C1_ASSET_OWN_OWNER*	No. The key is Asset Ownership ID and Sequence Number.		CIPVAOOV		CIPVAOOI
Asset Ownership Characteristic	*C1_ASSET_OWN_CHAR*	No. The key is Asset Ownership ID, Characteristic Type, and Sequence Number.		CIPVAOCV		CIPVAOCI
Asset Ownership Log	*C1_ASSET_OWN_LOG*	No. The key is Asset Ownership ID, Sequence Number.		CIPVAOLV		CIPVAOLI
Asset Ownership Log Parameter	*C1_ASSET_OWN_LOG_PARM*	No. The key is Asset Ownership ID, Sequence, and Message Parameter Sequence.		CIPVAOPV		CIPVAOPI

Asset Suggestions

This maintenance object includes a character large object field that your organization may be using to capture implementation specific data as defined by your business objects. For records of this type, the process to insert the records to the staging table is responsible for populating the data in this CLOB as per the record's business object schema.

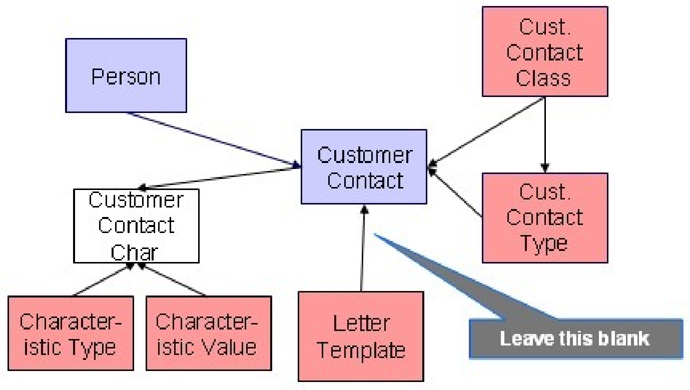
Transaction Data

This section describes the tables in which your transaction data (e.g., tax forms, payments, customer contacts, etc.) resides.

Customer Contact

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
Characteristics	*CI_CC_CHAR*	No. The key is customer contact ID plus char type code	CIPVCCTV		CIPVCCTI

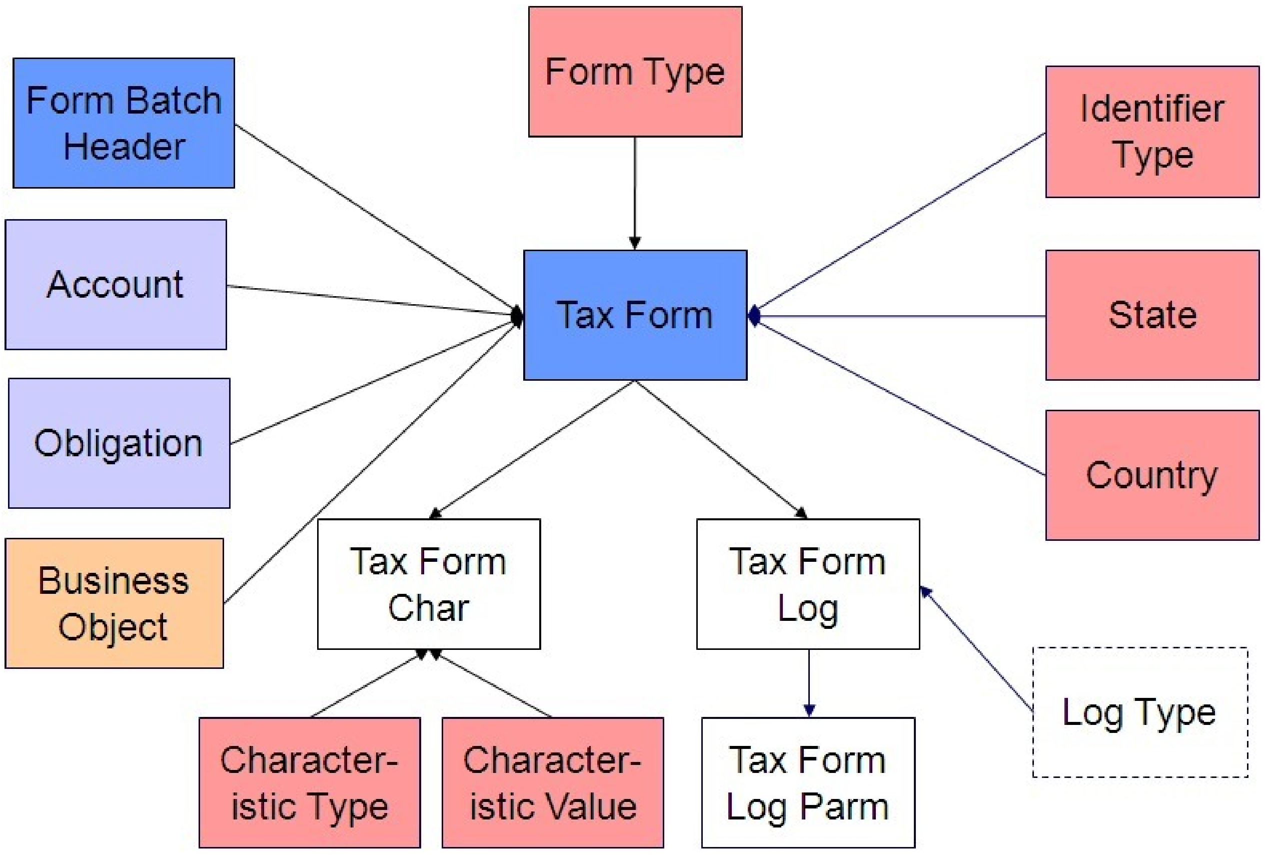
Customer Contact Suggestions

This maintenance object includes a character large object field that your organization may be using to capture implementation specific data as defined by your business objects. For records of this type, the process to insert the records to the staging table is responsible for populating the data in this CLOB as per the record's business object schema.

Tax Form

Tax Form Data Model

The following data model illustrates the Tax Form object.



Tax Form 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*
Tax Form	*CI_TAX_FORM*	Yes. *CI_TAX_FORM_K*	VAL-TXFR		CIPVTXFK	CIPVTXFI
Tax Form Characteristic	*CI_TAX_FORM_CHAR*	No. The key is Tax Form ID, Characteristic Type, and Sequence.		CIPVTFCV		CIPVTFCI
Tax Form Log	*CI_TAX_FORM_LOG*	No. The key is Tax Form ID and Sequence.		CIPVTFLV		CIPVTFLI
Tax Form Log Parameter	*CI_TAX_FORM_LOG_PARM*	No. The key is Tax Form ID, Sequence, and Message Parameter Sequence.		CIPVTLPV		CIPVTLPI

Tax Form Suggestions

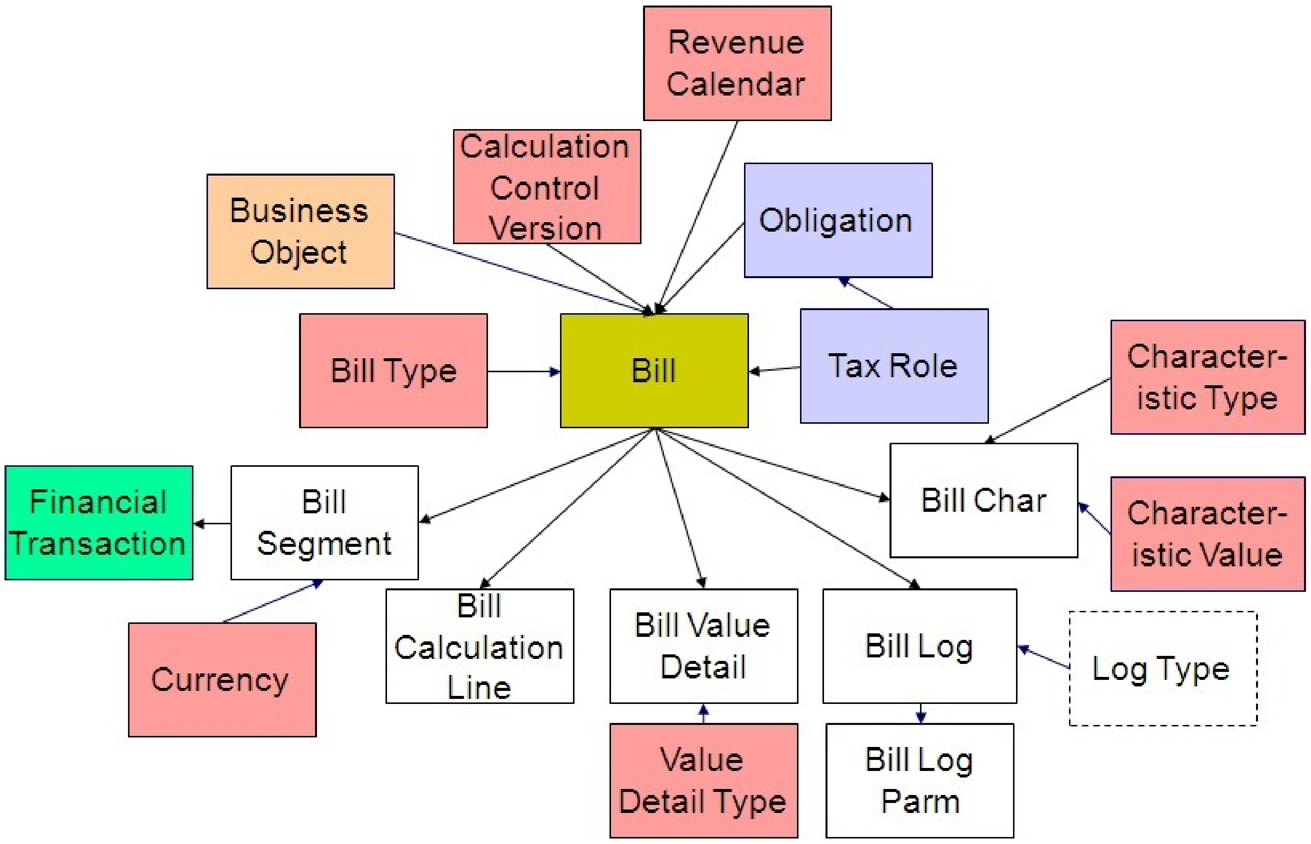
This maintenance object includes a character large object field that your organization may be using to capture implementation specific data as defined by your business objects. For records of this type, the process to insert the records to the staging table is responsible for populating the data in this CLOB as per the record's business object schema.

Bill

Bill Data Model

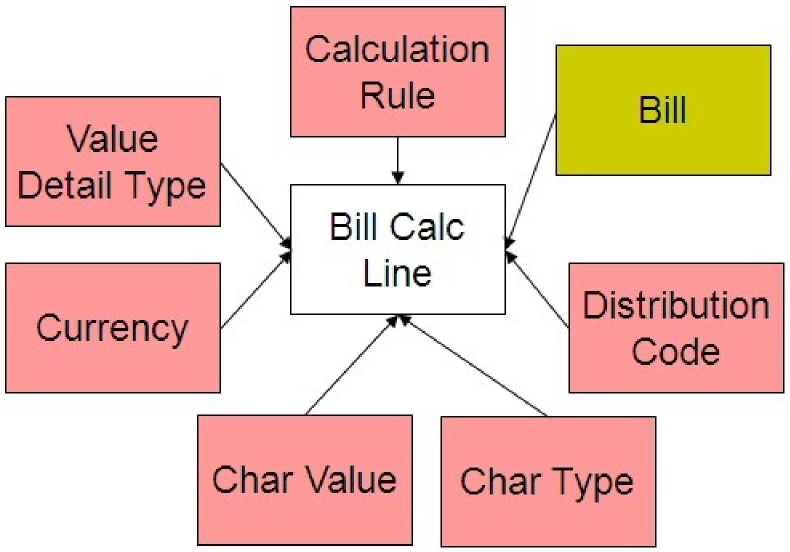
Bill - Main

The following data model illustrates the bill object.



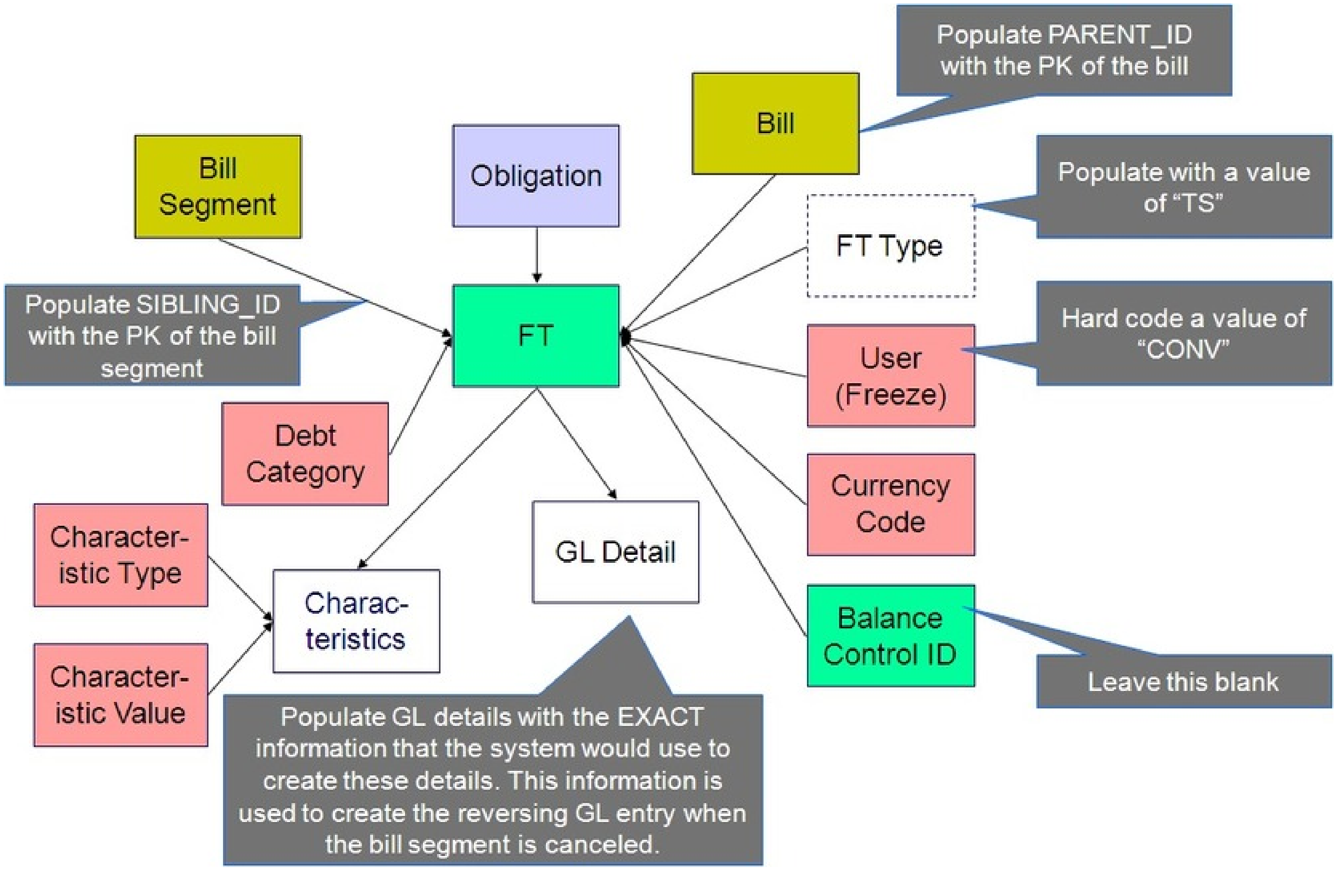
Bill - Calculation Line

The following data model illustrates the bill calculation line and important references.



Bill - FT

The following data model illustrates the FT that must be associated with a bill segment.


Populate PARENT_ID with the PK of the bill
Bill
Bill Segment
Obligation
Populate with a value of "TS"
FT Type
Populate SIBLING_ID with the PK of the bill segment
FT
Hard code a value of "CONV"
User (Freeze)
Debt Category
Currency Code
Character-istic Type
GL Detail
Charac-teristics
Balance Control ID
Leave this blank
Character-istic Value
Populate GL details with the EXACT information that the system would use to create these details. This information is used to create the reversing GL entry when the bill segment is canceled.

Bill Table Names

Data Model Name	*Table Name*	*Generated Keys*	*Referential Integrity Validation Batch Control*	*Key Assignment Batch Control*	*Insertion Batch Control*

Bill	C1_TAX_BILL	Yes C1_TAX_BILL_K	CTPVTBIV	CTPVTBIK Has dependencies	CTPVTBII
Bill Calculation Line	C1_TAX_BILL_CALC_LN	No. The key is bill ID plus sequence number.	CTPVTBCV		CTPVTBCI
Bill Segment	C1_TAX_BILL_SEG	Yes C1_TAX_BILL_SEG_K	CTPVTBSV	CTPVTBSK Has dependencies	CTPVTBSI
Bill Value Detail	C1_TAX_BILL_VAL_DTL	No. The key is bill id and a sequence number.	CTPVTBDV		CTPVTBDI
Bill Characteristics	C1_TAX_BILL_CHAR	No. The key is bill segment id, the char type, and a sequence number.	CTPVTBHV		CTPVTBHI
Bill Log	C1_TAX_BILL_LOG	No. The key is bill id and a sequence number.	CTPVTBLV		CTPVTBLI
Bill Log Parameter	C1_TAX_BILL_LOG_PARM	No. The key is bill id, sequence number and a parm sequence number.	CTPVTBPV		CTPVTBPI
FT (financial transaction)	CI_FT	Yes CI_FT_K	CIPVFTFV	CIPVFTXK Has dependencies	CIPVFTFI
FT Characteristic	CI_FT_CHAR	No. The key is FT id, char type code and a sequence number	CIPVFTCV		CIPVFTCI
FT GL (FT general ledger)	CI_FT_GL	No. The key is FT id and a GL sequence number	CIPVFTGV		CIPVFTGI

Bill Suggestions

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"

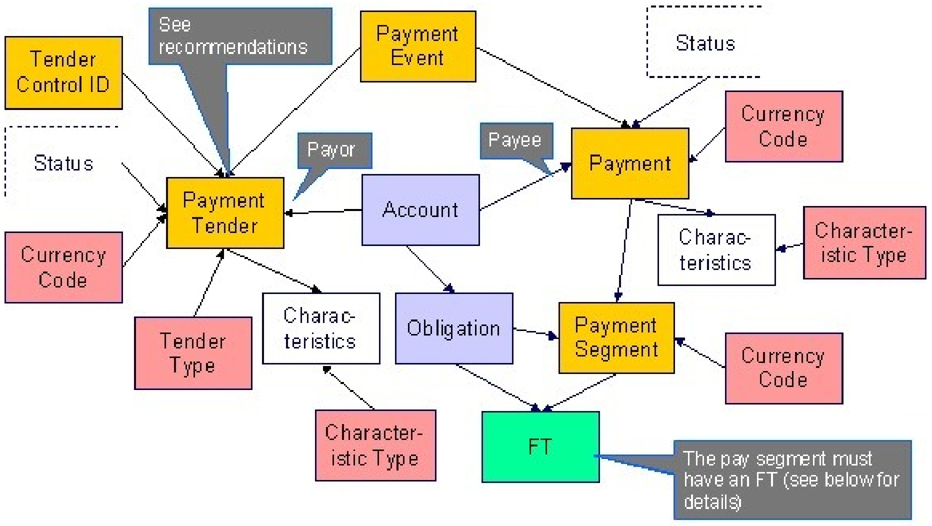
- ACCOUNTING_DT should be set to the current date
- SCHED_DISTRIB_DT should be left blank
- CURRENCY_CD should be the currency on the installation record
- BAL_CTL_GRP_ID should be left blank
- XFERRED_OUT_SW should be set to "Y"
- PARENT_ID should be set to the bill ID
- SIBLING_ID should be set to 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 the system would use to create them. This information is used to create the reversing GL entry when the bill segment is canceled.

Payment

Payment Data Model

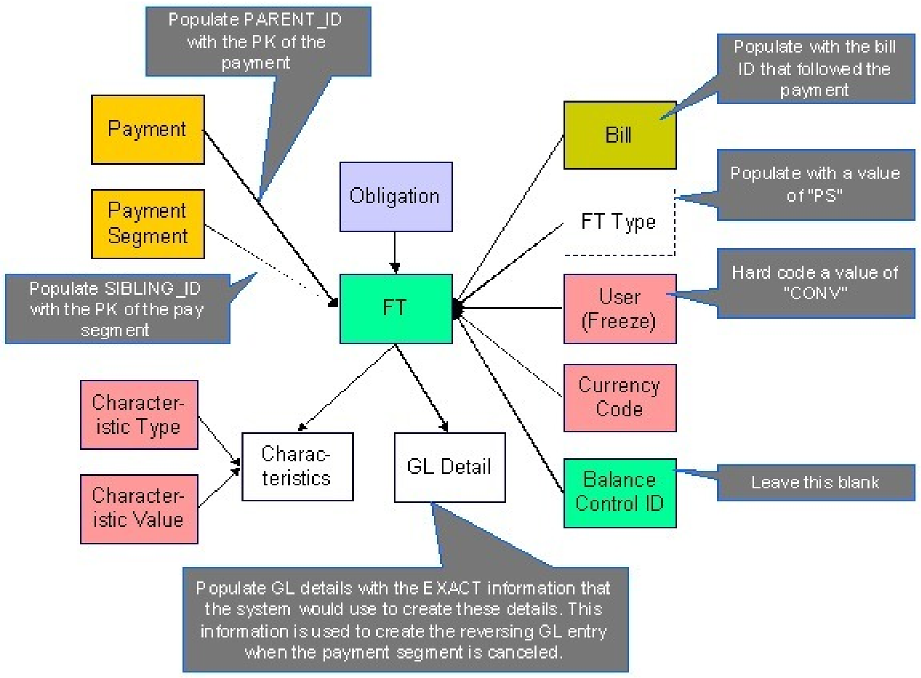
Payment - Main

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.



Populate PARENT_ID with the PK of the payment
Populate with the bill ID that followed the payment
Payment
Bill
Populate with a value of "PS"
Obligation
Payment Segment
FT Type
Populate SIBLING_ID with the PK of the pay segment
FT
User (Freeze)
Hard code a value of "CONV"
Currency Code
Character-istic Type
Charac-teristics
GL Detail
Balance Control ID
Leave this blank
Character-istic Value
Populate GL details with the EXACT information that the system would use to create these details. This information is used to create the reversing GL entry when the payment segment is canceled.

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 Characteristic	*CI_PAY_CHAR*	No. The key is PAY_ ID, plus a sequence number and a char type	CIPVPYCV		CIPVPYCI
Payment Event	*CI_PAY_EVENT*	Yes *CI_PAY_ EVENT_K*		CIPVPYEK *Has dependencies*	CIPVPYEI
Payment Tender	*CI_PAY_TNDR*	Yes *CI_PAY_TNDR_ K*	CIPVTNDV	CIPVTNDK *Has dependencies*	CIPVTNDI
Payment Tender Characteristic	*CI_PAY_TNDR_ CHAR*	No. The key is PAY_ TENDER_ID, plus a sequence number and a char type	CIPVTNCV		CIPVTNCI
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 See Optional FK Note below.
FT Characteristic	*CI_FT_CHAR*	No. The key is FT id, char type code and a sequence number	CIPVFTCV		CIPVFTCI

FT GL (FT general ledger)	*CI_FT_GL*	No. The key is FT id and a GL sequence number	CIPVFTGV	CIPVFTGI

Payment Suggestions

The SEQ_NUM field on the payment should be left blank. This is part of the primary key for the Payment Event Distribution Details, which are not part of the conversion process.

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 the system would use to create them. This information is used to create the reversing GL entry when the payment segment is canceled.

Optional FK Note

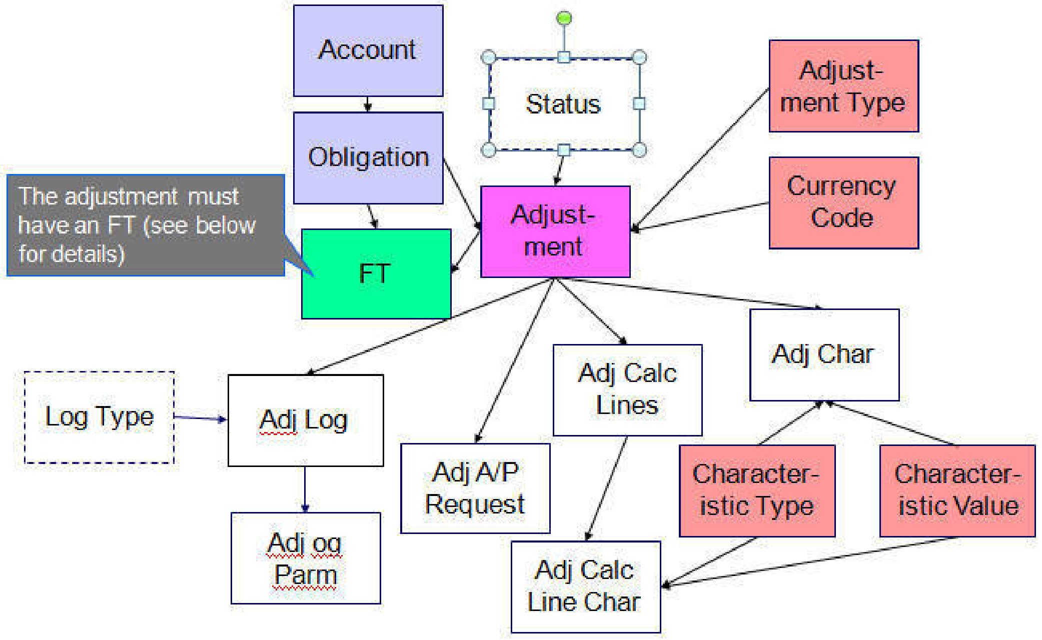
FTs for payments have an optional foreign key BILL_ID to the Bill (Classic) table (CI_BILL). Conversion for classic billing locations is no longer supported. However, the key generation program for Bill (CIPVBILK) must still be run. Refer to *The Big Picture of Key Assignment* for more information.

Adjustment

Adjustment Data Model

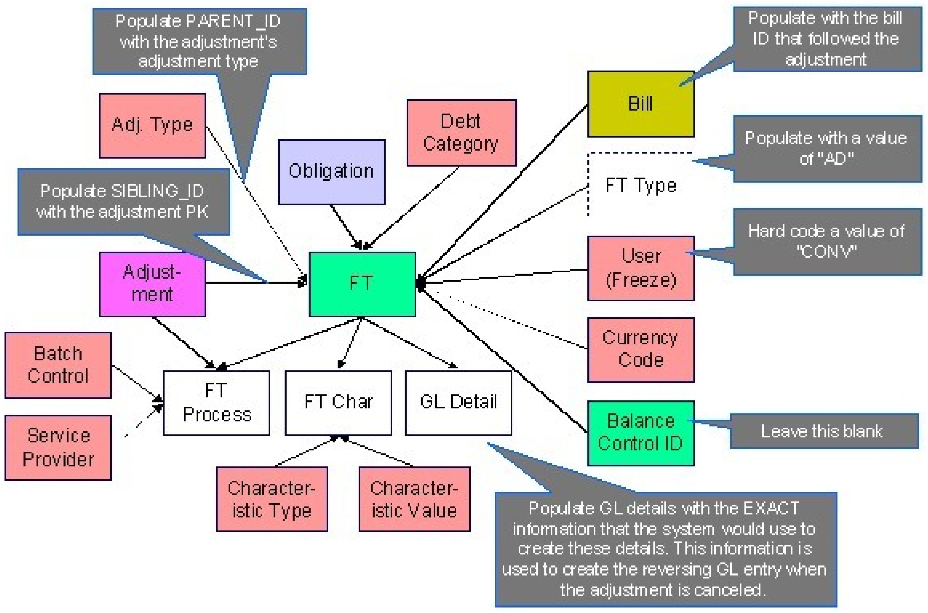
Adjustment - Main

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.


Populate PARENT_ID with the adjustment's adjustment type
Populate with the bill ID that followed the adjustment
Bill
Adj. Type
Debt Category
Obligation
Populate with a value of "AD"
FT Type
Populate SIBLING_ID with the adjustment PK
Hard code a value of "CONV"
User (Freeze)
Adjust-ment
FT
Currency Code
Batch Control
FT Process
FT Char
GL Detail
Balance Control ID
Leave this blank
Service Provider
Character-istic Type
Character-istic Value
Populate GL details with the EXACT information that the system would use to create these details. This information is used to create the reversing GL entry when the adjustment is canceled.

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 A/P Request	*CI_ADJ_APREQ*	Yes *CI_ADJ_APREQ_K*	CIPVAPRV	CIPVAPRK *Has dependencies*	CIPVAPRI
Adjustment Calc Lines	*CI_ADJ_CALC_LN*	No. The key is adjustment id and a sequence number	CIPVACLV		CIPVACLI
Adjustment Calc Line Characteristics	*CI_ADJ_CL_CHAR*	No. The key is adjustment id, a sequence number and characteristic type.	CIPVCLCV		CIPVCLCI
Adjustment Log	*CI_ADJ_LOG*	No. The key is adjustment id and sequence.	CIPVADLV		CIPVADLI
Adjustment Log Message Parameters	*CI_ADJ_LOG_PARM*	No. The key is adjustment id, sequence and parameter sequence.	CIPVADPV		CIPVADPI
FT (financial transaction)	*CI_FT*	Yes *CI_FT_K*	CIPVFTFV	CIPVFTXK *Has dependencies*	CIPVFTFI See Optional FK Note below.

FT Characteristic	*CI_FT_CHAR*	No. The key is FT id, char type code and a sequence number	CIPVFTCV	CIPVFTCI
FT GL (FT general ledger)	*CI_FT_GL*	No. The key is FT id and a GL sequence number	CIPVFTGV	CIPVFTGI
FT Process	*CI_FT_PROC*	No. The key is FT id and a sequence number	CIPVFTPV	CIPVFTPI
Adjustment Characteristic	*CI_ADJ_CHAR*	No. The key is adjustment id, char type code and a sequence number	CIPVADCV	CIPVADCI

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 the system would use to create them. This information is used to create the reversing GL entry when the adjustment is canceled.

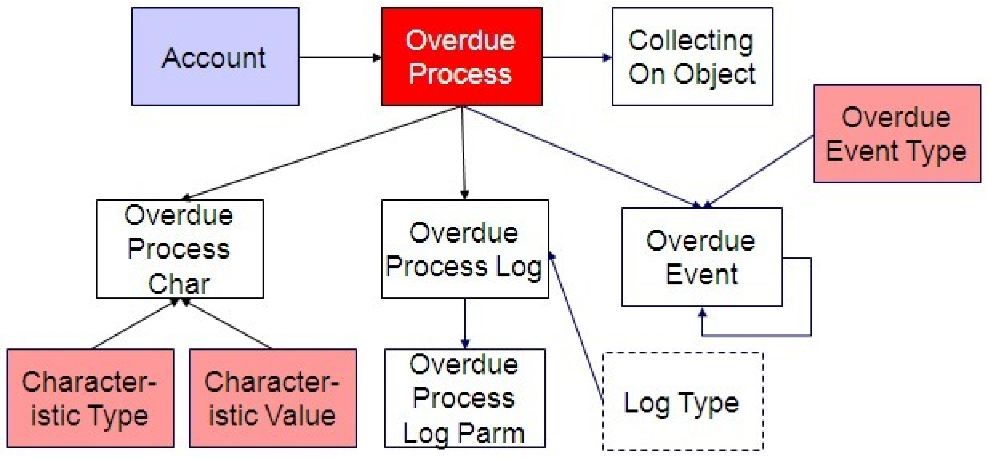
Optional FK Note

FTs for adjustments have an optional foreign key BILL_ID to the Bill (Classic) table (CI_BILL). Conversion for classic billing locations is no longer supported. However, the key generation program for Bill (CIPVBILK) must still be run. Refer to *The Big Picture of Key Assignment* for more information.

Overdue Process

Overdue Process Data Model

The following data model illustrates the Overdue Process object.



Overdue Process Table Names

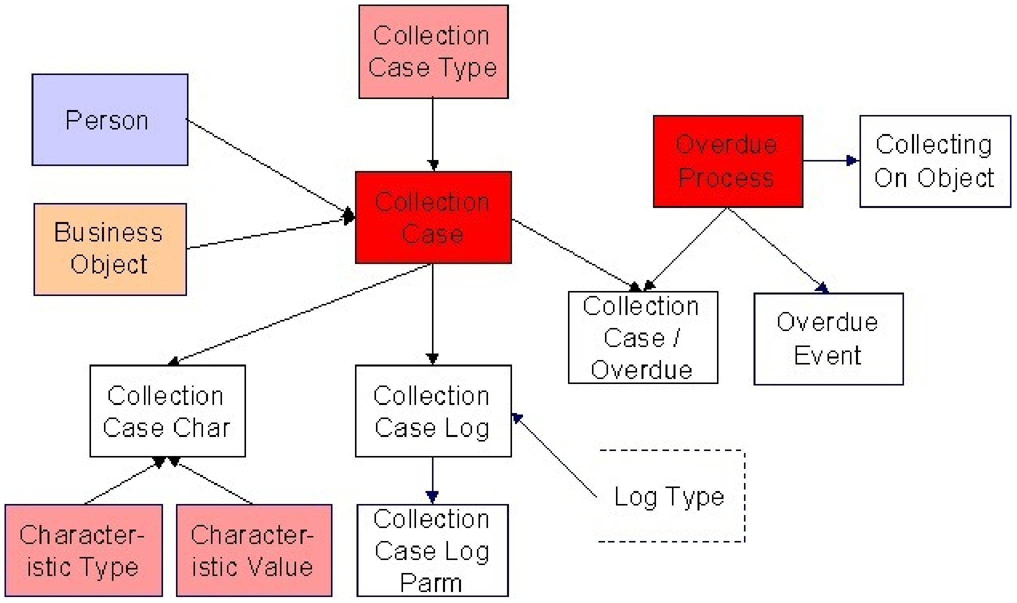
Data Model Name	***Table Name***	***Generated Keys***	***Object Validation Batch Control***	***Referential Integrity Validation Batch Control***	***Key Assignment Batch Control***	***Insertion Batch Control***
Overdue Process	*CI_OD_PROC*	Yes. *CI_OD_PROC_K*	VAL-ODPR	CIPVODPV	CIPVODPK *Has dependencies*	CIPVODPI
Overdue Process Collecting On Object	*CI_OD_PROC_OBJ*	No. The key is Overdue Process ID and Sequence.		CIPVODOV		CIPVODOI
Overdue Process Event	*CI_OD_EVT*	No. The key is Overdue Process ID and Sequence.		CIPVODEV		CIPVODEI
Overdue Process Event Dependencies	*CI_OD_EVT_DEP*	No. The key is Overdue Process ID, Event Sequence and Sequence.		CIPVODDV		CIPVODDI

Overdue Process Log	*CI_OD_PROC_ LOG*	No. The key is Overdue Process ID and Sequence.	CIPVODLV	CIPVODLI
Overdue Process Log Parameter	*CI_OD_PROC_ LOGPARM*	No. The key is Overdue Process ID, Sequence and Message Parameter Sequence.	CIPVOLPV	CIPVOLPI

Collection Case

Collection Case Data Model

The following data model illustrates the Collection Case object.



Collection Case 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 Case	*CI_COLL_CASE*	Yes.*CI_COLL_ CASE_K*	VAL-CLCS	CIPVCCSV	CIPVCCSK *Has dependencies*	CIPVCCSI
Collection Case Characteristic	*CI_COLL_ CASE_CHAR*	No. The key is Collection Case ID. Characteristic		CIPVCCCV		CIPVCCCI

		Type, and Sequence.		
Collection Case Log	*CI_COLL_ CASE_LOG*	No. The key is Collection Case ID and Sequence.	CIPVCCLV	CIPVCCLI
Collection Case Log Parameter	*CI_COLL_ CASE_LOG_ PARM*	No. The key is Collection Case ID, Sequence, and Message Parameter Sequence.	CIPVCCPV	CIPVCCPI
Collection Case Overdue Process	*CI_COLL_ CASE_OD*	No. The key is Collection Case ID and Overdue Process ID.	CCIPVCCOV	CCIPVCCOI

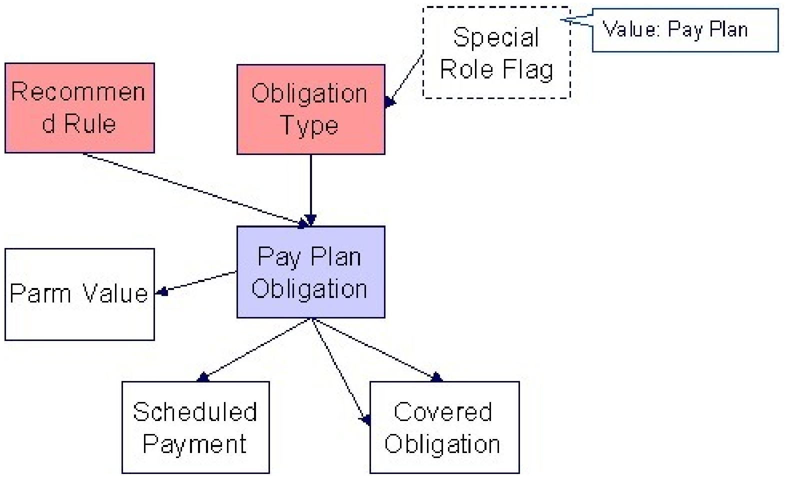
Collection Case Suggestions

This maintenance object includes a character large object field that your organization may be using to capture implementation specific data as defined by your business objects. For records of this type, the process to insert the records to the staging table is responsible for populating the data in this CLOB as per the record's business object schema.

Payment Plan

Payment Plan Data Model

The following data model illustrates the Payment Plan objects.



Payment Plan Table Names

Data Model Name	*Table Name*	*Generated Keys*	*Referential Integrity Validation Batch Control*	*Key Assignment Batch Control*	*Insertion Batch Control*

Covered Obligation	*CI_NB_SA*	No. The key is SA_ ID plus CVRD_SA_ ID	CIPVNBSV		CIPVNBSI
Scheduled Payments	*CI_NB_SCHED_ PAY*	Yes *CI_NB_ SCHED_PAY_K*	CIPVNSPV	CIPVNSPK *Has dependencies*	CIPVNSPI
Pay Plan Parameters	*CI_SA_NB_PARM*	No. The key is SA_ ID plus Sequence Number.	CIPVNPMV		CIPVNPMI

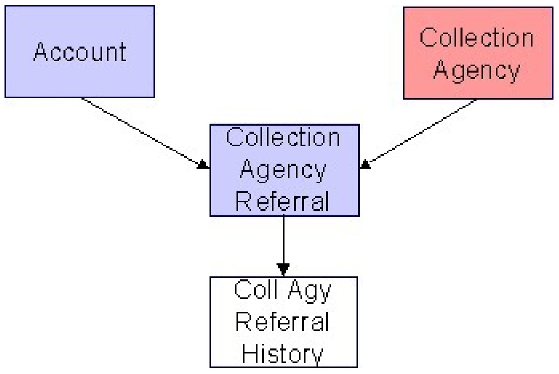
Payment Plan Suggestions

This maintenance object includes a character large object field that your organization may be using to capture implementation specific data as defined by your business objects. For records of this type, the process to insert the records to the staging table is responsible for populating the data in this CLOB as per the record's business object schema.

Collection Agency Referral

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 Agency Referral	*CI_COLL_AGY_REF*	Yes. *CI_COLL_ AGY_REF_K*	CIPVCARV	CIPVCARK	CIPVCARI
Collection Agency Referral History	*CI_COLL_AGY_HIS*	No. The key is collection agency referral id and characteristic type code	CIPVARHV		CIPVARHI

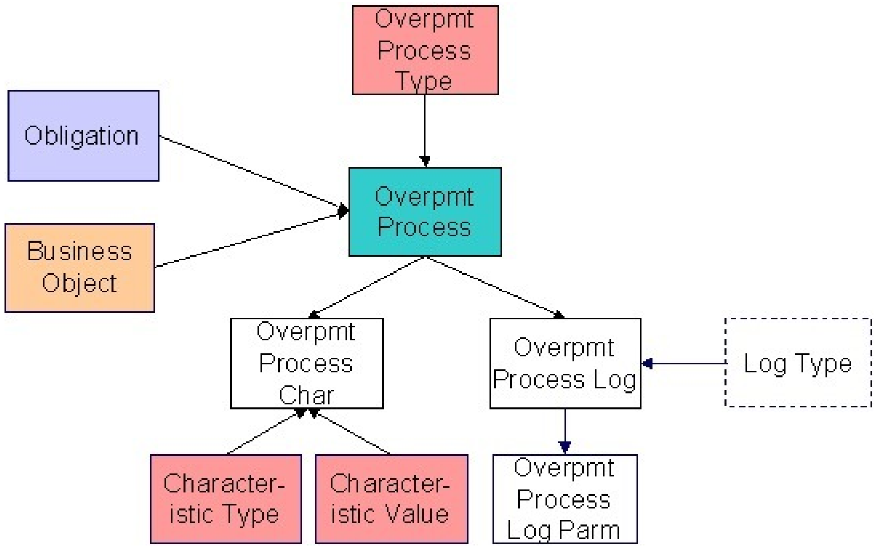
Collection Agency Referral Suggestions

N/A

Overpayment Process

Overpayment Process Data Model

The following data model illustrates the Overpayment Process object.



Overpayment 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*
Overpayment Process	*CI_OP_PROC*	Yes. *CI_OP_PROC_K*	VAL-OVPY	CIPVOPPV	CIPVOPPK	CIPVOPPI
Overpayment Process Character	*CI_OP_PROC_CHAR*	No. The key is Overpayment Process ID, Characteristic Type, and Sequence.		CIPVOPCV		CIPVOPCI
Overpayment Process Log	*CI_OP_PROC_LOG*	No. The key is Overpayment Process ID and Sequence.		CIPVOPLV		CIPVOPLI
Overpayment Process Log Parameter	*CI_OP_PROC_LOG_PARM*	No. The key is Overpayment Process ID, Sequence, and Message		CIPVOPMI		CIPVOPMI

		Parameter Sequence.				

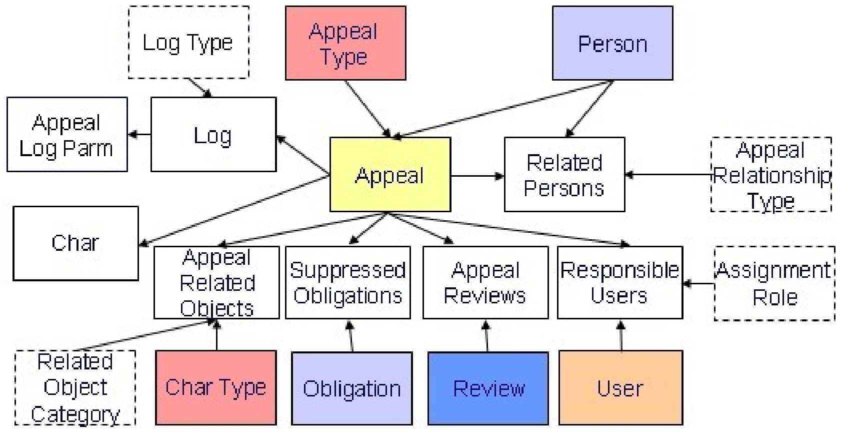
Overpayment Process Suggestions

This maintenance object includes a character large object field that your organization may be using to capture implementation specific data as defined by your business objects. For records of this type, the process to insert the records to the staging table is responsible for populating the data in this CLOB as per the record's business object schema.

Appeal

Appeal Data Model

The following data model illustrates the Appeal object.



Appeal 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***
Appeal	*C1_APPEAL*	Yes. *C1_ APPEAL_K*	VAL-APPL	CIPVAPPV	CIPVAPPK	CIPVAPPI
Appeal Related Person	*C1_APPEAL_ PER*	No. The key is Appeal ID, Relationship Type, and Person ID.		CIPVAPEV		CIPVAPEI
Appeal Related Object	*C1_APPEAL_ REL_OBJ*	No. The key is Appeal ID, Characteristic Type, and Person ID.		CIPVAPOV		CIPVAPOI
Appeal Suppressed Obligation	*C1_APPEAL_ SUPP_SA*	No. The key is Appeal ID and Obligation ID.		CIPVAPSV		CIPVAPSI

Appeal Review	*C1_APPEAL_REVIEW*	No. The key is Appeal ID and Review ID.	CIPVAREV	CIPVAREI
Appeal Responsible User	*C1_APPEAL_RESP_USER*	No. The key is Appeal ID, Assignment Role Flag, and User ID.	CIPVARUV	CIPVARUI
Appeal Characteristic	*C1_APPEAL_CHAR*	No. The key is Appeal ID, Characteristic Type, and Sequence.	CIPVAPHV	CIPVAPHI
Appeal Log	*C1_APPEAL_LOG*	No. The key is Appeal ID and Sequence.	CIPVAPLV	CIPVAPLI
Appeal Log Parameter	*C1_APPEAL_LOG_PARM*	No. The key is Appeal ID, Sequence, and Message Parameter Sequence.	CIPVALPV	CIPVALPI

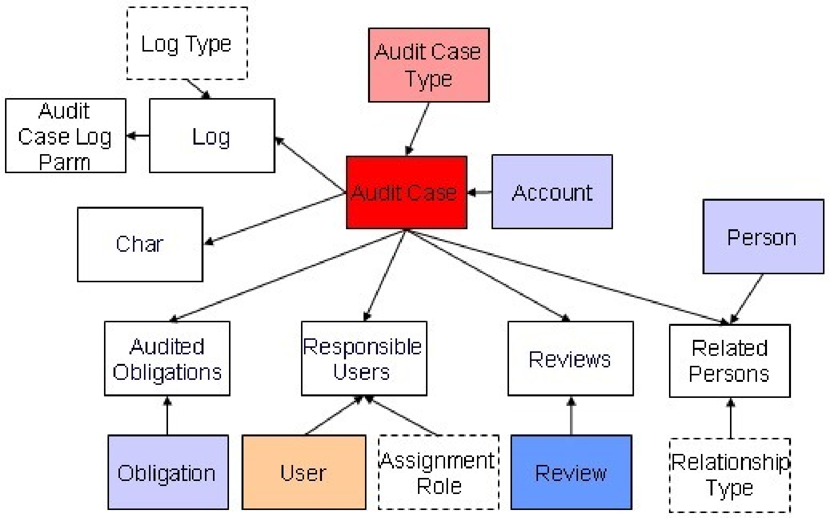
Appeal Suggestions

This maintenance object includes a character large object field that your organization may be using to capture implementation specific data as defined by your business objects. For records of this type, the process to insert the records to the staging table is responsible for populating the data in this CLOB as per the record's business object schema.

Audit Case

Audit Case Data Model

The following data model illustrates the Audit Case object.



Audit Case 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
Audit Case	*C1_AUDIT_CASE*	Yes. *C1_AUDIT_CASE_K*	VAL-AUDT	CIPVAUDV	CIPVAUDK	CIPVAUDI
Audit Case Related Person	*C1_AUDIT_CASE_PER*	No. The key is Audit Case ID, Relationship Type, and Person ID.		CIPVAUPV		CIPVAUPI
Audit Case Audited Obligation	*C1_AUDIT_CASE_SA*	No. The key is Audit Case ID and Obligation ID.		CIPVAUSV		CIPVAUSI
Audit Case Review	*C1_AUDIT_CASE_REVIEW*	No. The key is Audit Case ID and Review ID.		CIPVAURV		CIPVAURI
Audit Case Responsible User	*C1_AUDIT_CASE_RESP_USER*	No. The key is Audit Case ID, Assignment Role Flag, and User ID.		CIPVAUUV		CIPVAUUI
Audit Case Characteristic	*C1_AUDIT_CASE_CHAR*	No. The key is Audit Case ID, Characteristic Type, and Sequence.		CIPVAUCV		CIPVAUCI
Audit Case Log	*C1_AUDIT_CASE_LOG*	No. The key is Audit Case ID and Sequence.		CIPVAULV		CIPVAULI

Audit Case Log Parameter	*C1_AUDIT_CASE_LOG_PARM*	No. The key is Audit Case ID, Sequence, and Message Parameter Sequence.		CIPVAUMV		CIPVAUMI

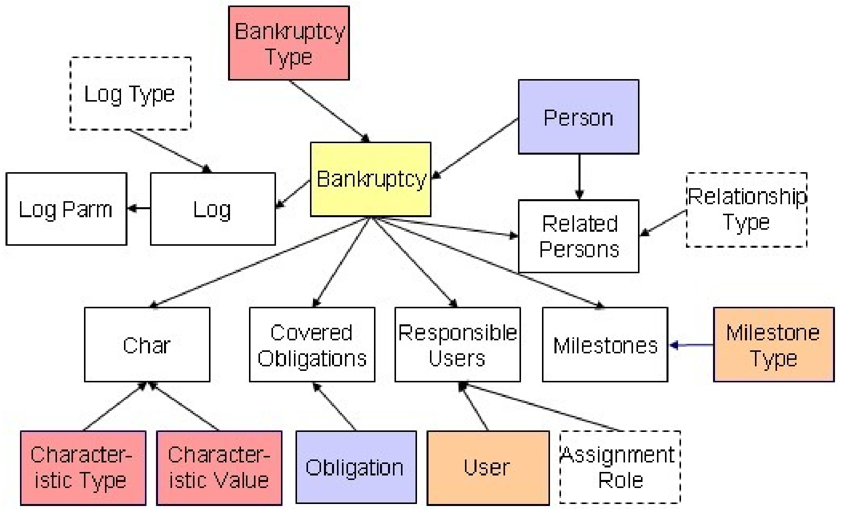
Audit Case Suggestions

This maintenance object includes a character large object field that your organization may be using to capture implementation specific data as defined by your business objects. For records of this type, the process to insert the records to the staging table is responsible for populating the data in this CLOB as per the record's business object schema.

Bankruptcy

Bankruptcy Data Model

The following data model illustrates the Bankruptcy object.



Bankruptcy 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***
Bankruptcy	*C1_BANKRUPTCY*	Yes. *C1_BANKRUPTCY_K*	VAL-BKCY	CIPVBKCV	CIPVBKCK	CIPVBKCI
Bankruptcy Related Person	*C1_BANKRUPTCY_PER*	No. The key is Bankruptcy ID, Relationship Type, and Person ID.		CIPVBKPV		CIPVBKPI

Bankruptcy Covered Obligation	*C1_ BANKRUPTCY_ SA*	No. The key is Bankruptcy ID and Obligation ID.	CIPVBKOV	CIPVBKOI
Bankruptcy Milestones	*C1_ BANKRUPTCY_ MILESTONE*	No. The key is Bankruptcy ID, Milestone Type and Sequence Number.	CIPVBKMV	CIPVBKMI
Bankruptcy Responsible User	*C1_ BANKRUPTCY_ RESP_USER*	No. The key is Bankruptcy ID, Assignment Role Flag, and User ID.	CIPVBKUV	CIPVBKUI
Bankruptcy Characteristic	*C1_ BANKRUPTCY_ CHAR*	No. The key is Bankruptcy ID, Characteristic Type, and Sequence.	CIPVBKHV	CIPVBKHI
Bankruptcy Log	*C1_ BANKRUPTCY_ LOG*	No. The key is Bankruptcy ID and Sequence.	CIPVBKLV	CIPVBKLI
Bankruptcy Log Parameter	*C1_ BANKRUPTCY_ LOG_PARM*	No. The key is Bankruptcy ID, Sequence, and Message Parameter Sequence.	CIPVBKAV	CIPVBKAI

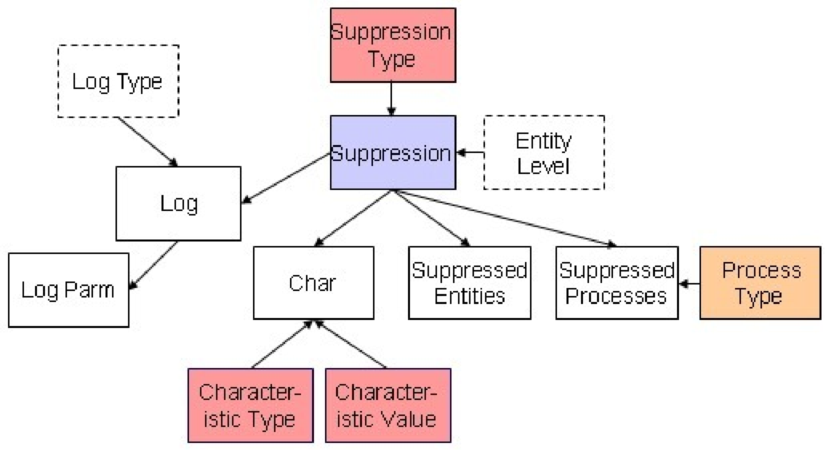
Bankruptcy Suggestions

This maintenance object includes a character large object field that your organization may be using to capture implementation specific data as defined by your business objects. For records of this type, the process to insert the records to the staging table is responsible for populating the data in this CLOB as per the record's business object schema.

Suppression

Suppression Data Model

The following data model illustrates the Suppression object.



Suppression 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***
Suppression	*C1_ SUPPRESSION*	Yes. *C1_ SUPPRESSION_ K*	VAL-SUPP	CIPVSUPV	CIPVSUPK	CIPVSUPI
Suppression Entity	*C1_ SUPPRESSION_ ENTITY*	No. The key is Suppression ID, Suppression Entity ID.		CIPVSUEV		CIPVSUEI
Suppression Process	*C1_ SUPPRESSION_ PROCESS*	No. The key is Suppression ID, Suppression Process flag.		CIPVSURV		CIPVSURI
Suppression Characteristic	*C1_ SUPPRESSION_ CHAR*	No. The key is Suppression ID, Characteristic Type, and Sequence.		CIPVSUCV		CIPVSUCI
Suppression Log	*C1_ SUPPRESSION_ LOG*	No. The key is Suppression ID and Sequence.		CIPVSULV		CIPVSULI
Suppression Log Parameter	*C1_ SUPPRESSION_ LOG_PARM*	No. The key is Suppression ID, Sequence, and Message Parameter Sequence.		CIPVSUMV		CIPVSUMI

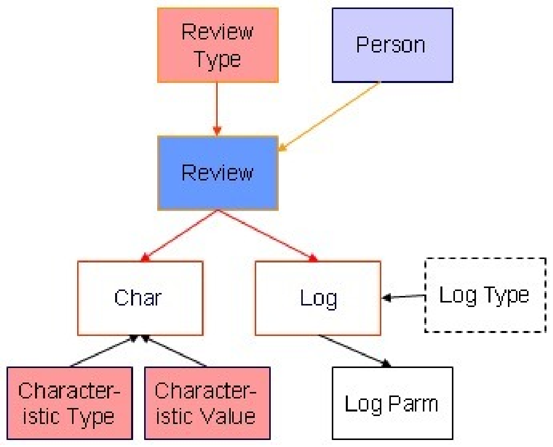
Suppression Suggestions

This maintenance object includes a character large object field that your organization may be using to capture implementation specific data as defined by your business objects. For records of this type, the process to insert the records to the staging table is responsible for populating the data in this CLOB as per the record's business object schema.

Review

Review Data Model

The following data model illustrates the Review object.



Review 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***
Review	*C1_REVIEW*	Yes. *C1_ REVIEW_K*	VAL-REVW	CIPVREVV	CIPVREVK	CIPVREVI
Review Characteristic	*C1_REVIEW_ CHAR*	No. The key is Review ID, Characteristic Type, and Sequence.		CIPVRVCV		CIPVRVCI
Review Log	*C1_REVIEW_ LOG*	No. The key is Review ID and Sequence.		CIPVRLVV		CIPVRVLI
Review Log Parameter	*C1_REVIEW_ LOG_PARM*	No. The key is Review ID, Sequence, and Message Parameter Sequence.		CIPVRLPV		CIPVRLPI

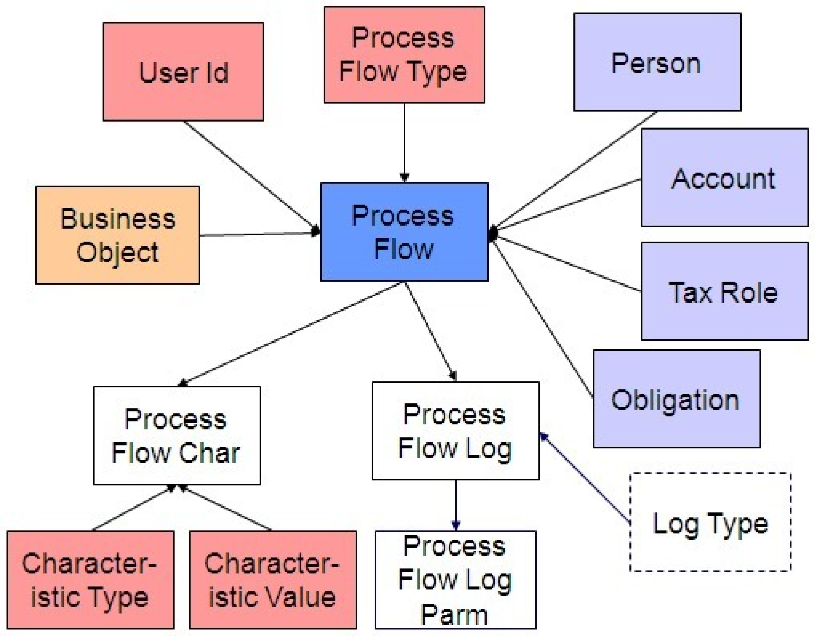
Review Suggestions

This maintenance object includes a character large object field that your organization may be using to capture implementation specific data as defined by your business objects. For records of this type, the process to insert the records to the staging table is responsible for populating the data in this CLOB as per the record's business object schema.

Process Flow

Process Flow Data Model

The following data model illustrates the process flow object.



Process Flow 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*
Process Flow	*CI_PROC_FLOW*	Yes.*CI_PROC_FLOW_K*	VAL-PRFL	CIPVPRFV	CIPVPRFK *Has dependencies*	CIPVPRFI
Process Flow Characteristic	*CI_PROC_FLOW_CHAR*	No. The key is Process Flow ID, Characteristic Type, and Effective Date.		CIPVPFCV		CIPVPFCI
Process Flow Log	*CI_PROC_FLOW_LOG*	No. The key is Process Flow ID and Sequence.		CIPVPFLV		CIPVPFLI

Process Flow Log Parameter	*CI_PROC_FLOW_LOG_PARM*	No. The key is Process Flow ID, Sequence and Message Parameter Sequence.	CIPVPLPV	CIPVPLPI

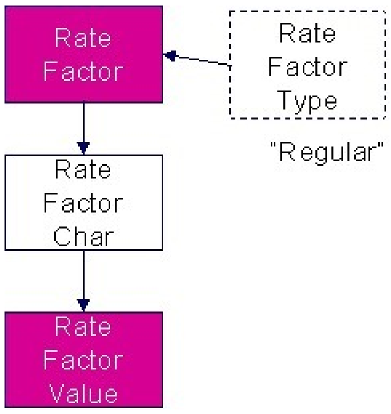
Process Flow Suggestions

This maintenance object includes a character large object field that your organization may be using to capture implementation specific data as defined by your business objects. For records of this type, the process to insert the records to the staging table is responsible for populating the data in this CLOB as per the record's business object schema.

Rate Factor Value

Rate Factor Value Data Model

The following data model illustrates the rate factor objects.



Rate Factor Value Table Names

Data Model Name	***Table Name***	***Generated Keys***	***Referential Integrity Validation Batch Control***	***Key Assignment Batch Control***	***Insertion Batch Control***
Rate Factor Value	*CI_BF_VAL*	No. The key is BF_CD plus CHAR_TYPE_CD plus CHAR_VAL plus EFFDT	CIPVBFVV		CIPVBFVI (Not threadable)

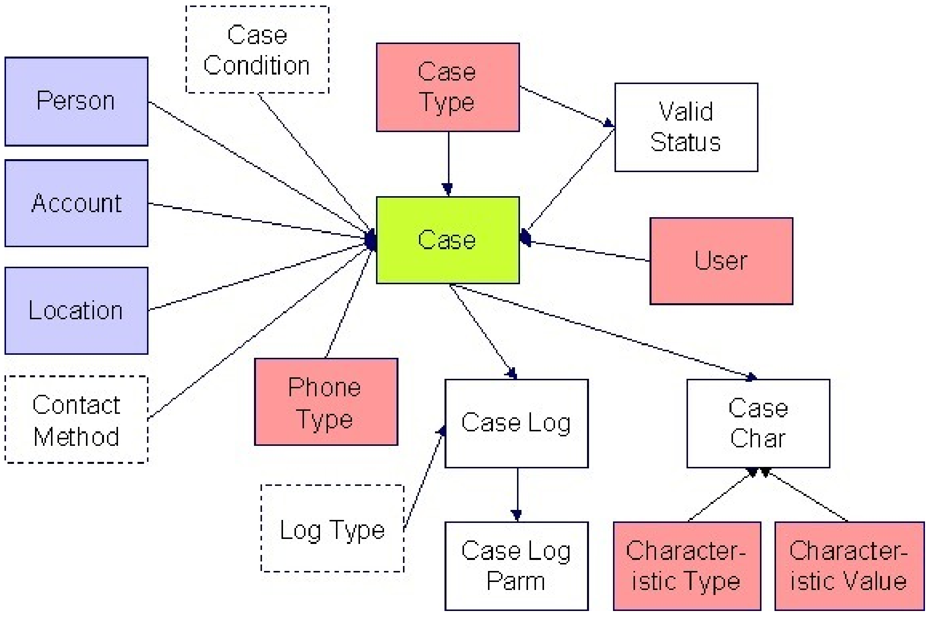
Rate Factor Value Suggestions

N/A.

Case

Case Data Model

The following data model illustrates the Case object.



Case 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
Case	CI_CASE	Yes. CI_CASE_K	VAL-CASE		CIPVCSEK	CIPVCSEI
Case Characteristic	CI_CASE_CHAR	No. The key is case id, char type code and a sequence number		CIPVCCHV		CIPVCCHI
Case Log	CI_CASE_LOG	No. The key is case id and a log sequence number		CIPVCLGV		CIPVCLGI
Case Log Parameter	CI_CASE_LOG_PARM	No. The key is case id, log sequence number and a parameter sequence number		CIPVCPAV		CIPVCPAI

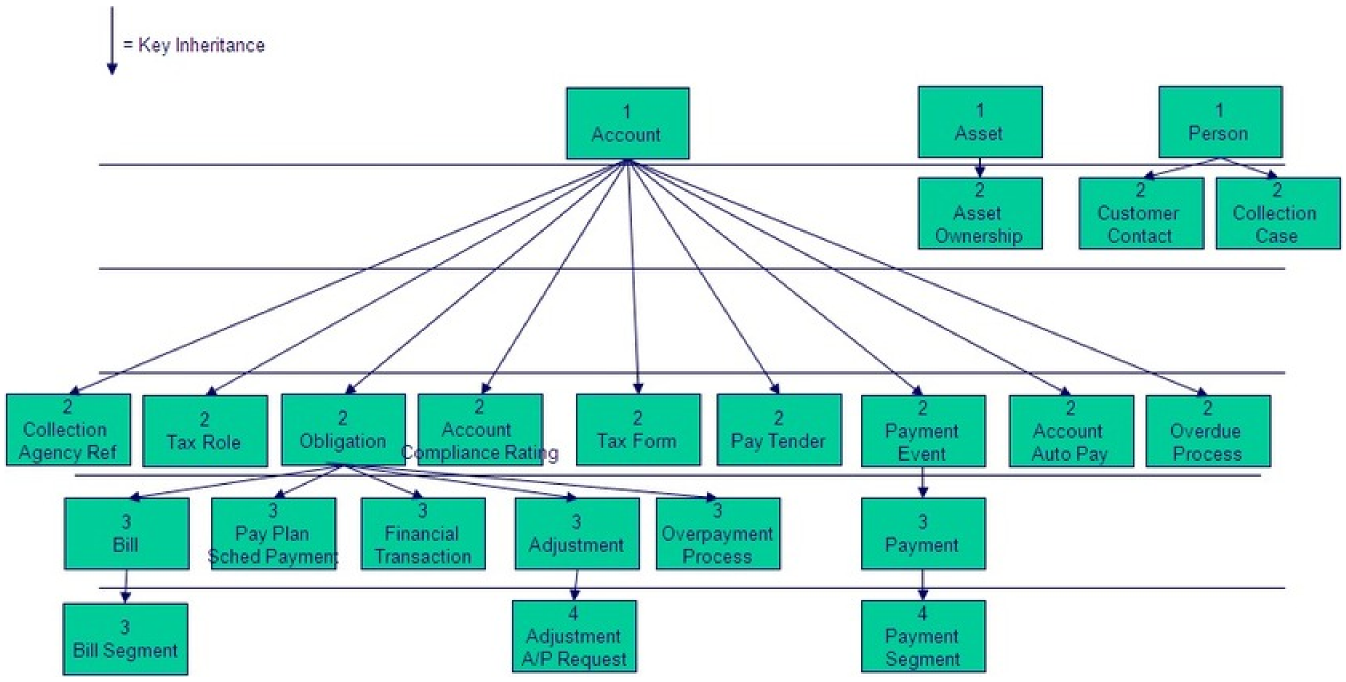
Case Suggestions

This maintenance object includes a character large object field that your organization may be using to capture implementation specific data as defined by your business objects. For records of this type, the process to insert the records to the staging table is responsible for populating the data in this CLOB as per the record's business object schema.

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.

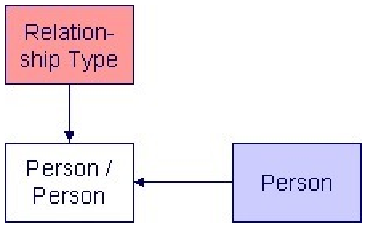


Please refer to the staging tables sections above for the respective names of the programs to allocate each table's keys.

CAUTION: 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 this course:



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 a logical grouping of entities. For example, form related entities are one color and taxpayer information related entities are a different color.

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 / Person entity is related to the Person object and does not have its own maintenance object.

The following table describes the colors utilized in the documentation:

Color	***Grouping***
🟦	Taxpayer 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.
⬜	Child table - the entity is maintained in respect of a higher level entity.
⬚	Lookup - 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 Lookup user interface.
🟧	Metadata.
🟪	Adjustment
🟩	Financial Transaction
🟪	Rates
🟨	Billing
🟥	Collection

🟦	Overpayment
🟦	Forms
🟩	Case

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, the line between account type and Account illustrates that an account type may have many Accounts, but an Account may be part of a single account type.

Appendix B - Multiple Owners In A Single Database

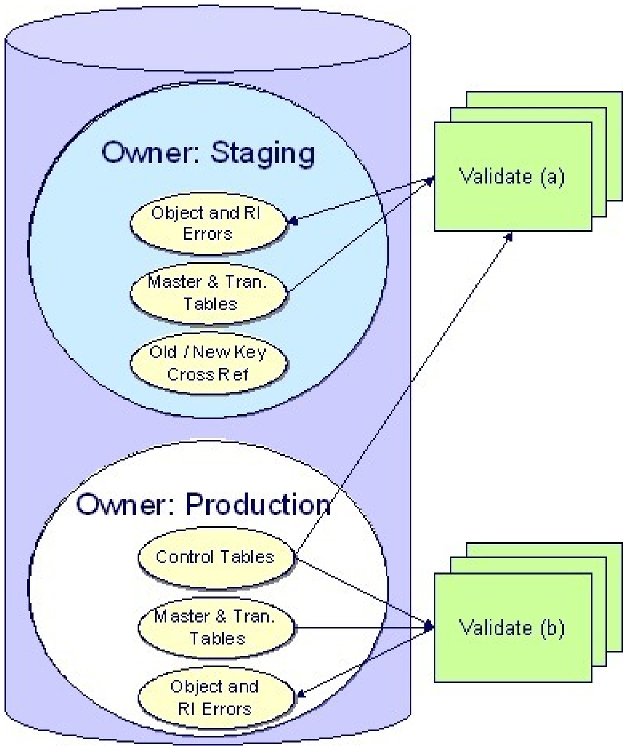
In the schematic referenced in the *Introduction*, you'll notice that there are two table owners in the system database. We refer to the first owner as "staging" and the second owner as "production".

The staging owner is linked to the tables into which you insert your pre-validated data. These tables have an owner ID of **CISSTG**.

NOTE: 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.

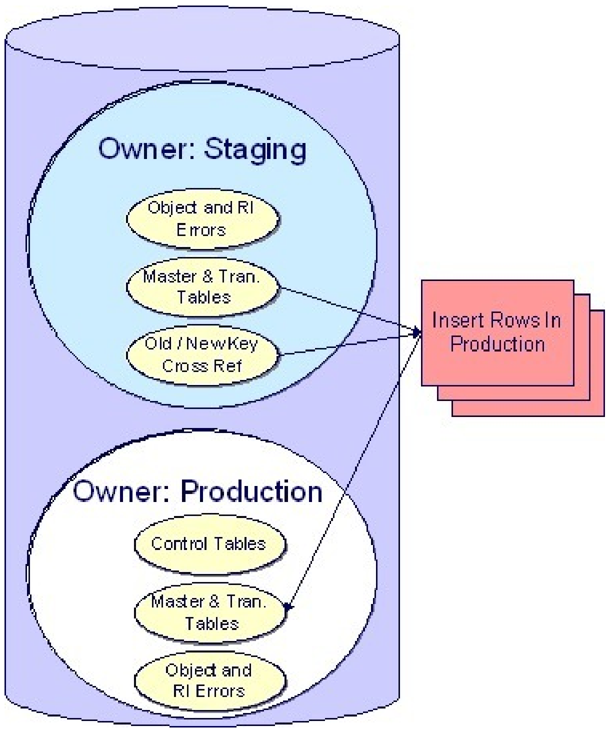
This is accomplished as follows:

- 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.
- Tender Source. References a suspense obligation.

Appendix D - Index Creation Script

The following script extracts the index DDL for CK tables and saves them in a script called ***create_ck_index.sql***.

```
set pagesize 0
set linesize 3000
set long 500000
set longchunksize 500000
set trimspool on
set feed off
spool alterindex.sql
SELECT 'ALTER INDEX '||
u.index_name||' PARALLEL 32 NOLOGGING ;' FROM USER_INDEXES u where u.table_name like 'CK%';
spool off
@alterindex.sql
spool create_ck_index.sql
SELECT REPLACE ( DBMS_METADATA.GET_DDL('INDEX',u.index_name),'"','')||';' FROM USER_INDEXES u where u.table_n
%';
SELECT 'ALTER INDEX '||
u.index_name||' NOPARALLEL LOGGING ;' FROM USER_INDEXES u where u.table_name like 'CK%';
spool off;
exit;
```

The ***create_ck_index.sql*** script can be used to recreate CX_ID and CI_ID indexes on the CK_<table_name> after all *key assignment tiers* have executed with MODE = "I":