Sigma Systems, Inc.

Overview Class Diagrams for the Kuali Student Accounts Receivables Management (KSA-RM) System

Sigma Systems
March 2012

Classes: KSA-RM

Change Log

Author	Date	Changes
Paul	3/2/2012	Added a changelog.
Paul	3/8/2012	Cleaned up account classes.
Paul	3/9/2012	Incorporated local types into structure.
Paul	3/14/2012	Added payment billing attribute to Debit Class. Minor changes to add creatorId to account support classes. Numerous minor alterations due to data model rationalization.
Paul	3/16/2012	Added isDefault to account helper classes. Account.kimEntityId became Account.entityId
Paul	3/29/2012	Changed Document to add creatorId. Removed FollowUpMemo class. Memo class now contains priorMemo, and followUpMemo (no longer followUpMemoId).
Paul	4/5/2012	Added Allocation class.
Paul	5/29/2012	Documented AuditibleEntity and altered Activity to be able to trace such entities.
Paul	5/30/2012	Cleaned up Activity.
Paul	5/30/2012	Added UI classes.
Paul	5/31/2012	UI override and version.
Paul	6/1/2012	Added dob to ChargeableAccount for bio lookup. Added general ledger classes.
Paul	6/2/2012	Documented AccountProtectedInformation (oversight) Created cash tracking classes.
Paul	6/7/2012	Cleaned up document formatting, adding overview sections. Added AccountPreference class to support KSA-RM-RM (and others going forwards).
Paul	6/8/2012	Added BatchReceipt
Paul	6/11/2012	Added Language class and tweaked InstalledLanguage to remove items that are covered by Language.
Paul	6/25/2012	Added AccessControl.
Paul	7/2/2012	Added Ach class. Minor changes to Refund class.
Paul	7/3/2012	Version removed from UserInterfaceString classes. TransactionTypePermission became TransacationMaskRole. AccountPreference became UserPreference to be in line with uPortal.
Paul	7/13/2012	Addition to rollup for XML name. XmlDocument class added.
Paul	7/18/2012	Added code to auditable entities for XML matching.
Paul	7/20/2012	Added Company and Collection classes (very basic starting point for collections)
Paul	8/1/2012	Added GL Type to Transaction model for changes relating to payment billing.
Paul	8/2/2012	Changes to TransactionType model to handle more flexible general ledger types.
Paul	8/3/2012	Moved glOverride etc. to the transaction class, to allow both charges and payments to be overridden. Added session and in session status to general ledger tables.
Paul	8/13/2012	Removed session to simplify PA logic.
Paul	8/14/2012	Added recognitionDate to the transaction model.

Paul	8/15/2012	Worked on the TransactionList class to support rule-based payment application. Added AccountBlockType class.

Contents

KSA-RM Class Diagrams

Repeated Patterns

There are a number of repeating patterns which are assumed to be generally understood. Many classes have a creatorId, editorId and lastUpdate set of attributes. creatorId references the identifier for the creating entity, and editorId references the last entity that altered the object. This identifier is the authenticated name of the user who created or changed the type, and mimics the accountId field of an account. In the default configuration, this is derived from the principal id from KIM, which is most often a login name, or net id (for example, in an institution using first initial/ last name, it would be jdoe, fsmith, etc.) creationDate is the date when the entity came into existence, lastUpdate is the date/time stamp when the entity was last changed. There is a name field which is the standard name of the type, and also have a “description” field, which is primarily used in the UI to assist the user in making a decision. There is also a code field, which is unique, and is used for matching the types in a simplified manner, for example in the XML import process, referenced types are referenced by” code” .For example, the class BankType defines different bank account details that might be stored. A type might be “ACH”, and if the user enquires as to what that is, the system might explain what an ACH transaction is, and what information needs to be captured in order to use an ACH transaction. Many of these “Types” are extensions of the “AuditableEntity” class.

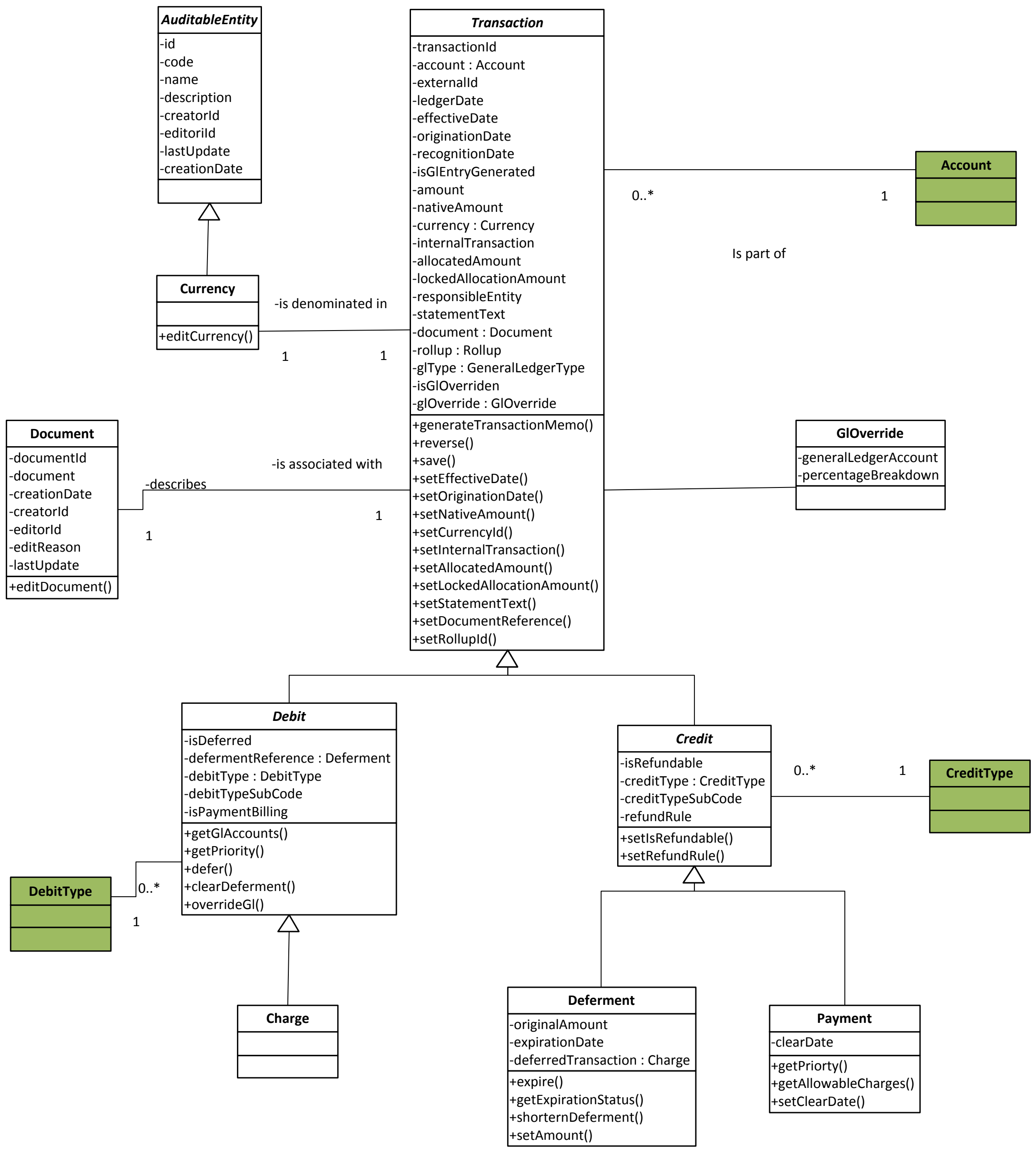
Any change to an object that has these fields will trigger an event in the activity log (Class Activity)

There are a number of “Type” classes, which are considered to be self-explanatory.

Class diagrams in green are defined more fully elsewhere in the document. Class diagrams in red are as yet undefined.

Attribute definitions are included where they are KSA classes.

Class Transaction Overview



Generic

The Transaction class is one of the most important classes in the KSA system. Transactions are stored in a table called KSSA_TRANSACTION and contain the fundamental information that is used to support a number of complex operations. Once a transaction is saved to the KSA ledger, few parts of it can be altered. (Exceptions would be allocation amounts, rollup identifier, etc.) The meaning of the attributes is defined in the data model documentation. If attributes are defined in this document, either implicitly or explicitly, and they conflict with the data model definitions, the data model definition is the document of record.

Class Transaction

Transaction is a purely abstract class. Its concrete classes are Charge, Deferment and Payment. Once a transaction has been set up with the minimum information (accountId, externalId, amount, responsibleEntity, debitType || creditType) These attributes cannot be altered once set, except for amount, in the case of a deferment.

Attributes and Initial States

If the following fields are not provided, they will be set to default values as listed. Some fields are automatically set on certain events.

transactionId	Populated by the system when the save() method is called. When the transaction is written to the database, a unique identifier is assigned to the transaction.
account	References the account object to which the transaction belongs. Passed by the constructor.
ledgerDate	Populated by the system when the save() method is called.
effectiveDate (*)	Date of instantiation.
originationDate (*)	Date that the transaction is effective with respect to ageing.
recognitionDate (*)	Period for which the transaction is recognized on the general ledger. By default, this is set to the effectiveDate, unless overridden.
amount (*)	Amount of the transaction set during instantiation. In the case of a deferment, this can also be set when the amount of the deferment is reduced or expired.
nativeAmount (*)	As amount.
isGlEntryGenerated	False. Set when GL transactions are created.
currency (*)	System currency object.
internalTransaction (**)	False
allocatedAmount (**)	0.00
lockedAllocationAmount (**)	0.00
statementText (**)	Derive from defaultStatementText, through creditType or debitType.
document (**)	null
rollup (**)	Derive from defaultRollupId through creditType or debitType.
glType	Zero by default. Refers to the general ledger type from which GL entries will be produced.
isGlOverriden	Are the standard general ledger codes to be used, or are they overridden?
glOverride []	Range of different general ledger overrides for the transaction.

Credits and Children

isRefundable (**)	False
refundRule (**)	Null
creditType (*)	Set by constructor.
creditTypeSubCode (*)	Inferred from constructor and date.

Deferments Only

originalAmount	The original value of the deferment.
expirationDate (**)	This field is a required piece of information in order to establish a deferment. It should be noted that deferments are only created through the defer() method of a Charge (or any future derived class of Debit)
deferredTransaction (**)	This is a required field for the establishment of a deferral. It is set to the id of the Charge whose defer() method was called.

Payments Only

isRefundable (**)	Derive default from credit type, unless overridden.
refundRule (**)	Derive from credit type, unless overridden.
clearDate (**)	The clearDate is calculated from the ledgerDate plus the default clearing period as referenced through the creditType model.

Debits and Children

isDeferred (**)	False. Set to true when the defer() method is called.
defermentId (**)	Null. Set by the defer() method.
debitType (*)	Set by the constructor. The date of the transaction allows the construction of a pointer to the correct type of debit, including the subcode.
debitTypeSubCode (*)	Set by inference from date and debitType. Stored here for information only.
isPaymentBilling	Boolean that tells the system that this transaction is part of a payment plan. Set during instantiation.

Items marked with a (*) can be set either during instantiation (through overloaded constructors) or through appropriate setters, until the save() method is called. After that point, these values are set and cannot be changed. Any attempt to alter these attributes after the save() method has been called should produce an exception.

Items marked with a (**) can be altered after a transaction has been saved to the ledger. Note that not all can be altered with setter methods. For example, while aCharge can have its isDeferred attribute altered, this would only happen through a call to its defer() method.

Methods

setAllocatedAmount (newAllocationAmount)
Sets the allocatedAmount to the value of newAllocationAmount. Cannot exceed (amount-lockedAllocationAmount). Generally only called by KSA-PA
setLockedAllocationAmount (newLockedAllocationAmount)
Sets the lockedAllocationAmount to the value of newLockedAllocationAmount. Cannot exceed (amount-allocatedAmount).

save()

Sets ledgerDate to the current date, gets the next transaction number and populates transactionId, then writes the transaction to the TRANSACTION table. Once a transactionId has been generated, all attempts to alter (*) attributes will throw an exception.

reverse(memo)

The reverse method allows a CSR to reverse a charge on an account. To do this, it does the following:
Instantiate a new transaction with the same creditType || debitType, and the same (but negated) amount.
Clear any pre-existing payment allocations through the payment allocation service.
Set both of the transactions to have the same lockedPaymentAllocation
Optionally, depending on configuration, for a Charge, mark the transactions as internalTransaction
Create a memo on the account with the memo text passed to the method using generateTransactionMemo()
Call payment allocation to rebalance the account.

generateTransactionMemo(memo)

Creates a Memo linked to a transaction. If a memo is already linked to the transaction, automatically generated a follow-up memo, at the end of the memo chain.

Charge.getGlAccounts()

Returns a list of general ledger accounts and percentage allocations for this transaction. This is done by querying DebitType.

Charge.getPriority

Get the priority of the debit, referenced in DebitType.

Charge.defer(expirationDate, memo)

Defer a transaction by creating a pre-filled Deferment transaction with the details of the charge. In addition to creating the deferment, the two transactions will be allocated together using lockedAllocationAmount. A transaction memo is also generated by this process.

Charge.defer(expirationDate, memo, defermentAmount)

Defer a transaction by creating a pre-filled Deferment transaction with the details of the charge. The deferment is only for the amount of the defermentAmount, rather than the total value of the charge, as with defer()

Charge.clearDeferment()

Called by Deferment.expire()

Charge.overrideGl (gl

Deferment.expire()

Expire the deferment on a charge. Also expired the memo associated with the deferment. Clears any lockedAllocationAmounts, and sets the amount of the transaction to zero.

Deferment.getExpirationStatus()

Returns true if expired, false if unexpired.

Deferment.shorternDeferment (newDefermentExpirationDate, memo)

Checks the new date is sooner than the old date, then sets the expiration date to a new date. Creates a memo explaining the change.

Payment.getPriority()

Gets the priority of the transaction by looking up the value in CreditType.

Payment.getAllowableCharges ()

Returns a list of allowable charges (DebitType) that the payment is allowed to pay off.

Payment.setClearDate (newClearDate)
Payments usually set their clearDate based on the ledgerDate, modified by a parameter in CreditType. This allows the default value to be overridden.

Class GlOverride

Generic

Simple associative class to permit the storage of general leger accounts and its breakdown percentages to the associated general ledger accounts, when overridden in the transaction.

Attributes and Initial States

generalLedgerAccount	Set by constructor.
percentageBreakdown	Set by constructor.

Methods

Class Currency

Generic

The currency class stores information relating to currencies. Before a transaction can be denominated in a currency (including the system-wide currency) the currency must exist in this class. Instantiating a currency automatically causes a persistent copy of the currency. Currencies may be edited, but not destroyed.

Attributes and Initial States

All currency attributes are established during instantiation. There are no default values.

Though currencyCode is expected to be the ISO standard currency code, the system does not make any attempt to enforce this.

Methods

Currency.editCurrency (existingCurrencyId, newCurrencyName)
If existingCurrencyId exists, replace CurrencyName with newCurrencyName. Otherwise, throw an exception.

Class Document

Generic

The document class provides a way of storing information about a transaction that can be displayed to the user.

Attributes and Initial States

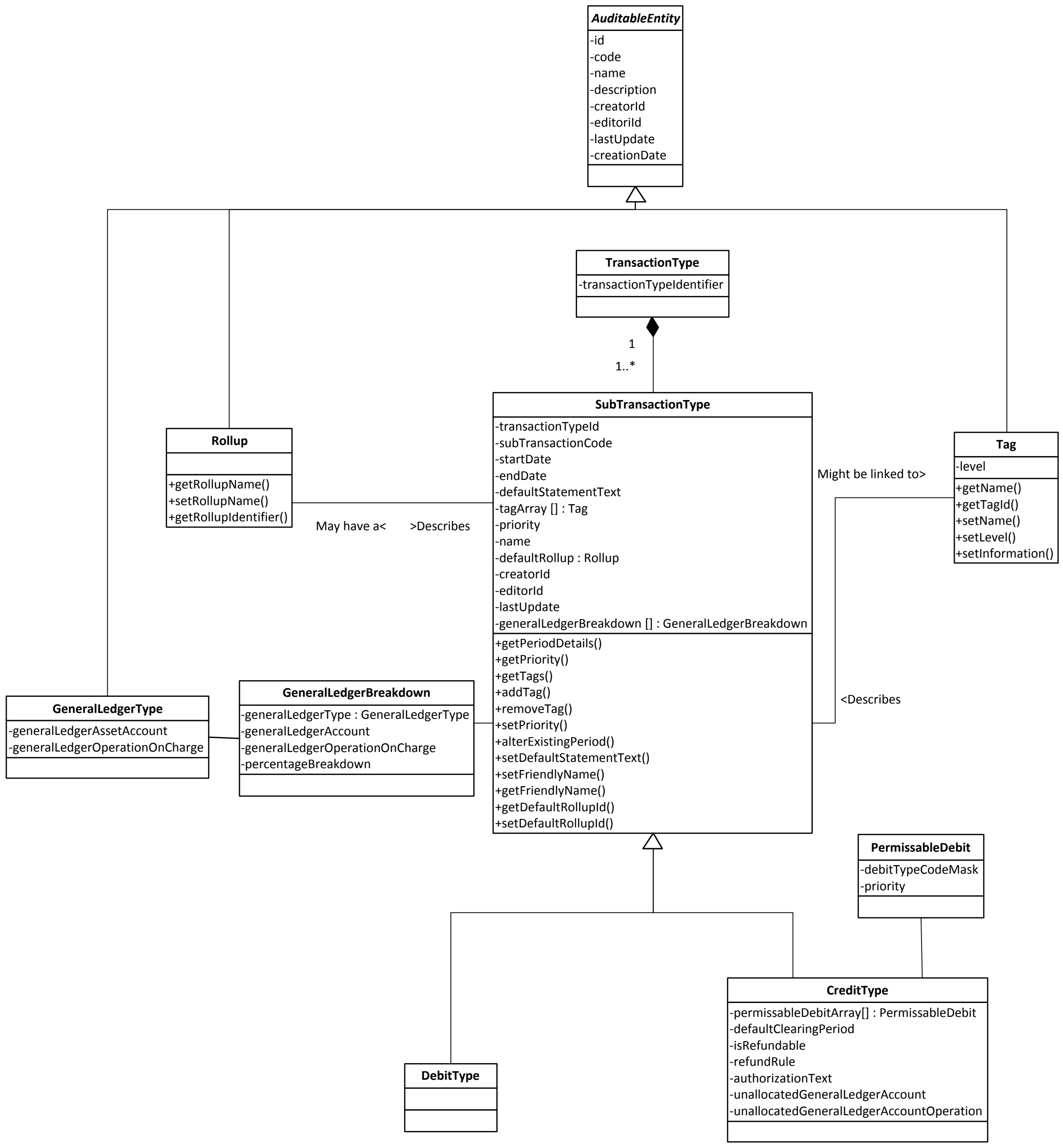
If the following fields are not provided, they will be set to default values as listed. Some fields are automatically set on certain events.

documentId	Field is populated on instantiation.
document	The document is passed during instantiation.
creationDate	Date on which the document was created, populated on instantiation.
creatorId	Populated on instantiation
editorId	Null
editReason	Null
lastUpdate	Set to creationDate

Methods

editDocument (newDocument, reason)
Replace the document with newDocument, editReason with reason, lastUpdate with current date, and editorId with current user.

Class Transaction Type Overview



Generic

Transaction types are fundamental to the way payments and charges are tracked, recorded, and allocated within KSA. Any transaction on the system has a Transaction Type, which defines a number of characteristics about the transaction. As an example, a transaction may have a transaction code of “SF100” which would show on a statement (by default) as “Mandatory Student Fee”, and might reconcile to a specific general ledger account or accounts.

Transactions types can change over time. That is to say that the way a certain transaction code acts is not always constant. Because of this, transaction types are subdivided into sub-transaction types, which are defined by date ranges. To continue the example, SF100 might reconcile to one general ledger account until 12/31/2012, but on 1/1/2013, it might reconcile to a different general ledger account(s).

Sub-transaction types then divide into two major groups, which are credit and debit types.

Class Transaction Type

Attributes and Initial States

transactionTypeIdentifier	Field is populated on instantiation. The identifier is freeform and is defined by the institution. An institution may use a predefined set of codes, for example a 3+3 identifier (like TUT001, is tuition, subcode 001). The identifier format is mostly irrelevant to the system.

Sub-transaction Types and children

subTransactionCode	Field is populated on instantiation. This is an automatic number field.
startDate	Unless otherwise provided, this is set to the current date.
endDate	Null. Note that a null in the dateEnd field implies that this is the current period. In the event of a new period being added that is defined as the current, the pervious current sub-transaction code must be brought to an end, by setting its dateEnd to the day preceding the dateStart of the new current sub-transaction code.
defaultStatementText	Must be set in the constructor.
tagArray	Null. This array is populated through the addTag() method.
priority	Set in the constructor.
name	Set in the constructor.
defaultRollup	Null
creatorId	Set to the entity id of the user who instantiated the sub-transaction type.
editorId	Null
lastUpdate	Assigned the system date.
generalLedgerBreakdown []	Assigned during creation.

Debit Type Only

Nothing extra.

Credit Type Only

permissableDebitArray	Null. Set via method.
defaultClearingPeriod	0 unless overridden.
isRefundable	True unless overridden.
refundRule	Null unless overridden.
authorizationText	UI component that indicates the type of data that is required for the authorization for an external payment

	transaction.
unallocatedGeneralLedgerAccount	If the credit cannot be allocated to a charge, then this is the general ledger account that will record the unallocated amount. This may be referred to as “unearned income” or “suspense”.
unallocatedGeneralLedgerAccountOperation	(C)redit or (D)ebit.

Class Rollup

Generic

Rollup is a very simple class that manages rollup identifiers. Rollups are a way of grouping types of transactions into a single, expandable group.

Attributes and Initial States

Id	Populated at instantiation
name	Populated at instantiation.
creatorId	Derived from creating entity.
editorId	Null
lastUpdate	System date at instantiation
description	UI assistive text.
code	Short name that is used in XML documents to identify this rollup. This must be unique. Uniqueness is checked on instantiation.

Class GeneralLedgerType

Generic

Simple type class that allows the storage of more than one type of general ledger breakdown. For example, when a charge is transferred to a third-party account, a different set of accounting strings may be used.

Attributes and Initial States

generalLedgerAssetAccount	Set at instantiation.
generalLedgerOperationOnCharge	Set to C(redit) or D(ebit). Answers the question, when a standard charge is resolved to this account, do we issue a credit or debit to the general ledger. Note that for payments, this field is ignored.

Class GeneralLedgerBreakdown

Generic

Simple associative class to permit the storage of general leger accounts and its breakdown percentages to the associated general ledger accounts.

Attributes and Initial States

generalLedgerType	Named type of the general ledger breakdown, allowing assignment of the transactions to different general ledger accounts (for

	example, third party, payment billing, etc.)
generalLedgerAccount	Set by constructor.
generalLedgerOperationOnCharge	Set to C(redit) or D(ebit). Answers the question, when a standard charge is resolved to this account, do we issue a credit or debit to the general ledger.
percentageBreakdown	Set by constructor.

Class PermissableDebit

Generic

Simple associative class to store the permissible debit array for a CreditType.

Attributes and Initial States

debitTypeCodeMask	Set by constructor.
priority	Set by constructor.

Class Tag

Generic

Tags are very similar to rollups, except a transaction may have more than one tag, whereas it may only have one rollup.

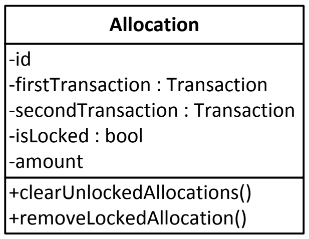
Attributes and Initial States

tagIdentifier	Populated at instantiation
tagName	Populated at instantiation.
creatorIdentifier	Derived from creating entity.
editorIdentifier	Null
lastUpdate	System date at instantiation
level	Set during instantiation
information	Set during instantiation
code	Simplified name of the tag that can be passed to the system from other systems (example, via an XML upload).

Class Allocation

Generic

The allocation class keeps track of all allocations made for an account. Although each transaction carries the allocation amounts (both locked and "unlocked") this is insufficient for reversing allocations, and it is a summary of all the allocations made on a transaction. The allocation class is used to record which transaction was allocated to which transaction, for what amount, and whether or not the allocation was a locked allocation or a regular allocation (payment application cannot reallocate locked allocations, whereas regular allocations can be reallocated automatically by the system, and cleared before payment application.)


Allocation
-id
-firstTransaction : Transaction
-secondTransaction : Transaction
-isLocked : bool
-amount
+clearUnlockedAllocations()
+removeLockedAllocation()

Attributes and Initial States

Id	Identifier for the allocation. Automatically generated at instantiation.
firstTransaction	The first of the two transactions that are allocated together. Passed by constructor.
secondTransaction	The second of the transactions. Passed by construction.
isLocked	States the type of allocation. Passed by constructor.
amount	Value, in system currency, of the allocation. Passed by constructor.

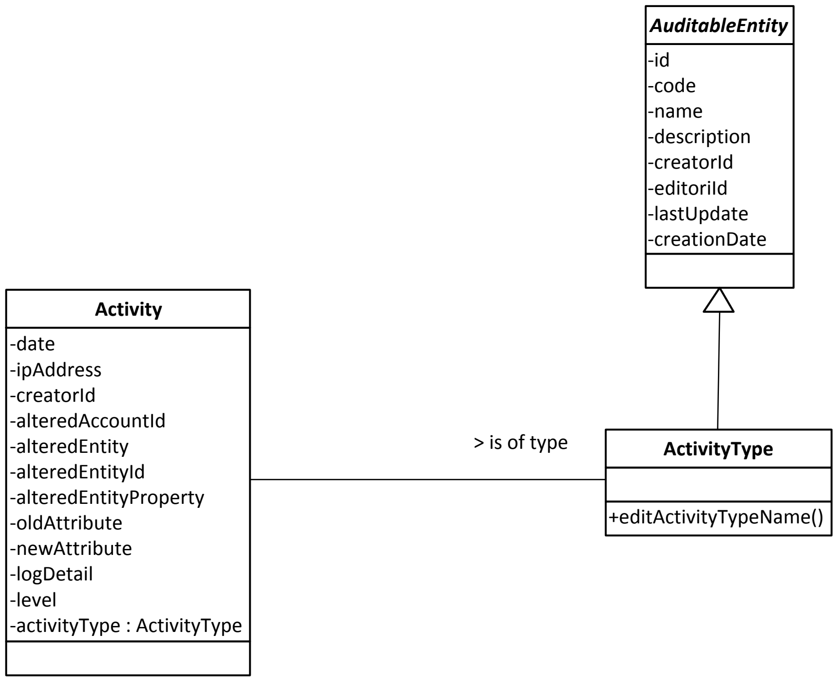
Methods

clearUnlockedAllocations()
Clears all the unlocked allocations for an account from the system. Often, before payment application, this is the starting point.
removeLockedAllocation(id)
When a locked allocation is being removed, the specific allocation is removed with this method.

Class Activity Overview

Generic

The activity class is a structure for logging events within the system. All items are set during instantiation and the object is persisted. All attributes have standard getters, but post-instantiation, there is no setters for the values.



Activity

Attributes and Initial States

date	Set to system date at instantiation
ipAddress	Set in constructor.
creatorId	Set in constructor.
alteredAccountId	Set in constructor.
alteredEntity	Set in constructor.
alteredEntityId	Set in constructor.
alteredEntityProperty	Set in constructor.
logDetail	Set in constructor.
oldAttribute	Set in constructor
newAttribute	Set in constructor.
level	Set in constructor.
activityType	Set in constructor.

Methods

There are standard getter methods for all the attributes, but no setters.

The constructor must pass all values, except date (populated with system date at point of instantiation). The MAC address is optional. alteredAccountId is optional if a global entity is altered (example, Currency).

ActivityType

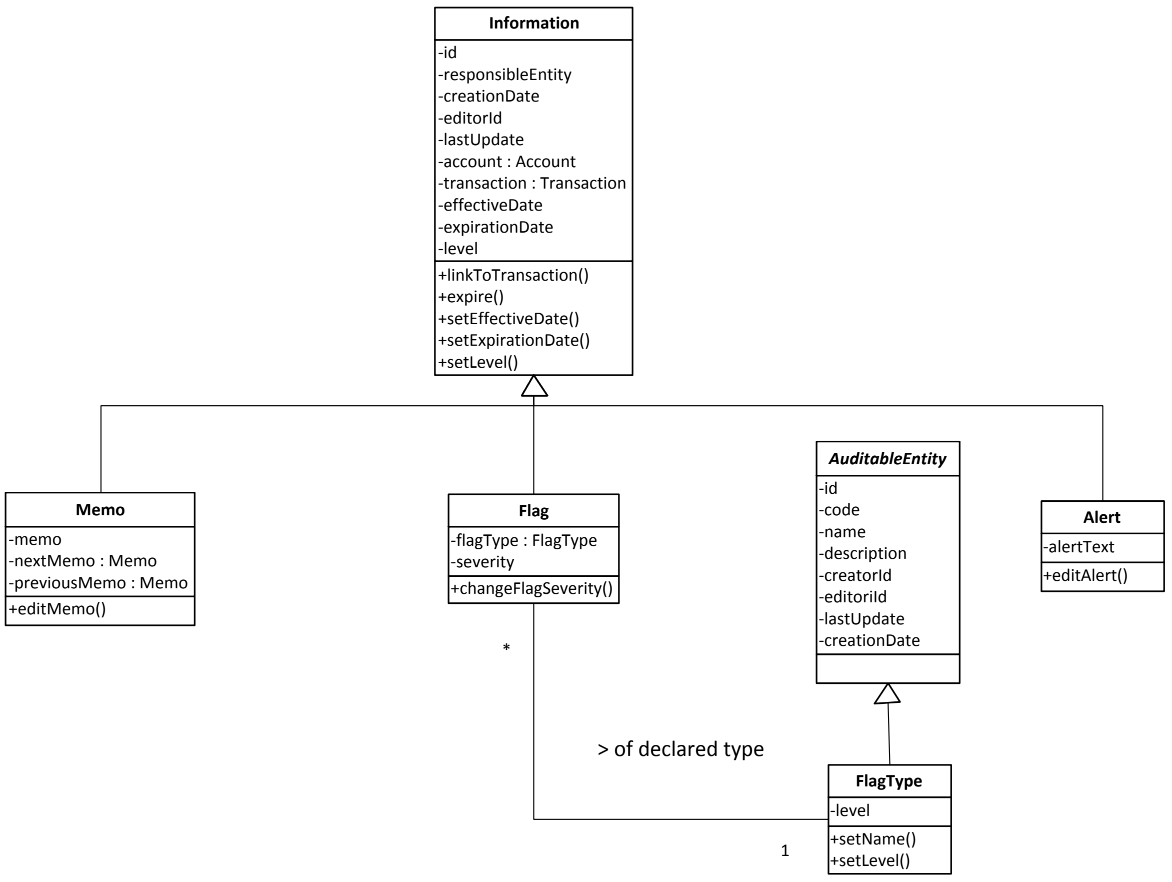
Attributes and Initial States

identifier	Set during instantiation.
activityName	Set in constructor.
creatorId	Set during instantiation.
editorId	Null
lastUpdate	System date.

Methods

editActivityTypeName (newActivityName)
Validate the name, then place into object.

Class Information Overview



Generic

The information class is a class that is used to store different types of information about either accounts or transactions. There are three basic types of information. Memos, which are freeform text items, which can be entered by CSRs, to provide a history of when and why certain things have happened on the account. Certain automated events may also trigger memos to be created (for example a fee assessment, or an online payment might trigger a memo event). Alerts are similar to memos, but they are displayed to the user in a different way, and are considered more transient. Alerts are freeform, and could be used for any number of purposes, but fundamentally, an alert is used to bring important information to the attention of the user. Flags are equally used to bring important information to the attention of the user, however, unlike alerts, flags are always of a pre-defined type, and can therefore be used as part of a rule making process. For example, a student might have bounced a check, so the "Bounced Check" flag is set. This value may be used in the rules system to decide if they are allowed to pay by check in the future. A severity attribute also allows the engine to distinguish between levels of a flag. To continue the example, a client who bounces a check once might have a low severity assigned, however, if a second check is bounced, the severity is set to high.

Class Information

Attributes and Initial State

Id	The unique identifier for the piece of information. Generated on instantiation.
responsibleEntity	Populated at instantiation.
creationDate	Established automatically at instantiation.
account	Established at instantiation
transaction	If applicable, established at instantiation. Otherwise set to null.
effectiveDate	Set on instantiation either through constructor or default to system date.
expirationDate	Initially set to null.
level	Set on instantiation. In the case of a flag, this value is derived from FlagType.level

Methods

Any method that alters the state of a method will trigger a change to editorId and lastUpdate.

correct(depends on override)
Overridden method in the child classes. Certain users have the ability to change the memos, flags and alerts of others. Doing so will trigger a change to lastUpdate and editorId
linkToTransaction(transactionId)
If a memo point, etc. points at a specific transaction, it can be assigned via this method.
expire()
Set the expiration date to current date.
setEffectiveDate (newEffectiveDate)
Validates that the newEffectiveDate is valid (especially with regards to any expirationDate and then sets the value.
setExpriationDate
Validates that the expiration date is after the effectiveDate and then sets the value.
setLevel(newLevel)
Assigns newLevel to level.

Class Memo

Attributes and Initial State

Memo	The actual plain text of the memo, set during instantiation.
nextMemo	Null. This value is set when createFollowUp() is called
previousMemo	Null unless created through createFollowUp()

Methods

correct (newMemo)
After validation, changes the memo to newMemo
createFollowUp(parameters will follow the instantiator for Memo)
Creates a new memo, and sets the followUpMemoId to the identifier of the new memo.

Class Flag

Attributes and Initial State

flag	Created by the system on instantiation.
severity	Set by the constructor.

Methods

setSeverity (newSeverity)
Changes the severity attribute to newSeverity.

Class FlagType

Attributes and Initial State

flagId	Created by the system on instantiation.
name	Set by the constructor.
creatorId	Set during instantiation.
level	Set during instantiation
description	Set during instantiaion.

Methods

setName (newName)
Changes the name of the flag.
setLevel (newLevel)
Set the level for the flag.

Class Alert

Attributes and Initial State

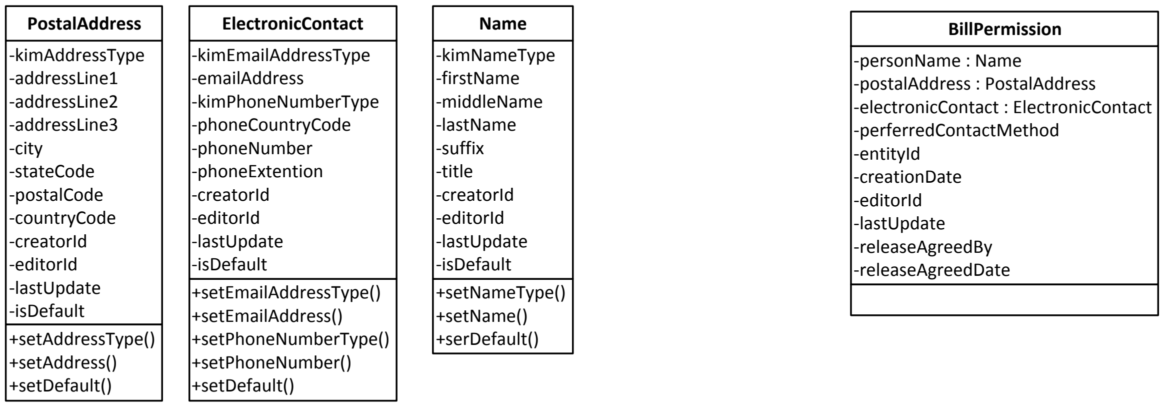
alertText	Set during instantiation.

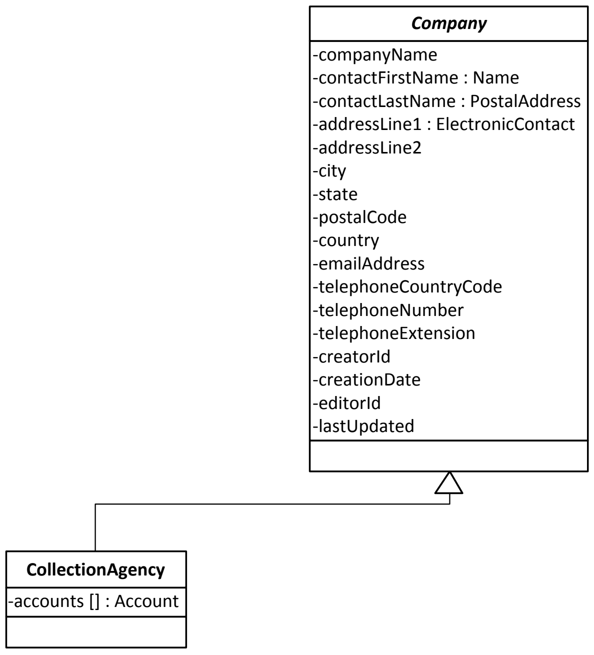
Class Account Overview

Helper Classes

The Account class is a central class that permits the system to tie together the transactions and other information into meaningful pieces of information that can then be used as the basis for billing, and for access to the system. In most instances, most of the information in the Account class is derived from KIM. Wherever possible, authorized changes to identity information made in KSA will be passed back to KIM.

The following helper classes are used to manage personal information relating to the different accounts. They are referenced in several of the account classes. The attribute initial values are either set by constructor, or are harvested from KIM.


PostalAddress
-kimAddressType
-addressLine1
-addressLine2
-addressLine3
-city
-stateCode
-postalCode
-countryCode
-creatorId
-editorId
-lastUpdate
-isDefault
+setAddressType()
+setAddress()
+setDefault()
ElectronicContact
-kimEmailAddressType
-emailAddress
-kimPhoneNumberType
-phoneCountryCode
-phoneNumber
-phoneExtention
-creatorId
-editorId
-lastUpdate
-isDefault
+setEmailAddressType()
+setEmailAddress()
+setPhoneNumberType()
+setPhoneNumber()
+setDefault()
Name
-kimNameType
-firstName
-middleName
-lastName
-suffix
-title
-creatorId
-editorId
-lastUpdate
-isDefault
+setNameType()
+setName()
+serDefault()
BillPermission
-personName : Name
-postalAddress : PostalAddress
-electronicContact : ElectronicContact
-perferredContactMethod
-entityId
-creationDate
-editorId
-lastUpdate
-releaseAgreedBy
-releaseAgreedDate


Company
-companyName
-contactFirstName : Name
-contactLastName : PostalAddress
-addressLine1 : ElectronicContact
-addressLine2
-city
-state
-postalCode
-country
-emailAddress
-telephoneCountryCode
-telephoneNumber
-telephoneExtension
-creatorId
-creationDate
-editorId
-lastUpdated
CollectionAgency
-accounts [] : Account

Class PostalAddress

Attributes and Initial State

kimAddressType	KIM is capable of storing several types of addresses. (Home, Office, Permanent, Billing). As KSA is not a system of record for the student address, it links its addresses to the KIM store. This field records the address type that was pulled from KIM. Is this field is null, then the address is a "local" address (i.e. local to KSA) and is therefore not copied from, or to be returned to KIM.
addressLine1	The first line of the address, in most cases, as returned from KIM.
addressLine2	The second line of the address, in most cases, as returned from KIM.
addressLine3	The third line of the address, in most cases, as returned from KIM.
city	The name of the city in which the address lies. No political distinction is made in the data model between different types of areas, as may be the case in certain locales.
stateCode	If the address contains some form of subdivision code (most commonly US states, Canadian provinces, etc.) it is stored here.
postalCode	If used, a postal code.
countryCode	ISO 3166 country code.
isDefault	False.

Methods

setAddressType (newKimAddressType)
Validate that the new KIM address type is valid for the student. If the address is appropriate (as defined by business rules, then get the new address from KIM and import it into the class.
setAddress (newAddress)
If the new address has a KIM type, import and attempt to update KIM. Where KIM cannot be updated (due to validation or policy) set the address type to null.

Class ElectronicContact

Attributes and Initial State

kimEmailAddressType	KIM is capable of storing several types of email address. (Home, work) As KSA is not a system of record for this information, it links its addresses to the KIM store. This field records the address type that was pulled from KIM. Is this field is null, then the address is a "local" address (i.e. local to KSA) and is therefore not copied from, or to be returned to KIM.
emailAddress	Set from KIM or by passed value.
kimPhoneNumberType	As with other KIM types, the type flag, to locate the phone number within the KIM system.
phoneCountryCode	Country code, stored in ISO format (gb, us, fr, sv, etc.)
phoneNumber	The full telephone number of the contact.
phoneExtension	An Extension to the phone system, if needed.
isDefault	False.

Methods

setEmailAddressType (newEmailAddressType)
Validates that the newEmailAddressType is valid, and exists for the KIM account in question, and is appropriate. If so, get the new address, and store.
setEmailAddress (kimEmailAddressType, emailAddress)
If permitted, check that KIM will accept the new email address. If not, set the KIM type to null, and set as the new email address.
setPhoneNumberType (newPhoneNumberType)
As for setEmailAddressType ()
setPhoneNumber (newPhoneNumberType, newPhoneNumberCountry, newPhoneNumber, newPhoneNumberExtension)
As for setEmailAddress()

Class Name

Attributes and Initial State

kimNameType	The name type as it is stored in KIM, or, if null, this is a local name, which will not be pulled back from KIM.
firstName	First name of the entity.
middleName	Middle name of the entity.
lastName	Last name of the entity.
suffix	Suffixed part of name, including generation codes.
title	Preferred title, ex. Mr. Miss, etc.
isDefault	False.

Methods

setNameType (newNameType)
Validate name type is valid in KIM and if so, fetch name, and store.
setName (newName)
Validate the name type if present against the KIM name types, and if allowed, update the name in both KIM and KSA.

Class BillPermission

Attributes and Initial State

personName	Set at instantiation
postalAddress	Set at instantiation.
electronicContact	Set at instantiation.
preferredContactMethod	Set at instantiation.
creationDate	Set by the constructor.
entityId	Set by the constructor.
editorId	Set by the setter methods.
lastUpdate	Set by the setter methods.

releaseAgreedBy	Set at instantiation.
releaseAgreedDate	Set at instantiation.

Class Company

An abstract class allowing the creation and storage of generic “Company” (non-human actors) in KSA.

Attributes and Initial State

tradingName	Set at instantiation.
contactName []	Name record, recording the person to contact.
Address []	PostalAddress record(s)
electronicContact []	ElectronicContact record(s)

Class CollectionAgency

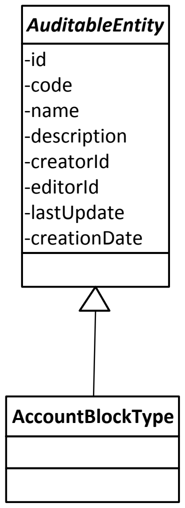
Attributes and Initial State

accounts []	Accounts assigned to this collection agency.

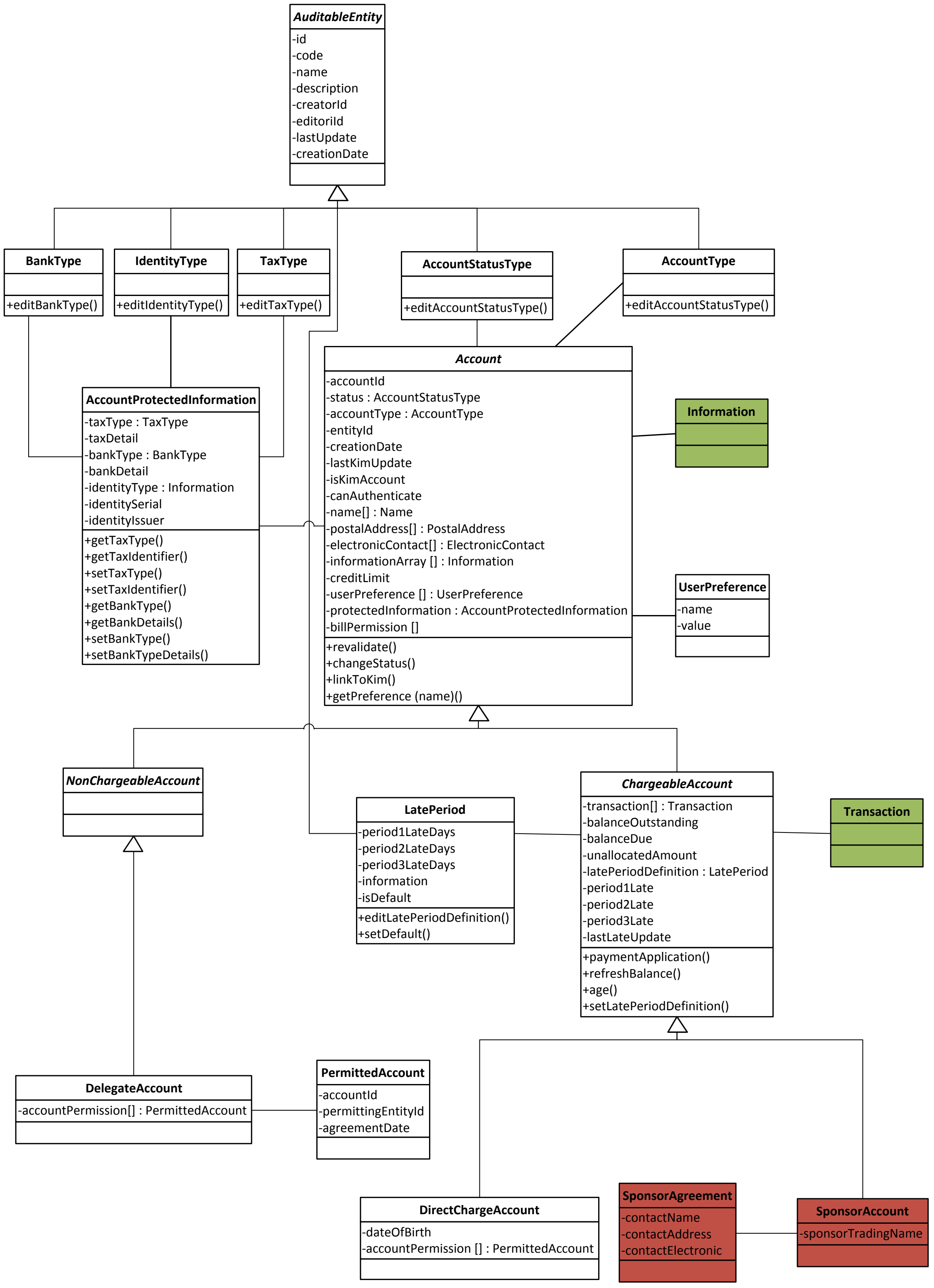
Class AccountBlockType

Generic

Simple auditable entity to store the types of account blocks available on the system.



Class Account



Generic

Accounts in KSA are one of the primary classes that tie together huge amounts of data within the system. All accounts have an accountId, which is used throughout the system to label transactions and memos as part of the same group. Accounts can either be ChargeableAccounts (ones that can have charges levied against them) or NonChargeableAccounts, for example, a parental account which exists solely to provide access to their son or daughter's account.

Class Account

Attributes and Initial State

accountId	Generated automatically by KSA
status	Null, set via method.
accountType	Null, set via method.
entityId	If the account is populated from KIM (an Authenticated account) then this value will be set during instantiation. If this is an unauthenticated account, this value will be null. If this is a non-KIM based account, coming from an external system, this will be the identifier from that system.
creationDate	Set to system date at instantiation. This is the date the account was created in the KSA system.
lastKimUpdate	When the account is a KIM account, this is set to creationDate on instantiation.
isKimAccount	Boolean. If true, the account is derived from KIM. If false, the account is KSA specific.
canAuthenticate	Boolean. If true, the account is permitted to authenticate into the KSA system. If false, this is not permitted.
name[]	Name of the account owner. See Class Name. Set during instantiation either by value, or derived from KIM.
address[]	Address of the account owner. See Class Address. Set during instantiation either by value, or derived from KIM.
electronicContact[]	Email and telephone number of account holder. See Class ElectonicContact. Set during instantiation either by value, or derived from KIM.
information[]	Array of type "information"
creditLimit	Set to default for system at instantiation.
userPreference	Array of objects of type UserPreference.
protectedInformation	Link to the protected information fields for the account.

Methods

revalidate()
If the account is a KIM account, then KSA pulls the identity attributes that it uses from KIM and refreshes the KSA account with that information.
changeStatus (newStatus)
Ensure that the new status is a valid status, then set status to newStatus.
linkToKim (kimEntityId)
If the account is not a KIM account (isKimAccount is false) then alter the attribute, and revalidate against KIM.

Class NonChargeableAccount

Abstract Class

Class Delegate Account

Attributes and Initial State

name	Name of the account owner. See Class Name. Set during instantiation either by value, or derived from KIM.
electronicContact	Email and telephone number of account holder. See Class ElectonicContact. Set during instantiation either by value, or derived from KIM.
accountPermission []	List of accounts that this account can access, and the permissions and restrictions based on that.

Methods

Class ChargeableAccount

Abstract Class.

Attributes and Initial State

transaction[]	Array of links to transactions. Empty at instantiation.
balanceOutstanding	0.00
balanceDue	0.00
unallocatedAmount	0.00
latePeriodDefinition	Set to the default LatePeriodDefinition
period1Late..period3Late	0.00
lastLateUpdate	Null at instantiation. Set to current date and time when ageing process runs.

Methods

paymentApplication()
Run the payment application process, then call refreshBalance()
refreshBalance()
Recalculate the balanceOutstanding, balanceDue and unallocatedAmount attributes.
age()
Set the periodxLate attributes according to the time definitions referenced in latePeriodDefinition using the ageing methodology as defined elsewhere in this document.
setLatePeriodDefinition(newLatePeriodDefinition)
Check validity of newLatePeriodDefintion. Trigger a memo. Change definition if valid, then call paymentApplication(), refreshBalance() and age()

Class DirectChargeAccount

Generic

The direct-charge account is the most common type of account. It could be generalized as a “student account” however, many institutions extend the use of these accounts to non-student actors (for example, a childcare facility extended to students might also allow faculty to use the service, and have their “student” account be billed for the charges. As such, the account is a direct-charge account, which is defined as an account which may be billed directly for charges. This would differ from a sponsor account, which is traditionally billed indirectly for activities accrued on another account.

Attributes and Initial State

dateOfBirth	Set during account setup from KIM or external source.
accountPermission[]	Set to empty at instantiation.

Methods

Class DelegateAccount

Special account type that can only be used to access third-party account information through the web portal. This type of account cannot accept charges

Attributes and Initial State

Class SponsorAccount

Attributes and Initial State

sponsorTradingName	Set during instantiation.
address	Address of the account owner. See Class Address. Set during instantiation either by value, or derived from KIM.
electronicContact	Email and telephone number of account holder. See Class ElectonicContact. Set during instantiation either by value, or derived from KIM.

Class LatePeriod

Generic

In general, accounts are aged on a 30/60/90 day period. In KSA, they are measured relative to the effectiveDate of the transaction. To provide flexibility, the LatePeriodDefinitions permits the creation of other late period types, to be applied to certain types of accounts. The envisioned use case is to permit sponsor accounts longer periods of time to pay.

Attributes and Initial State

latePeriodId	Autonumbered field.
name	Set by constructor.
period1LateDays.. period3LateDays	Set by constructor.
description	Set by constructor, information to tell admin when a late period is appropriate.
isDefault	False

Methods

editLatePeriodDefinition (newName, newPeriod1, newPeriod2, newPeriod3, newInformation)
Validate information, and then assign values within the object.
setDefault()
Find the current default period and set isDefault to false. Set this.defaultPeriod to true.

Class AccountStatusType

Account statuses will provide a very basic ability to enforce certain holds on the account. These are not yet defined.

Attributes and Initial State

Id	Autonumber set during instantiation.
name	Set in constructor.
description	Set in constructor.

Class AccountType

The account type is used to provide configurable account types to the system, so that an institution can stream account holders into different "buckets", for example by year of study, or by program, etc.

Attributes and Initial State

Id	Autonumber set during instantiation.
name	Set in constructor.
description	Set in constructor.

Class AccountProtectedInformation

Generic

AccountProtectedInformation stores potentially sensitive data. All access to this class is audited. These data are stored in a different table to other account information fields, and few accounts that have direct access to the database tables would have any type of access to this table. The class stores three major types of protected information, tax information (commonly the social security number in the US) banking information (usually ACH routing information in the US) and identity validation information (for example a passport or driver's license). These types of information can be configured in TaxType, BankType and IdentityType which are standard dictionary types in the KSA system.

Attributes and Initial State

taxType	Defined tax type, as stored in TaxType.
taxDetail	The actual tax identifier as described by TaxType.
bankType	The bank type being defined, as stored in BankType
bankDetail	The actual bank details, as described by bankType
identityType	The type of identity that has been seen.
identitySerial	The serial number of the document proving identity.
identityIssuer	The issuing authority responsible for the document.

Class UserPreference

Generic

UserPreference are used to store simple key/value pairs for the user. These can be accessed easily to allow the system to be configured to the individual user.

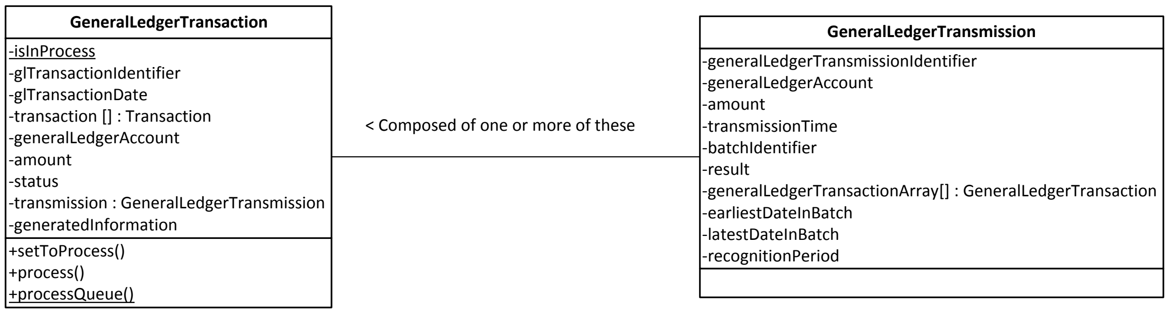
Attributes and Initial State

name	Set during instantiation.
value	Set during instantiation.

Methods

Setter and instantiation will cause a memo event to record the change.

Class GeneralLedger* Overview



Generic

As different types of transactions are recorded within the KSA module, frequently, those transactions have to also be recorded on the general ledger. A number of configurable factors go into the derivation of the correct amount and general ledger accounts to which a transaction is sent, and a fuller description of these factors can be found in the explanation of the Transaction class. Once it is discerned that a transaction will be transmitted to the general ledger, and the appropriate accounts and amounts have been discovered, a GeneralLedgerTransaction is instantiated. This records the value being transmitted to the general ledger, as well as a reference to the KSA transaction that causes the general ledger transaction(s).

The GeneralLedgerTransmission class is responsible for the actual transmission of transactions to the general ledger. Depending on configuration, transactions may be transmitted in real time, in batch by transaction, or in rolled up batches.

Class GeneralLedgerTransaction

Attributes and Initial State

isInProcess	A class-level Boolean. If it is true, then the system will not permit a new transmission until this transmission is complete.
glTransactionIdentifier	A unique transaction identifier for this transaction. This is assigned at instantiation.
glTransactionDate	Datetime stamp when the general ledger transaction was generated by KSA.
transaction []	Pointer to the transaction that generated this entry. Passed by the constructor. If the GL transaction is collapsed from a group of transactions by the session cleanup after payment application, this can be an array of transactions.
generalLedgerAccount	Passed by the constructor.
amount	Passed by the constructor.
status	Defaults to Q (queue) or W (Waiting). Other values as the transaction processes are P (set to process) and C (completed processing). F (failed) means the transaction did not get sent to the general ledger.
transmission	Null. Pointer to the GeneralLedgerTransmission. This value will be

	null until status is C.
generatedInformation	Passed by the constructor. This string is used to assist a power user to understand where the transaction was generated. In most cases, GL transactions will be generated through the makeEffective() method, which will create general ledger transactions, or by the paymentApplication() method.

Methods

setToProcess ()
If the status is Q, and the class is not currently processing, then set the status to P.
process()
So long as the system isn't currently processing, process this transaction to the general ledger.
processQueue()
Class level method, causes the system to process all queued transactions.

Class GeneralLedgerTransmission

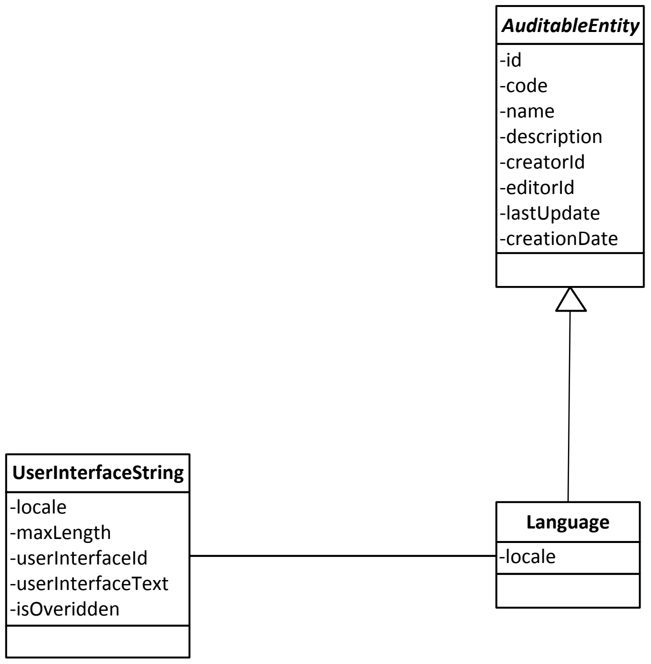
Attributes and Initial State

generalLedgerTransmissionIdentifier	Autonumbered field set during instantiation.
generalLedgerAccount	Set by constructor
amount	Set by constructor.
transmissionTime	Null at instantiation. Set when the actual transmission is made to the GL.
batchIdentifier	Null at instantiation. Set when the transmission is made to the GL in batch mode.
result	Null at instantiation. Set after the transmission is made to the GL.
generalLedgerTransactionArrray	Set by the constructor.
earliestDateInBatch	Set by the constructor.
latestDateInBatch	Set by the constructor.
recognitionPeriod	Null. May be calculated by rule set before transmission.

Class UserInterfaceString and InstalledLanguage Overview

Generic

These classes support UI localization by permitting simple lookups against a locale parameter and a UI identifier. So long as the appropriate language pack has been loaded, the system should be able to communicate with the user in that language. There is also a simple dictionary service called Language that can take the language code and give a name for use in the UI.



Class UserInterfaceString

locale	Passed by the constructor.
maxLength	Passed by the constructor.
userInterfaceId	Passed by the constructor.
userInterfaceText	Passed by the constructor. If the value is set by a setter, then isOveridden should be set to TRUE.
isOverriden	Set to false if the string comes from an XLIFF file. If the string is overridden internally, then this is set to true. The value of this alters how the system deals with imports after strings have been changed.

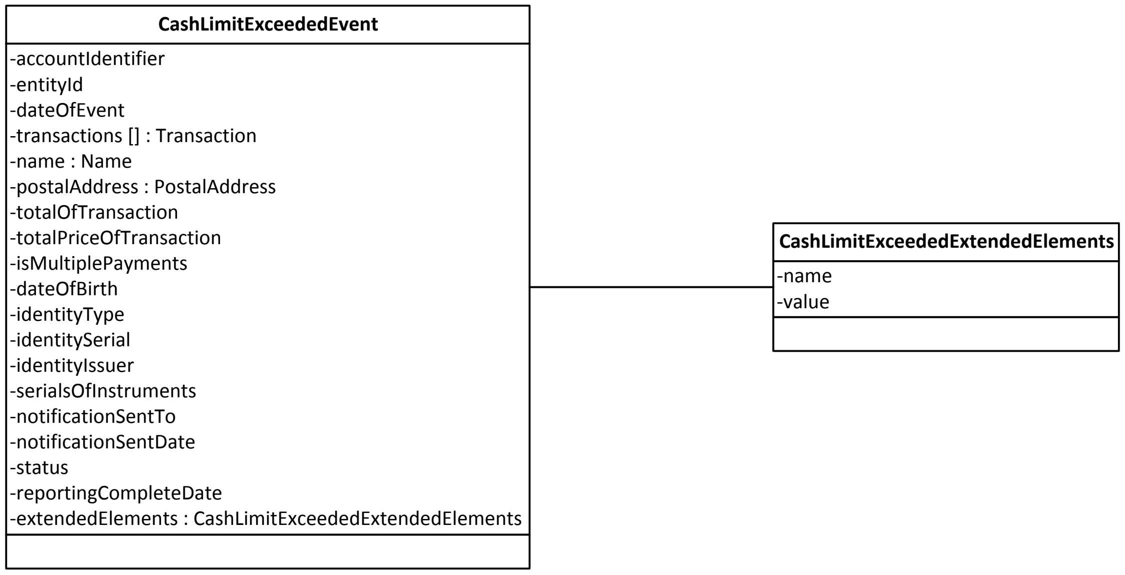
Class Language

locale	Specific code used to refer to a language, passed by the costructor.

Class CashLimitExceeded Overview (In progress)

This class exists to store information on accounts that have exceeded the amount of cash that can be passed through them without flagging the transaction to the IRS. The system is designed specifically to monitor the “combined cash limit tracking” as required by U.S. law, and to ease the filing of IRS form 8300. However, the system has been designed to be flexible enough to be used under other tracking systems that may exist in other countries. The cash limit, time period over which it is tracked, and which types of transaction are traced over the period of time.

Many of these attributes could be extracted by reference, however, this class is used to take a snapshot of the account at the time when the event happened, so that it can easily be transformed into a form and sent to the IRS or other body.



Class CashLimitExceededEvent

Attributes and Initial States

accountIdentifier	Account to which the overage relates.
entityId	The user who placed the transaction on the account that triggered the cash-limit flag.
dateOfEvent	The date on which the event (transaction that caused the event) occurred.
name	Name of the account holder (complex element).
transaction []	Array of transactions that are included in this report.
taxpayerIdentifier	As this system is designed to track information for the IRS' 8300 form, this will default to the U.S. social security number if it can be found.
postalAddress	Identifier for the address object that holds the account holder's address.
totalOfTransaction	Sum of all the transactions that have caused the event. Note that this number might be altered during the processing logic due to IRS reporting rules.
totalPriceOfTransaction	//Still being questioned with the stakeholders. The exact calculation for this field is not yet known, but is required for box 31 on the 8300.
isMultiplePayments	Boolean indicating whether or not this transaction is a single transaction or is composed of multiple transactions.
dateOfBirth	Date of birth of the account holder.
identityType	Type of document used to identify the account holder. (For example,

	driver's license)
identitySerial	Serial number of the identity document. (For example, driver's license number.)
identityIssuer	Name of the issuer of the document. (For example, State of Maryland.)
serialsOfInstruments	Serial numbers of any known monetary instruments used in the transaction.
notificationSentTo	Email address of the person to whom the email notification was sent (derived from cash.tracking.notify)
status	Status of the transaction. Q for queued, C for complete, I for ignored. Note that due to IRS reporting requirements, an event may be triggered by the system, but during processing, the rules are not met. If this is the case, the status will be set to I. See cash-tracking logic in the process document for more details.
reportingCompleteDate	Date when the legal reporting requirement was met. For the initial use of this feature, this will mean the date on which the 8300 was filed.
extendedElements[]	Array of type CashLimitExceededExtendedElements used to store extended values for the cash-limit tracking. These values are manipulated during the processing logic.

Class CashLimitExceededExtendedElements

Simple key-pair value class linking to the CashLimitExceededEvent. This is used by the rules engine to establish other values that are used to complete the cash-limit tracking form (8300 in the US.)

Attributes and Initial States

name	Name of element.
value	Value of element.

In the reference implementation, the following values will be calculated and completed in this class but the rules engine. These values refer to the values required on the US 8300. These relate to box 32 of the 8300 form.

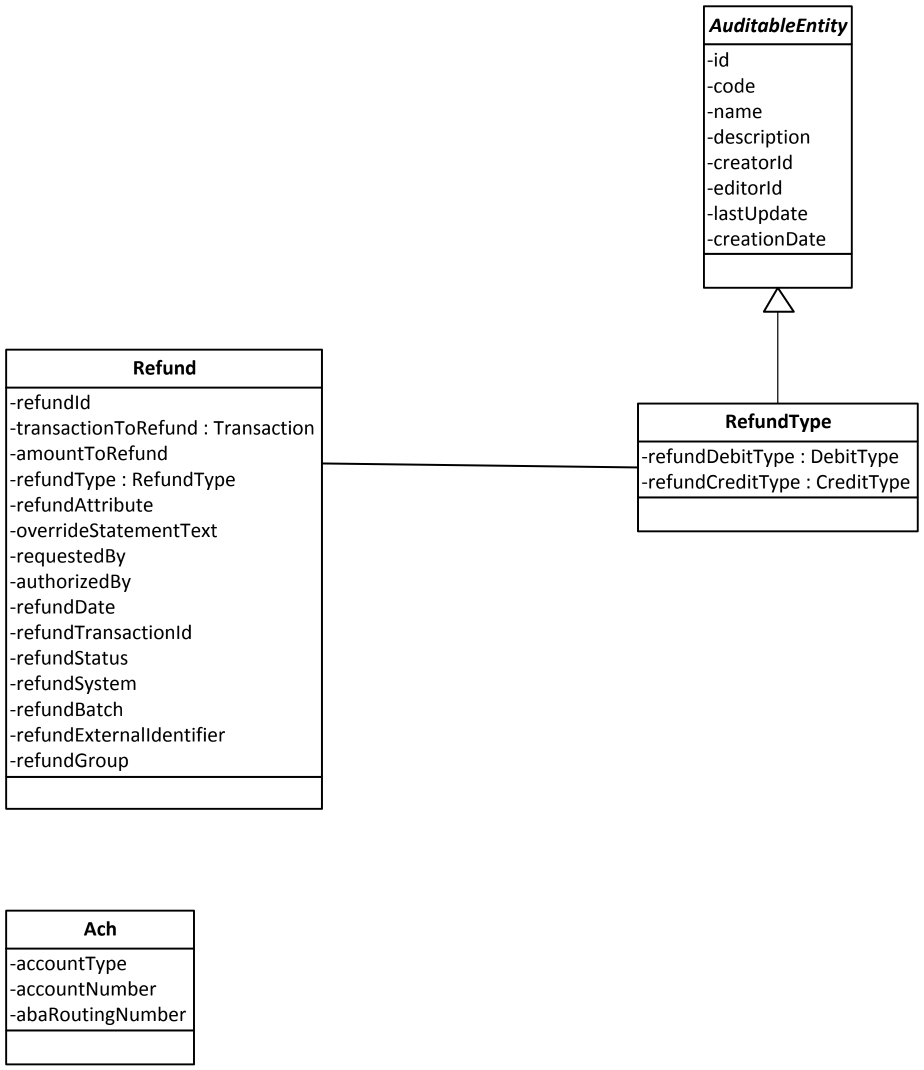
nativeCurrencyAmount	Amount of the transaction composed in native currency.
foreignCurrencyAmount	Amount of the transaction composed in a foreign currency.
foreignCurrencyIdentifier	Currency used in the foreign transaction.
cashiersChecksAmount	Amount of the transaction composed of cashier's checks.
moneyOrderAmount	Amount of the transaction composed of money orders.
bankDraftAmount	Amount of the transaction composed of bank drafts.
travelersChecksAmount	Amount of the transaction composed of traveler's checks.

Class Refund Overview (in progress)

Generic

The refund class is used to record the issuance of refunds. Refunds may be provoked by a number of actions, automated batch processed, user request, CSR request, but essentially, the system will go through a rule-based procedure to calculate which unused payments are eligible for refunds, how they are eligible (not all refunds happen in the same way, for example, check and cash payments may be refunded as a check, credit card payments may be refunded to the credit card, etc.)

Once the refund has been entered into this class, it will be picked up by the appropriate subsystem that will generate the actual refund, which will generate the actual refund transaction, and create a locked allocation between the refund and the payments that were refunded.



Class Refund

Attributes and Initial States

refundId	Autonumber identifier provided by at instantiation.
transactionToRefund	Identifier of the transaction that is being refunded. Furnished at by constructor.
amountToRefund	Value of the refund in system currency.
refundType	Pointer to the type of refund being issued. Passed by constructor.
refundAttribute	Depending on the type of refund, the attributes associated with creating that refund. For example, for an ACH refund, the ACH bank information needs to be passed. Null at instantiation unless passed by a constructor.
requestedBy	Entity identifier of the user who requested the refund. Likely values will be the user who ran a refund job, or the student requesting their own refund. Passed by the constructor.
authorizedBy	The user who authorized the refund. Null at instantiation.
refundDate	The date the refund was applied to the KSA account. Null at instantiation.
refundTransactionId	The transaction identifier for the produced transaction. This will be null at instantiation until the actual refund is produced. Note that a number of transactions may have a single refundTransactionId, as one large refund may be issued to cover several smaller credits.
refundStatus	A refund may be Unverified, Verified, Failed, Cancelled or Refunded. Once a transaction is refunded, the refundTransactionId must be completed. Set to U at instantiation.
refundSystem	Null at instantiation.
redundBatch	Null at instantiation.
refundExternalIdentifer	Null at instantiation. When the refund is completed, this value can be set to any external identification that shows the refund. For example, if the refund code from the credit card company is returned, it can be stored here.
refundGroup	If several refunds are issued under a single refund header (i.e. four transactions are refunded, but only one check is issued) then the refund group of all the refunds will be the same. This will be a UUID.

Class RefundType

Attributes and Initial States

refundDebitType	The transaction type for refunds that are made by this type. This is a debit type (“charge”), as it deducts the overpayment from the account.
refundCreditType	For refund types that send the refund back to a KSA account (“Account Refunds”, “Payoff Refunds”) this is the credit type (“payment”) that will be applied to the other account. For most refund types where the money is sent

Class Ach (Helper Class)

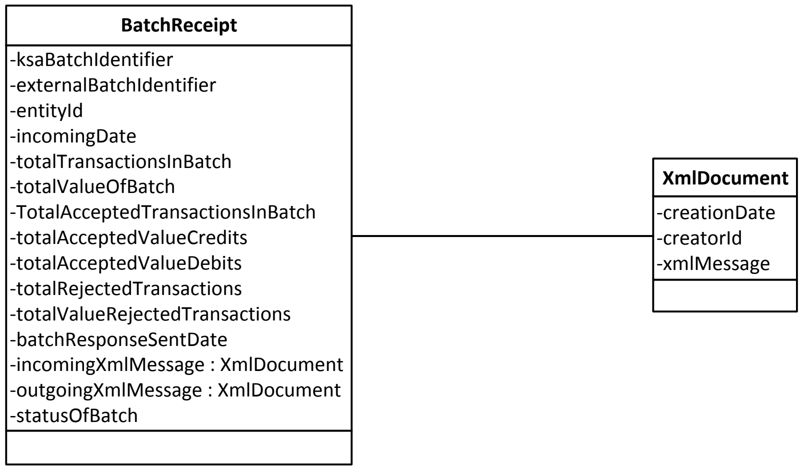
Simple helper class to allow us to store ABA routing numbers. This is a non-persisted class, used to unwrap banking details from the AccountProtectedInformation class.

accountType	C for checking, S for savings.
accountNumber	Account number (up to 20 digits). (17 is max for ACH)
abaRoutingNumber	9-digit ABA routing number.

Class BatchReceipt Overview

Generic

BatchReceipt is a simple class that records the status of batches of transactions that have been sent to the KSA system. Once a batch has been sent to the system, and it has been checked to see if it has already been transmitted, an object is instantiated to reflect this batch. The Xml class is a simple helper class used to store XML transmissions to/ from the system.


BatchReceipt
-ksaBatchIdentifier
-externalBatchIdentifier
-entityId
-incomingDate
-totalTransactionsInBatch
-totalValueOfBatch
-TotalAcceptedTransactionsInBatch
-totalAcceptedValueCredits
-totalAcceptedValueDebits
-totalRejectedTransactions
-totalValueRejectedTransactions
-batchResponseSentDate
-incomingXmlMessage : XmlDocument
-outgoingXmlMessage : XmlDocument
-statusOfBatch
XmlDocument
-creationDate
-creatorId
-xmlMessage

Class BatchReceipt

Attributes and Initial States

ksaBatchIdentifier	Set to the next KSA batch number at instantiation.
externalBatchIdentifier	Passed by constructor.
entityId	Passed by the system, reflecting the authenticated user or system that uploaded the batch.
incomingDate	DateTime, set at time of instantiation.
totalTransacitonsInBatch	Null
totalValueOfBatch	Null
totalAcceptedTransactionsInBatch	Null
totalAcceptedValueCredits	Null
totalAcceptedValueDebits	Null

totalRejectedTransactions	Null
totalValueRejectedTransactions	Null
batchResponseSentDate	Null
incomingXmlMessage	Null
outgoingXmlMessage	Null
statusOfBatch	Set to “Q” at instantiation.

Class XmlDocument

Attributes and Initial States

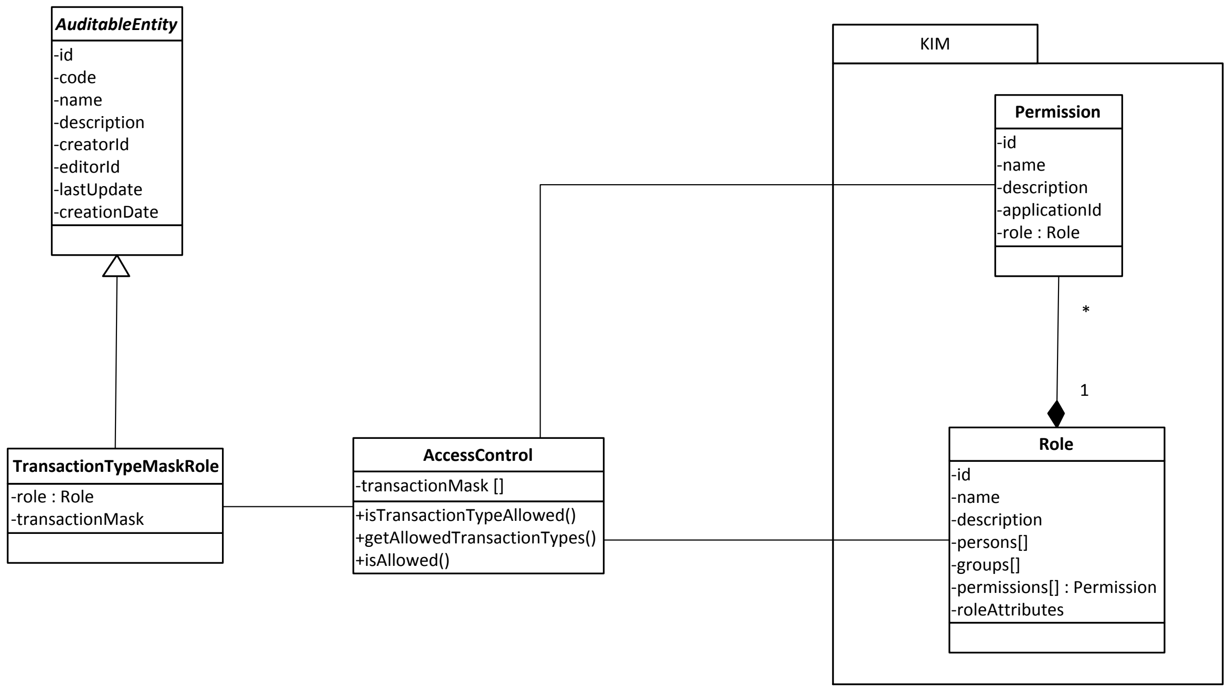
creationDate	Date that the XML message was created in the KSA system. For incoming XML, this will be the date/time that the message was stored.
creatorId	Entity identifier for the creator of the message (sender)
xmlMessage	The raw XML message.

Class AccessControl Overview (In Progress)

Generic

AccessControl is a KSA access point to KIM permission information. Most KIM permissions translate to simple Boolean permissions, which are documented in the Security Permissions document. These security permissions are mapped to methods that can answer questions about the logged in user, such as “canEditAccount” etc.

In addition, certain permissions can be used to designate access to certain ranges within KSA. This system is used to enforce policies relating to which transaction types a user has access to. Other access restrictions can be added through the AccessControl class.



Class AccessControl

This is a stateless class which is not persisted. The only class that is persisted is the TransactionTypePermission class, which stores the relationship between roles in KIM and allowable transaction types in KSA.

Attributes and Initial States

entityId	Entity to which these permissions apply. This is derived from the user identity in the session.
permission []	Array of Permissions in the KIM classes.
transactionMask []	Array of transaction masks, derived from Role/TransactionTypePermission mapping.
role []	Role of the user

Methods

isTransactionTypeAllowed(transactionTypeIdentifier)
Based on the current user, is the transaction type passed allowed.
getAllowedTransactionTypes()
Return a list of permissible transaction types for this user.
isAllowed (permissionName)
Returns true if the permissionName passed is associated with this user. False if not.

Class TransactionTypeMaskRole

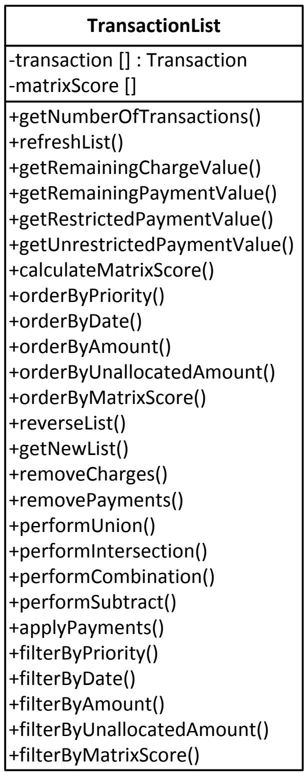
Note that this is a standard Auditable Entry. The name and description fields are set to allow a power user to easily select this type of permission. For example,

role	Role that maps to this transaction mask.
transactionTypeMask	Transaction mask defined as a Java regular expression. Where a user the given role, they are allowed to use any TransactionType where TransactionType.transactionTypeIdentifer matches the regular expression in this attribute.

Class TransactionList

Generic

The TransactionList is a wrapper for the Transaction class that provides easy access to groups of transactions so that they may be accessed more easily from the rules-driven payment application process. The class is not persisted, and most of its attributes are linked directly from the Transaction class. Most getter methods pull their values from the actual transaction. The class also allows the storage of the matrixRestrictionScore which is only calculated on-the-fly during payment application.



Attributes and Initial States

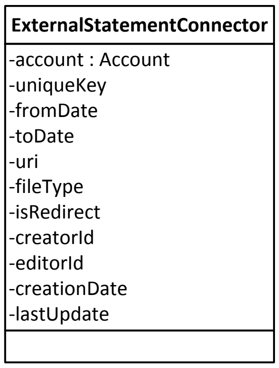
transaction []	Array of transactions in this list.
matrixScore []	Array of matrix scores for the transactions.

For method definitions, see the process document under the Payment Application Service.

Class ExternalStatementConnector

Generic

The external statement connector allows KSA to display statements that are generated in an external system. KSA is built on the principle that all communications in and out of the system are in XML, allowing schools to customize the look and feel of customer-facing items such as bills, etc. This class and the associated service allows an external piece of software to produce the bills in a format that can be consumed by users (for example, PDF statements). The software can then produce the statements and store them, and inform KSA where those statements are located so that they can be accessed by the user.



Attributes and Initial States

account	Account to which the bill belongs.
uniqueKey	Set by constructor. This is a key that allows an unauthenticated user to view the bill, if the system is configured to allow unauthenticated access.
fromDate	Set by constructor.
toDate	Set by constructor.
uri	Set by constructor.
fileType	Type of the file at this location. An external may wish to provide the statement in different formats. This field permits the system to track the different versions.
isRedirect	Boolean that tells the UI how to deal with the URI. If this is true, then the system will redirect the user to the URI listed. If false, the URI will be fetched, and sent to the user.