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.

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. lastUpdate is the date/time stamp when the entity was last changed. Many objects also have a “description” field, which is primarily used in the UI to assist the user in making a decision. 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.

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

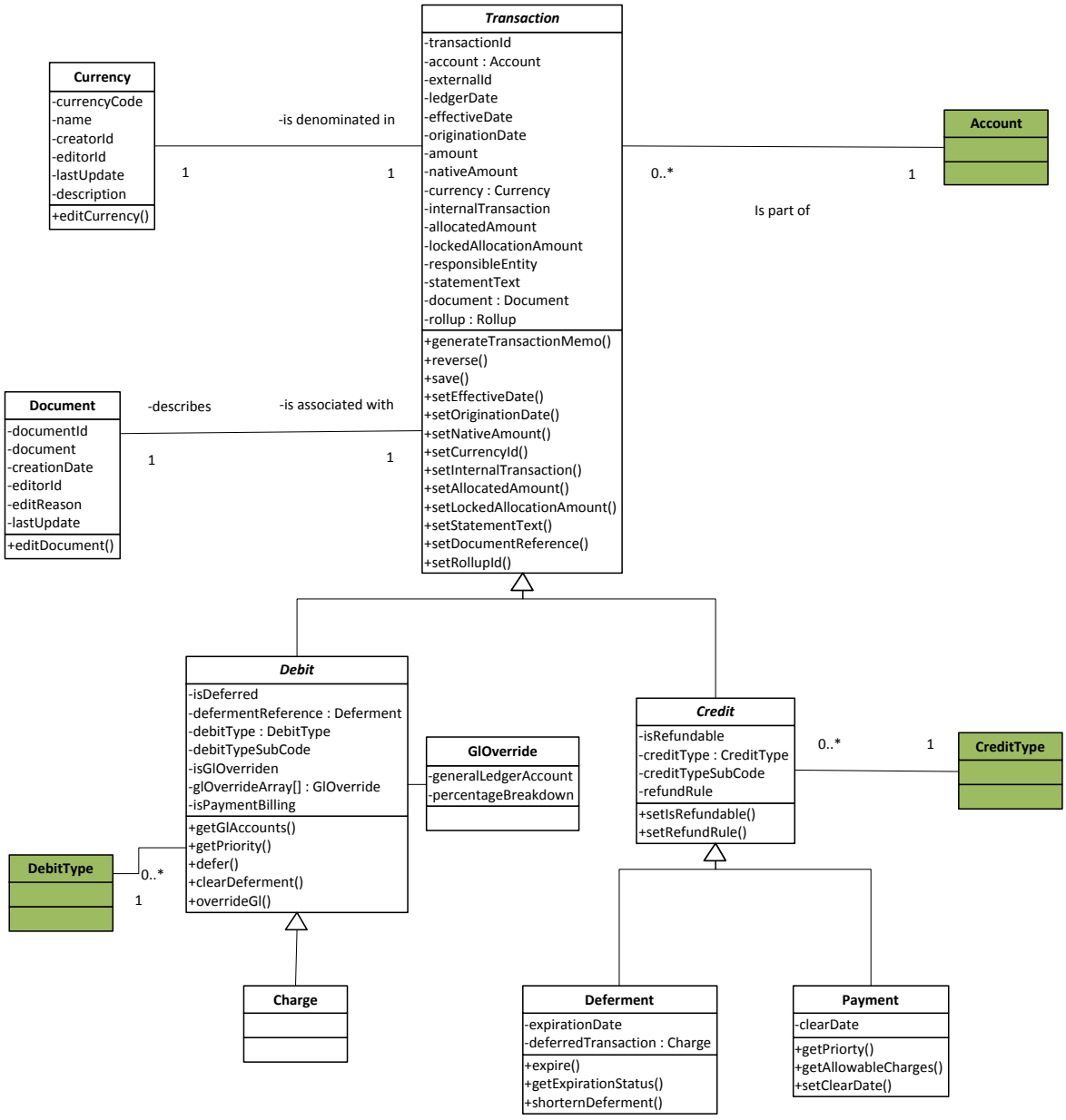
There are a number of “Type” classes, which are considered to be self explanatory.

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

There is a distinction between accountId, which is a generated ID within the KSA system, and other entity references (entityId, responsibleEntity, etc.) which reference authenticated responsibility for something. These will use the KIM entity identity for their values.

Attribute definitions are included where they are KSA classes.

Class Transaction



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.

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 of instantiation.
nativeAmount (*)	As amount.
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.

Credits and Children

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

Deferments Only

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.
isGlOverriden (*)	Null by default. Set through calls to the overrideGl() methods.
glOverrideArray[] (*)	If the General Ledger override is true, then this is an array of GlOverride objects that is consulted instead of the default DebitType general ledger account lists.
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

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

Attributes and Initial States

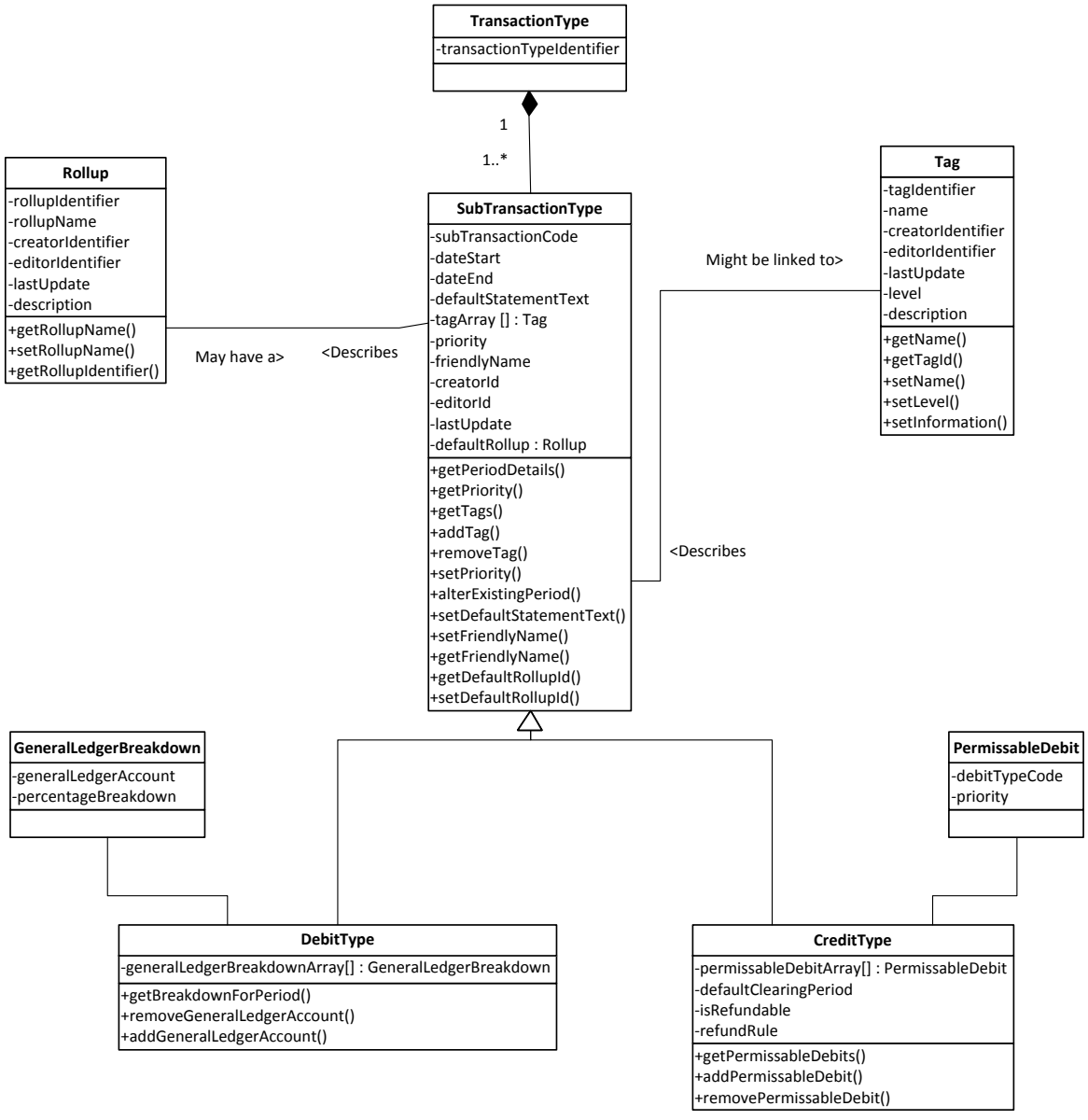
If the following fields are not provided, they will be set to default values as listed. Some fields are automatically set on certain events.

documentId	Field is populated on instantiation.
document	The document is passed during instantiation.
creationDate	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



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.

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.
dateStart	Unless otherwise provided, this is set to the current date.
dateEnd	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.
creatorId	Set to the entity id of the user who instantiated the sub-transaction type.
editorId	Null
lastUpdate	Assigned the system date.
defaultRollup	Null

Debit Type Only

generalLedgerBreakdownArray	Null. This array is populated through the addGeneralLedgerAccount() method.

Credit Type Only

permissableDebitArray	Null. Set via method.
defaultClearingPeriod	0 unless overridden.
isRefundable	True unless overridden.
refundRule	Null unless overridden.

Methods

getPeriodDetails()
Returns the transactionTypeIdentifier, dateStart, dateEnd and defaultStatementText for the SubTransactionType
getPriority()
Returns priority
getTags()
Returns a list of the tags associated with the Transaction Type
addTag(tagId)
Verifies that the tagId isn't already in the tagArray[] and if it is not present, adds the tagId to the tagArray[]
removeTag(tagId)
Verifies that the tagId is in the array, if removes it if it is present.
setPriority(newPriority)
Ensures that newPriorty is within range (positive) and then sets priority to newPriority
alterExistingPeriod (newDateStart)
Check the newEndDate is within range (after the current dateStart) and then set the value.
alterExistingPeriod (newDateStart, newDateEnd)
Check the validity of the dates with other sub transactions to ensure that the dates do not conflict, and then set the dates as appropriate.
setDefaultStatementText (newDefaultStatementText)
Ensure a non-null parameter, then set defaultStatementText.
setName (newName)
Ensure a non-null parameter, then set name.
getName ()
Return name
setDefaultRollupId (newRollupId)
Ensure that the rollup id is valid, and then set the defaultRollupId to newRollupId
getDefaultRollupId ()
Return defaultRollupId

DebitType only

getBreakdownForPeriod()
Returns the generalLedgerArray[] which includes both the general ledger account and the breakdown of the transactions.
removeGeneralLedgerAccount(generalLedgerAccountToRemove)
Ensure that the generalLedgerAccountToRemove is in the generalLedgerArray[], and then remove it.
addGeneralLedgerAccount (newGeneralLedgerAccount, percentageAllocation)
According to configuration, check that the GL account exists via a call to the GL. Then check that the

newGeneralLedgerAccount does not already exist in the generalLedgerArray[]. If not, add the new account and allocation to the array.
reassignGeneralLedgerAllocation (generalLedgerAccount, newAllocation)
Check that newAllocation is not over 99%, and the generalLedgerAccount is in the generalLedgerArray[] and then change its allocation to newAllocation.
validate()
Check that (a) the generalLedgerArray[] percentage does not exceed 99%, and (b) there is at least one, and only one 'bucket' account, with an allocation set to null.
save()
Call validate(), and if the breakdown is valid, persist the object.

CreditType only

getPermissableDebits()
Returns an array of permissible debits that a credit may pay, listed in order or priority. The debit array is, as yet, not entirely defined, however, it is expected that there will be some level of masking permitted, so for example, TUT### would permit the payment of TUT000 to TUT999. The priority level is a numerical value between 0-100, where 100 is paid off as top priority, and 0 is paid off as last priority.
addPermissableDebit (permissableDebitType, priority)
Check that the exact permissableDebitType does not exist in the array. (Masked versions are acceptable, as a school may choose to prioritize TUT100 over TUT### if they so choose, even though TUT100 is a member of TUT###. If the exact match is not found, Add the permissableDebitType and priority to the permissableDebitArray[]
removePermissableDebit (permissableDebitType)
Check that permissableDebitType exists in permissableDebitArray[], and if found, remove it.

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

rollupIdentifier	Populated at instantiation
rollupName	Populated at instantiation.
creatorIdentifier	Derived from creating entity.
editorIdentifier	Null
lastUpdate	System date at instantiation

Methods

getRollupName()
Returns rollupName

setRollupName (newRollupName)
Sets rollupName to newRollupName, and populates editorIdentifier and lastUpdate with new details.
getRollupIdentifier ()
Returns rollupIdentifier

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

generalLedgerAccount	Set by constructor.
percentageBreakdown	Set by constructor.

Methods

Class PermissableDebit

Generic

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

Attributes and Initial States

debitTypeCode	Set by constructor.
priority	Set by constructor.

Methods

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

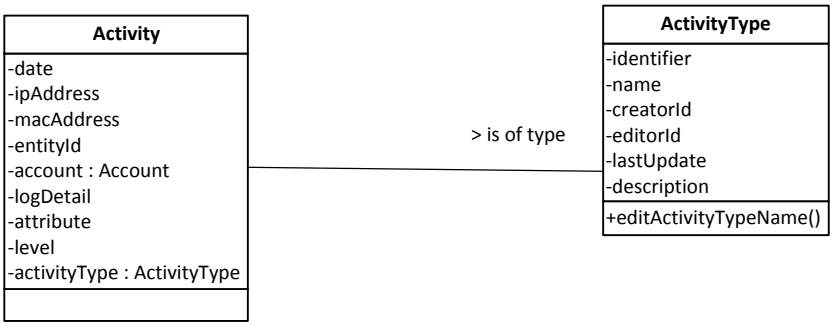
Methods

getTagName()
Returns tagName
setTagName (newTagName)
Sets tagName to newTagName, and populates editorIdentifier and lastUpdate with new details.
getTagIdentifier ()
Returns tagIdentifier
setLevel (newLevel)
Changes the level of the tag.
setInformation (newInformation)
Sets the information associated with the tag.

Activity

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.
macAddress	Set in constructor.
entityId	Set in constructor.
logDetail	Set in constructor.
attribute	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.

ActivityType

Attributes and Initial States

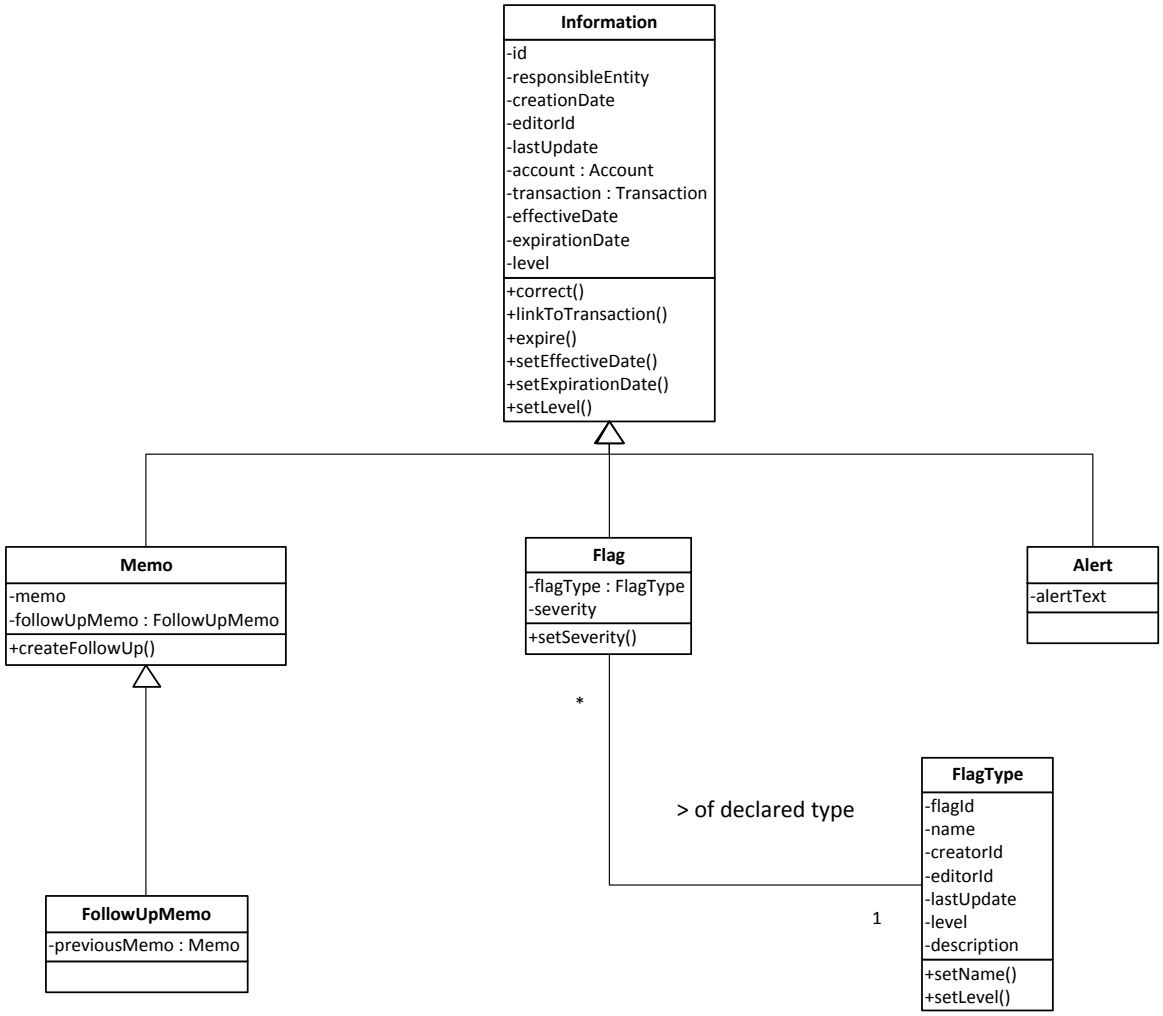
identifier	Set during instantiation.

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



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.
followUpMemoId	Null. This value is set when createFollowUp() is called

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 FollowUpMemo

Attributes and Initial State

previousMemo	Set to the identifier of the previous memo on instantiation.

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

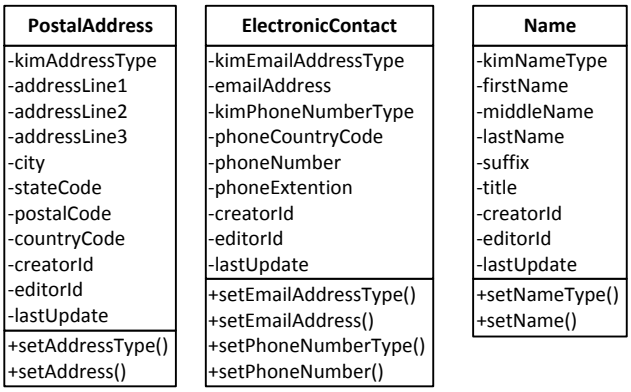
Attributes and Initial State

alertText	Set during instantiation.

Helper Classes for Account [Very much still under construction]

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.



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.

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.

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

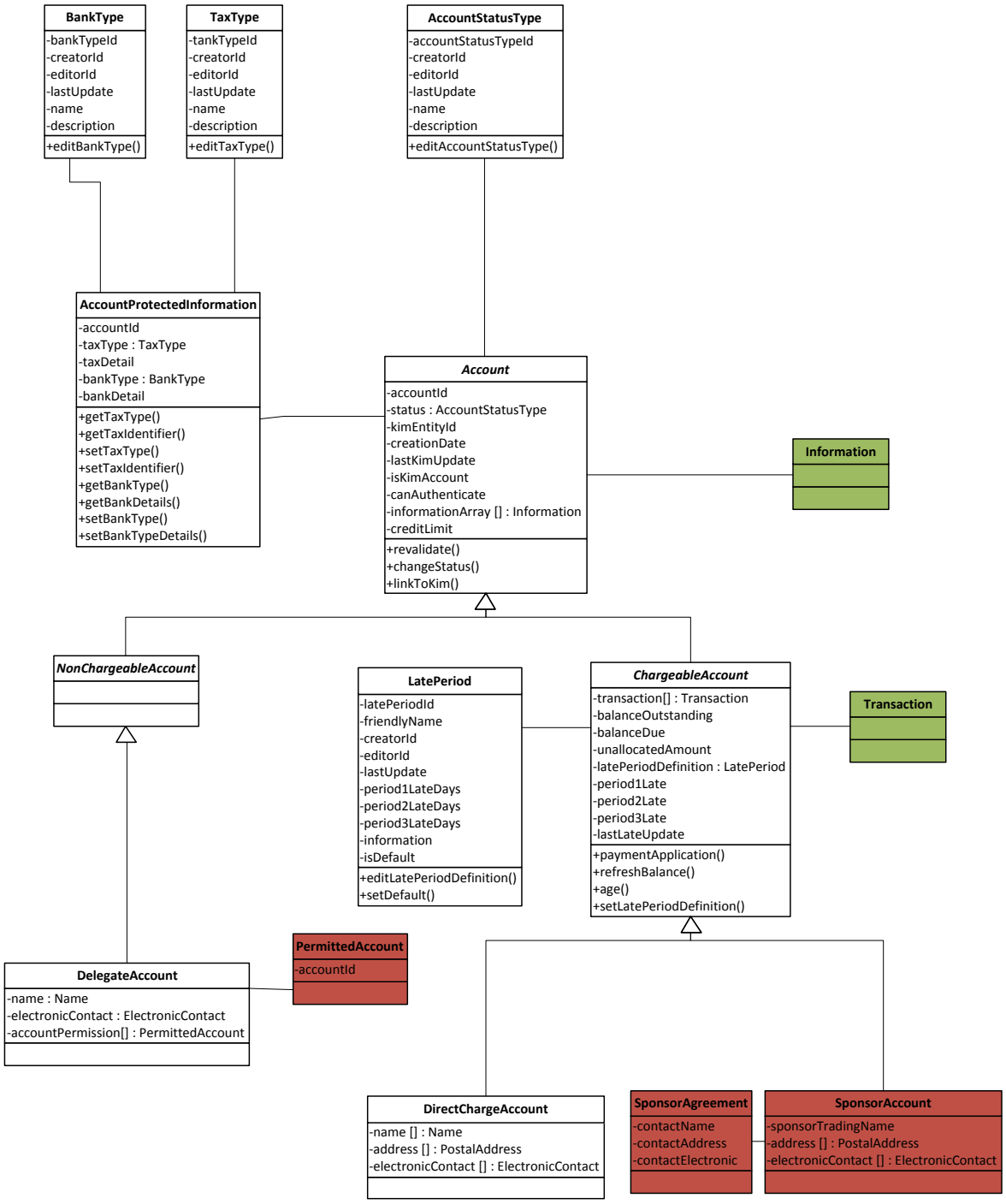
Attributes and Initial State

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.

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 Account


BankType
-bankTypeId
-creatorId
-editorId
-lastUpdate
-name
-description
+editBankType()
TaxType
-tankTypeId
-creatorId
-editorId
-lastUpdate
-name
-description
+editTaxType()
AccountStatusType
-accountStatusTypeId
-creatorId
-editorId
-lastUpdate
-name
-description
+editAccountStatusType()
AccountProtectedInformation
-accountId
-taxType : TaxType
-taxDetail
-bankType : BankType
-bankDetail
+getTaxType()
+getTaxIdentifier()
+setTaxType()
+setTaxIdentifier()
+getBankType()
+getBankDetails()
+setBankType()
+setBankTypeDetails()
Account
-accountId
-status : AccountStatusType
-kimEntityId
-creationDate
-lastKimUpdate
-isKimAccount
-canAuthenticate
-informationArray [] : Information
-creditLimit
+revalidate()
+changeStatus()
+linkToKim()
Information
NonChargeableAccount
LatePeriod
-latePeriodId
-friendlyName
-creatorId
-editorId
-lastUpdate
-period1LateDays
-period2LateDays
-period3LateDays
-information
-isDefault
+editLatePeriodDefinition()
+setDefault()
ChargeableAccount
-transaction[] : Transaction
-balanceOutstanding
-balanceDue
-unallocatedAmount
-latePeriodDefinition : LatePeriod
-period1Late
-period2Late
-period3Late
-lastLateUpdate
+paymentApplication()
+refreshBalance()
+age()
+setLatePeriodDefinition()
Transaction
PermittedAccount
-accountId
DelegateAccount
-name : Name
-electronicContact : ElectronicContact
-accountPermission[] : PermittedAccount
DirectChargeAccount
-name [] : Name
-address [] : PostalAddress
-electronicContact [] : ElectronicContact
SponsorAgreement
-contactName
-contactAddress
-contactElectronic
SponsorAccount
-sponsorTradingName
-address [] : PostalAddress
-electronicContact [] : ElectronicContact

FOR INITIAL PURPOSES, ONLY THE DIRECT CHARGE ACCOUNT IS IMPORTANT.

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.
kimEntityId	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.
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.
information[]	Array of type "information"
creditLimit	Set to default for system at instantiation.

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

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.

Methods

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.

Methods

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.

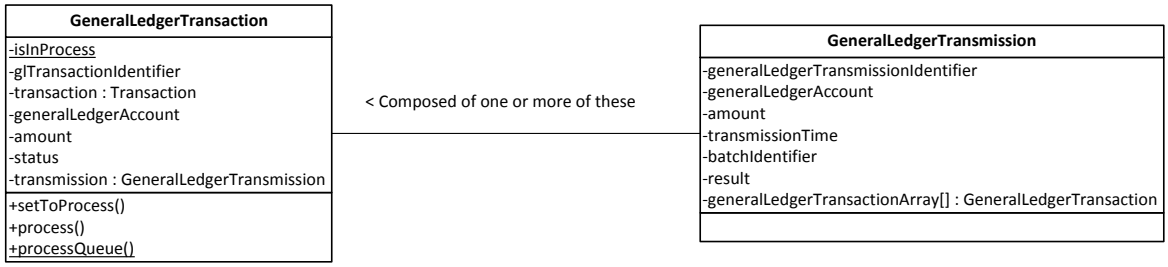
Attributes and Initial State

statusId	Autonumber set during instantiation.
Name	Set in constructor.
Description	Set in constructor.

Methods

editAccountStatusType (newName, newInformation)
Validate information, and then assign values within the object.

Class GeneralLedgerTransaction and GeneralLedgerTransmission



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.
transaction	Pointer to the transaction that generated this entry. Passed by the constructor.
generalLedgerAccount	Passed by the constructor.
amount	Passed by the constructor.
status	Defaults to Q (queue). Other values as the transaction processes are P (set to process) and C (completed processing).
transmission	Null. Pointer to the GeneralLedgerTransmission. This value will be null until status is C.

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.