OLE Batch/Scheduled Jobs Technical Documentation

Last Published September 16, 2015

Purpose

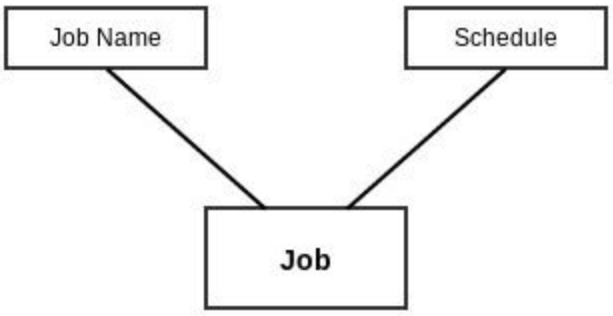
Batch/Schedule Jobs eDocs allows users to modify existing jobs' schedules. The schedule, defined as a cron expression, triggers the job. Only old jobs can be edited and new jobs cannot be created from the user interface. The jobs also include jobs inherited from Kuali Financial System (KFS) which are responsible mainly for the financial transaction related processes.

Dependencies (db tables)

Tables used in Batch Process creation

ole_dlvr_batch_job_t	Deliver Batch Jobs

Logical Data Model (Class Structure)



A Batch/Scheduled Jobs eDoc allows only to edit an existing job. One cannot create a new job from scratch through the user interface. The job comprises of the Job Name and the Schedule. The schedule is expressed in the form of a cron expression. The Cron expression comprises of 7 characters each representing seconds, minutes, hours, day of month, month, day of week and year, in that order. It takes in various different special characters such as * (all values), ? (no specific value), - (used to specify ranges), , (used to specify additional values), / (used to specify increments), L (last), W (weekday), # (used to specify the nth day of the month). For more information, please refer to this page.

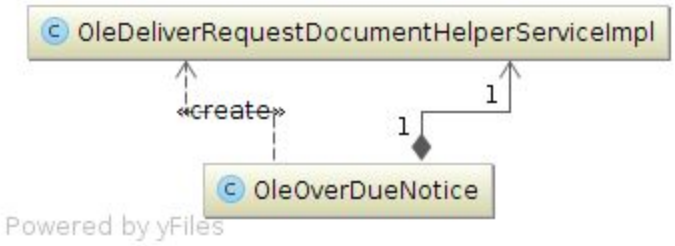
Physical Data Model (Database Schema)

The various data involved in the Batch/Scheduled Jobs screen are stored in the *ole_dlvr_batch_job_t* table. The *job_id* is the primary key. The job name goes into the *job_trg_nm* and the active indicator records the status of the job in *row_act_ind.* The cron expression which contains the schedule defined is recorded in the *job_cron_exprsn* column. Like all tables it has the *obj_id* and *ver_nbr* columns to take advantage of KRAD features. This is a standalone table and neither references from nor references to other OLE tables.

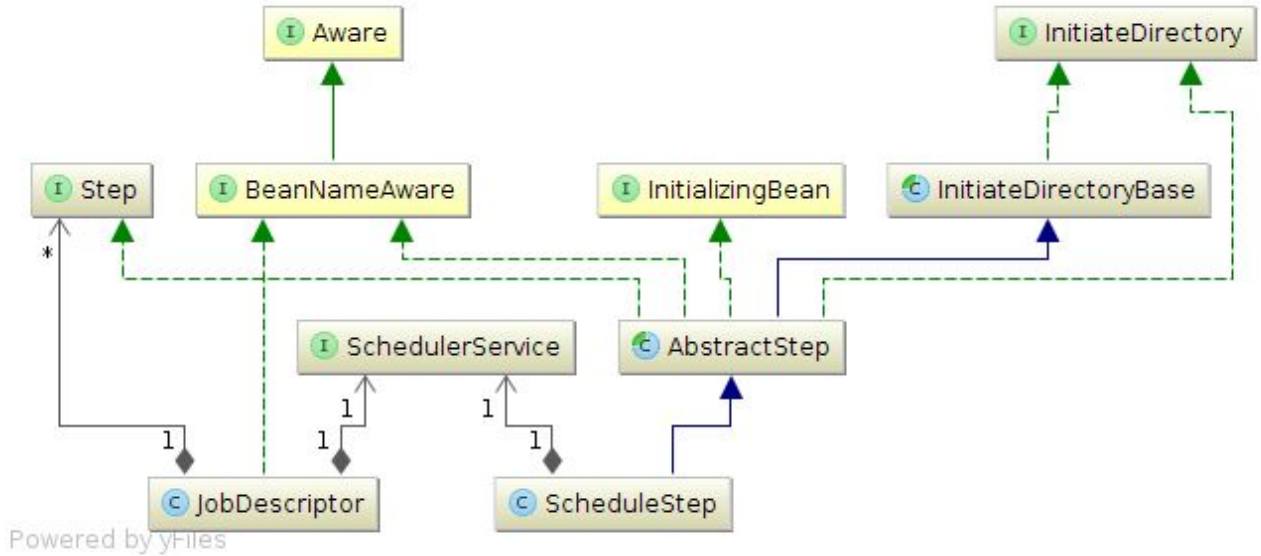
Service Interface Design (Java)

In OLE, job scheduling is done using Quartz Scheduler. The *MethodInvokingJobDetailFactoryBean* of Quartz Scheduler is used along with the Spring Framework and is mentioned in the

KRMSLocalSpringBeans.xml. The targetObject mentioned is the *OleOverDueNotice* class which calls respective methods in the *OleDeliverRequestDocumentHelperServiceImpl* class depending on the jobs.



In the case of jobs inherited from the Kuali Financial System (KFS), the *SchedulerService* class reads the job interval from a System Parameter and runs the job under the Scheduled types at the respective time interval.

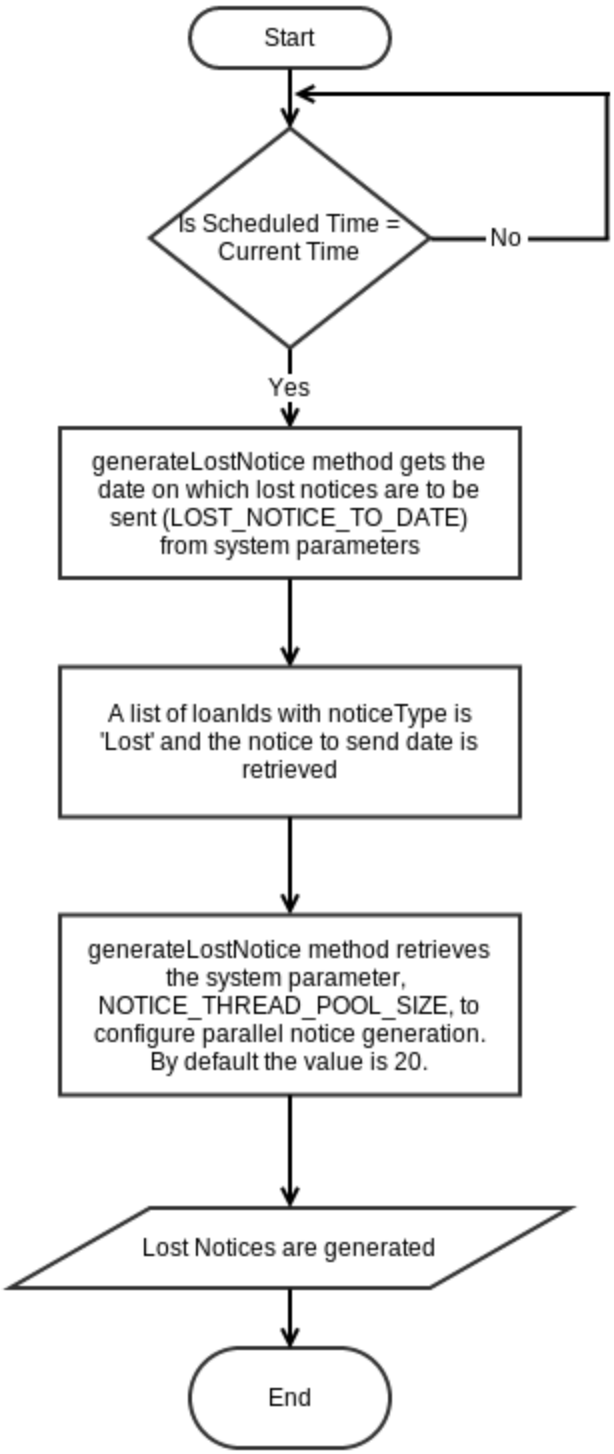


For more information, Javadocs can be found here.

Last Published September 16, 2015

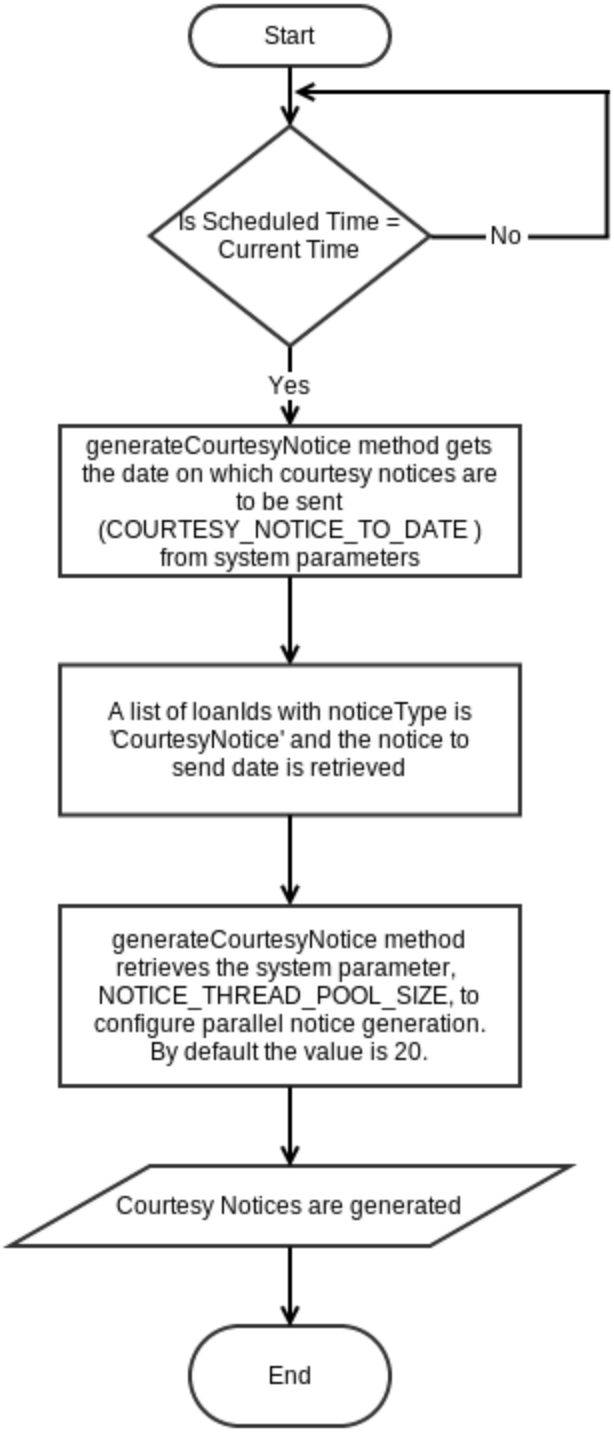
Flowchart

Generate Lost Notices Job



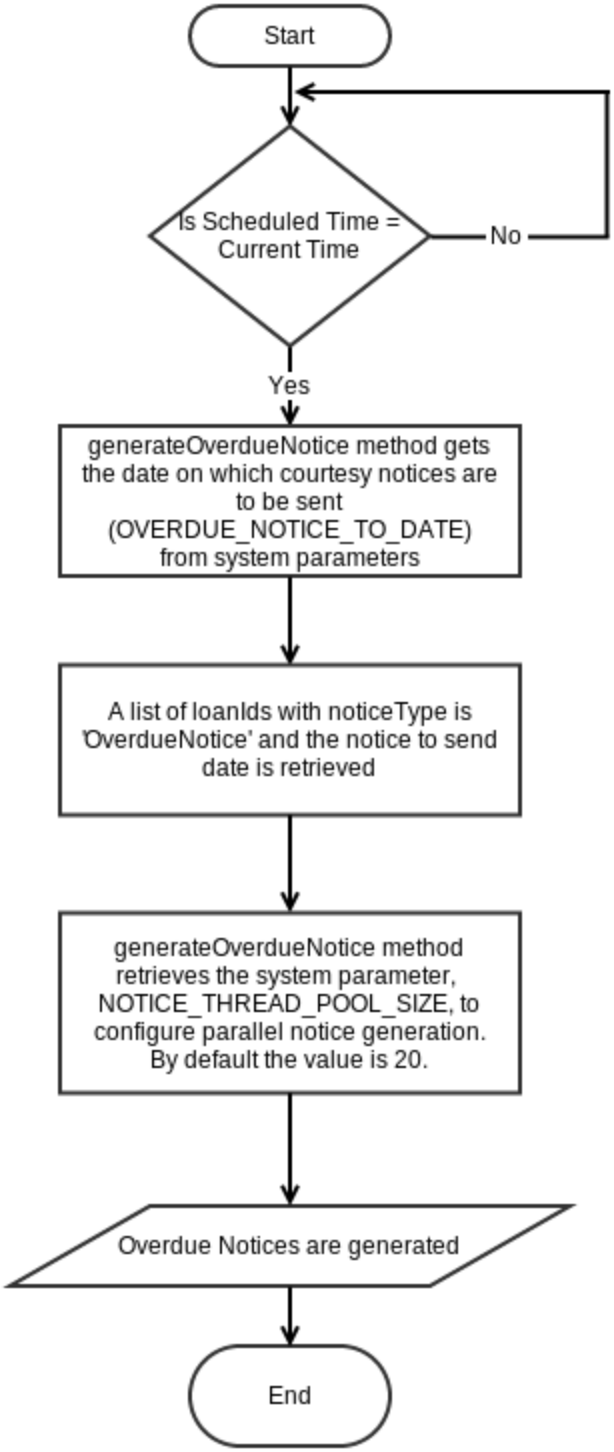
Last Published September 16, 2015

Generate Courtesy Notices Job



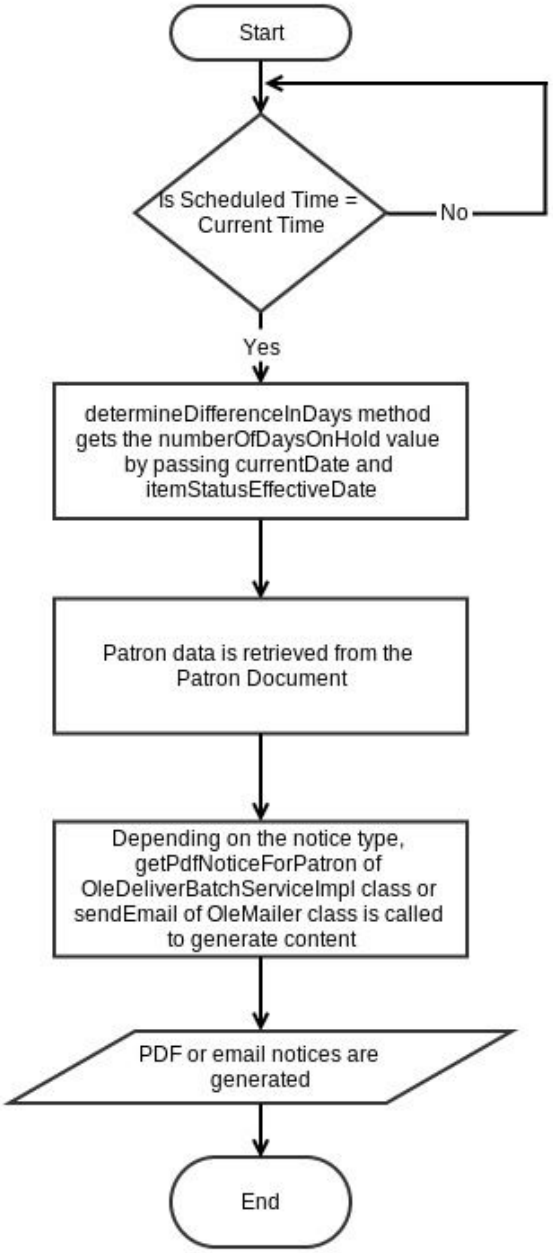
Last Published September 16, 2015

Generate Overdue Notices Job

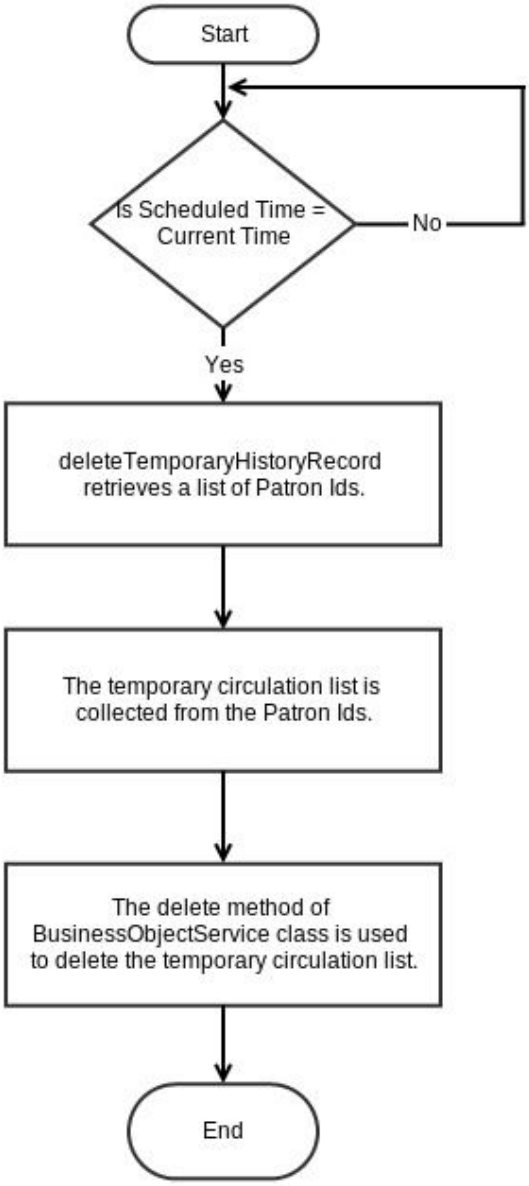


Last Published September 16, 2015

Generate Hold Courtesy Notices Job

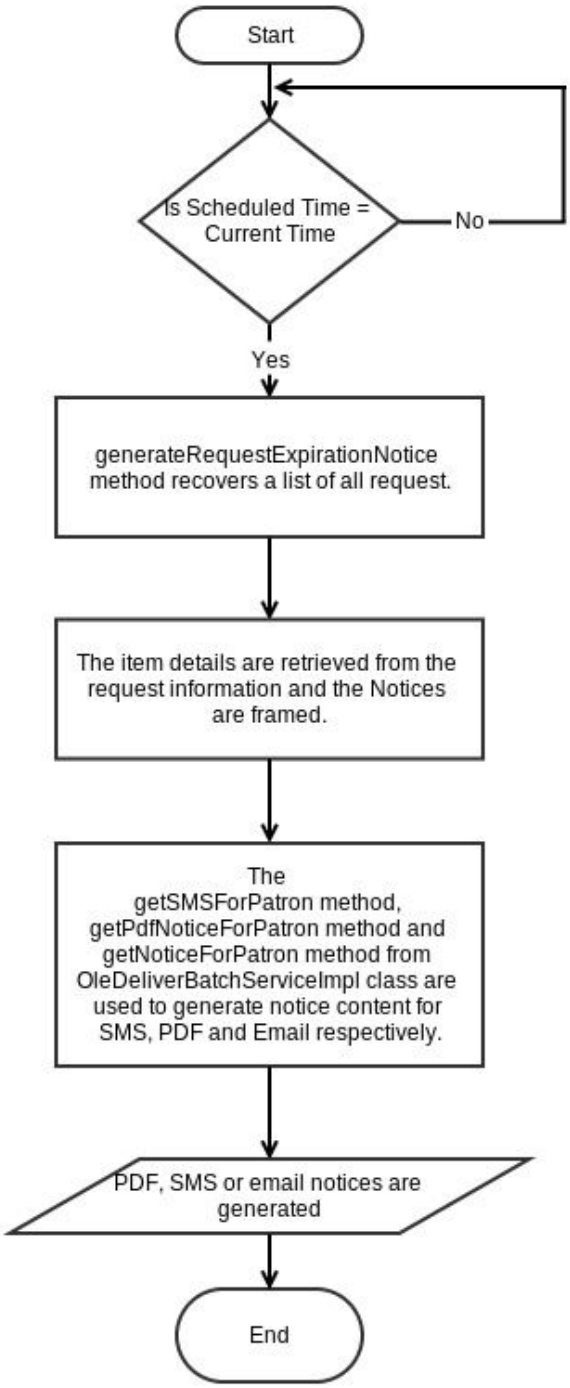


Delete Temporary History Records Job

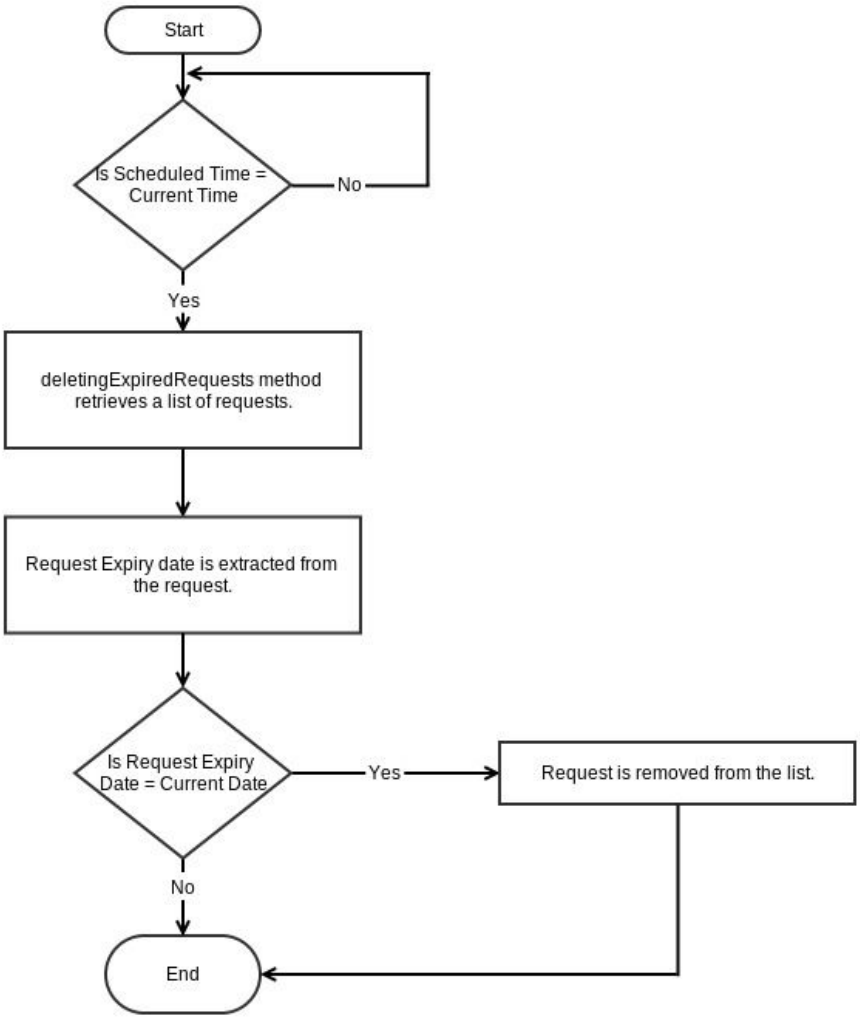


Last Published September 16, 2015

Generate Request Expiry Notice Job

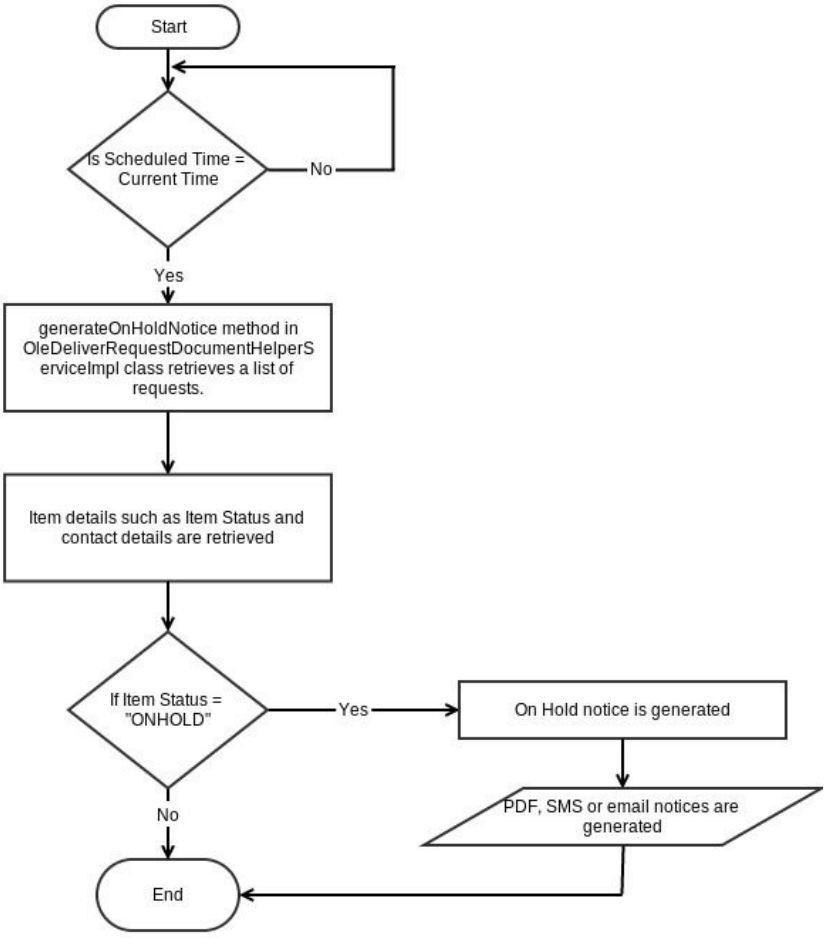


Delete Expired Requests Job



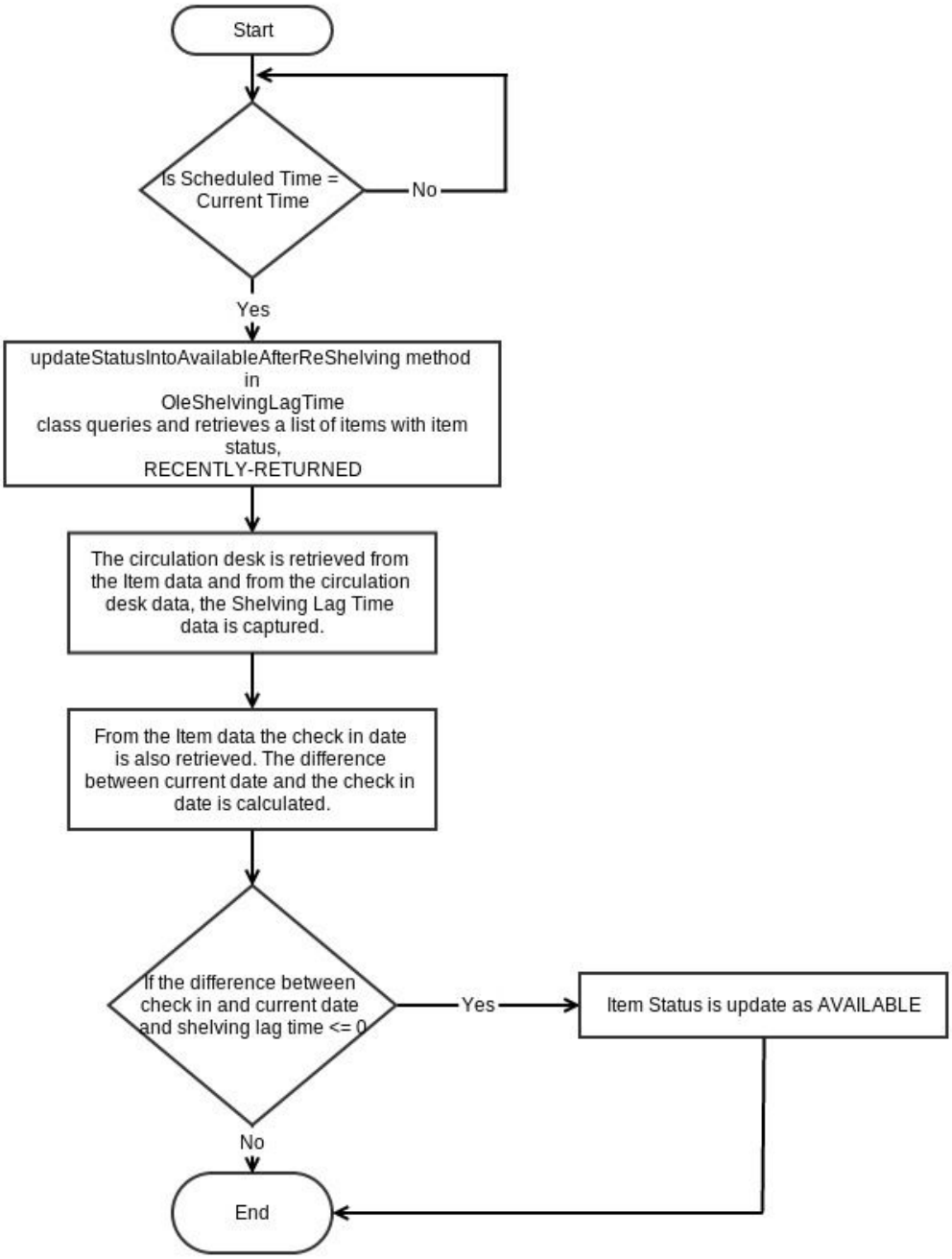
Last Published September 16, 2015

Generate On Hold Notice Job



Last Published September 16, 2015

Update Status to Available After Re-Shelving Job



Last Published September 16, 2015

Pseudo Code

The UI allows users to edit existing jobs' schedule which is in the form of a cron expression. More detailed information on how to construct cron expression can be found in the user documentation here.

Generate Lost Notices Job

The generateLostNoticesJob will generate Replacement Notice, Replacement Bill and updates the Item status to Lost.

- The job starts when the scheduled time configured as a cron expression is equal to the current time.
- The *generateLostNotice* method retrieves the date on which lost notices are to be sent from the system parameter, LOST_NOTICE_TO_DATE.
- A list of LoanIds are retrieved by passing the date and the notice type (Lost) to *getLoanIdsForOverudeNotices* (sic) method in the LoanWithNoticesDAO class.
- Loan documents are retrieved from the loanIds by passing it as parameter to the *getLaonDocumentsFromLaondId* (sic) method. The item information is retrieved by passing the loan documents to the *getLoanDocumentWithItemInfo* method.
- To boost performance, parallel execution of notice generation is employed. The system parameter, NOTICE_THREAD_POOL_SIZE is provided. If this is not specified then OLE considers 20 threads by default.
- The LostNoticesExecutor class handles the generation of Lost notices from the loan documents.

Generate Courtesy Notices Job

The generateCourtesyNoticesJob will generate and send out Courtesy Notices.

- The job starts when the scheduled time configured as a cron expression is equal to the current time.
- The *generateCourtesyNotice* method retrieves the date on which lost notices are to be sent from the system parameter, COURTESY_NOTICE_TO_DATE.
- A list of LoanIds are retrieved by passing the date and the notice type (CourtesyNotice) to *getLoanIdsForOverudeNotices* (sic) method in the LoanWithNoticesDAO class.
- Loan documents are retrieved from the loanIds by passing it as parameter to the *getLaonDocumentsFromLaondId* (sic) method. The item information is retrieved by passing the loan documents to the *getLoanDocumentWithItemInfo* method.
- To boost performance, parallel execution of notice generation is employed. The system parameter, NOTICE_THREAD_POOL_SIZE is provided. If this is not specified then OLE considers 20 threads by default.
- The CourtesyNoticesExecutor class handles the generation of Courtesy notices from the loan documents.

Generate Overdue Notices Job

The generateOverdueNoticesJob will generate and send out Overdue Notices.

- The job starts when the scheduled time configured as a cron expression is equal to the

current time.
- The *generateOverdueNotice* method retrieves the date on which lost notices are to be sent from the system parameter, OVERDUE_NOTICE_TO_DATE.
- A list of LoanIds are retrieved by passing the date and the notice type (OverdueNotice) to *getLoanIdsForOverudeNotices* (sic) method in the LoanWithNoticesDAO class.
- Loan documents are retrieved from the loanIds by passing it as parameter to the *getLaonDocumentsFromLaondId* (sic) method. The item information is retrieved by passing the loan documents to the *getLoanDocumentWithItemInfo* method.
- To boost performance, parallel execution of notice generation is employed. The system parameter, NOTICE_THREAD_POOL_SIZE is provided. If this is not specified then OLE considers 20 threads by default.
- The OverdueNoticesExecutor class handles the generation of Overdue notices from the loan documents.

Generate Hold Courtesy Notices Job

The generateHoldCourtesyNoticeJob will send the hold courtesy notice to the patron. This is sent when the hold period expires and the item is returned to the shelf or loaned to someone else.

- The job starts when the scheduled time configured as a cron expression is equal to the current time.
- The *determineDifferenceInDays* method gets the number of days on hold value by passing current date and Item status effective date.
- The maximum number of days on hold is retrieved from the circulation desk. When the number of days on hold value is greater than the maximum number of days on hold, the notice is generated.
- The Patron data is retrieved from the Patron document.
- The *getPdfNoticeForPatron* method of OleDeliverBatchServiceImpl class and *sentEmail* method of OleMailer class is used to generate content for PDF and Email, respectively.

Delete Temporary History Records Job

The deleteTemporaryHistoryRecordJob will delete the temporary circulation history records for the Patron.

- The job starts when the scheduled time configured as a cron expression is equal to the current time.
- The *deleteTemporaryHistoryRecord* method of OleDeliverRequestDocumentHelperServiceImpl class retrieves a list of Patron Ids.
- The temporary circulation list is collected from the Patron Ids. This is added into an array list.
- This list is passed to the *delete* method of the BusinessObjectService class which deletes the temporary circulation list.

Generate Request Expiry Notice Job

The generateRequestExpirationNoticeJob will send an ExpiredRequestNotice to the Patron when the request raised by the patron expires.

- The job starts when the scheduled time configured as a cron expression is equal to the current time.
- The *generateRequestExpirationNotice* method in

OleDeliverRequestDocumentHelperServiceImpl class recovers a list of all requests.
- The item details are retrieved from the request information and the notices are framed.
- The *getSMSForPatron, getPdfNoticeForPatron* and *getNoticeForPatron* methods from OleDeliverBatchServiceImpl class are used to generate notice content for SMS, PDF and email respectively.

Delete Expired Requests Job

The deletingExpiredRequestsJob will delete the expired request.

- The job starts when the scheduled time configured as a cron expression is equal to the current time.
- The *deletingExpiredRequests* method in OleDeliverRequestDocumentHelperServiceImpl class retrieves the list of requests.
- The request expiry date is taken from the request data and compared with the current date.
- If the date matches then the request is removed from the list.

Generate On Hold Notice Job

The generateOnHoldNoticeJob will send OnHoldNotice to the patron who had raised the request on an item when the item had been returned by the borrower and was available for pick up for the requestor until expiration date.

- The job starts when the scheduled time configured as a cron expression is equal to the current time.
- The *generateOnHoldNotice* method in OleDeliverRequestDocumentHelperServiceImpl class retrieves a list of all requests.
- Item details such as Item status and Contact details are retrieved from the request data.
- If the Item Status is ONHOLD then the On Hold Notice is generated and sent by means of PDF or SMS or email.

Update Status to Available After Re-Shelving Job

The updateStatusIntoAvailableAfterReshelvingJob will update the item status from ‘Recently-Returned’ to ‘Available’ after the item had been checked in and the shelving lag time for that circulation desk has been exhausted.

- The job starts when the scheduled time configured as a cron expression is equal to the current time.
- The *updateStatusIntoAvailableAfterReShelving* method in OleShelvingLagTime class queries and retrieves a list of items with item status as RECENTLY-RETURNED.
- The Circulation Desk details are retrieved from the item data.
- The Shelving Lag Time data is captured from the Circulation Desk details.
- The check in date of the item is also retrieved from the item data. The difference between current date and check in date is calculated.
- If the difference between this data and Shelving lag time is less than zero then the item status is update as AVAILABLE.

Jobs inherited from KFS

Receiving Payment Request Job

Purpose

This job reviews all Payment Request eDocs in "Awaiting Receiving" status for Line Item Receiving documents that have occurred for the same purchase order line(s) that are on the payment request. If found and the receiving quantity is greater than or equal to the payment request quantity then the pending action to the operator is approved. This then gets routed to the fiscal officer. This job would prove to be useful when Payment Request eDocs are not made to go to "Final" Status directly.

Psuedo Code:

- ➔ Payment Request eDocs with document status as "Awaiting Receiving" is filtered from the list of all Payment Request Documents using the *filterPaymentRequestByAppDocStatus* method.
- ➔ The Item details are extracted from the Payment Request Document in *isReceivingRequirementMet* method.
- ➔ The quantity of invoiced items and received items are got from the Item and compared in *isReceivingRequirementMet* method.
- ➔ If the number of received item is equal to or greater than the number of invoiced item then the document is approved.

Files Involved

- Bean definition in spring-purap.xml
- ReceivingPaymentRequestStep.Java
- PaymentRequestService.Java

Populate Prior Year Data Job

Purpose

This job populates Prior Year table with current year activity to be used in year end closing. This job is part of the Fiscal Year Rollover.

Pseudo Code

- ➔ The existing data in the Prior year Account table and the Organization table are purged in *purgePriorYearAccounts* and *purgePriorYearOrganizations* method of PriorYearAccountDao and PriorYearOrganizationDao class, respectively.
- ➔ The current Account and Organization data are moved to the prior year Account and Organization table using the methods, *copyCurrentAccountsToPriorYearTable* and *copyCurrentOrganizationsToPriorYearTable*, respectively.

In case of Accounts an additional step of purging Prior year ICR (Indirect Cost Recovery) Account and copying data from current ICR account to the Prior year ICR Account table. This probably can be removed in OLE given the absence of Contracts and Grants Module in OLE.

Files Involved

- Bean definition in spring-coa.xml
- UpdatePriorYearDataStep.Java

- PriorYearAccountService.Java
- PriorYearOrganizationService.Java
- PriorYearAccountDao.Java

Purge Reports and Staging Job

Purpose

This job deletes all files in the temporary directory that are over a day old. It helps in cleaning up old reports to make space for new ones.

Pseudo Code

➔ The *purgeFiles* method in FilePurgeService class checks for the existence of the directory.
➔ The files with custom ages is first purged. The method *purgeCustomAgeFiles* is used.
➔ Following this files without custom ages is also purged. The method *purgeDefaultFiles* is used.

Files Involved

- Bean definition in spring-sys.xml
- FilePurgeStep.Java
- FilePurgeService.Java
- FileUtil.Java
- FilePurgeDirectoryWalker.Java

Scrubber Job

Purpose

The Scrubber looks for a DONE file and performs the Scrubbing. It involves validation of account numbers for expired accounts, valid accounts and closed accounts and validation of Object Codes for valid object codes and active/inactive object codes. Transactions with exceptions are demerged as a group based on document number into a separate file.

Pseudo Code

➔ The first step the scrubber does is to create backup groups. The *createBackupGroup* method in OriginEntryGroupService class is used to create the backup groups.
➔ The next step is the PreScrubber step where Account related data such as Account record for the Account number, Presence of multiple Chart codes for an Account number, are validated. This happens in the *preprocessOriginEntries* method of PreScrubberService class.
➔ The next step is the Scrubber Sort step. The *compare* method in ScrubberSortComparator class takes care of this.
➔ The next step is the Scrubber step. This is the nightly process to scrub all incoming General Ledger transactions before posting to General Ledger tables. The *scrubEntries* method in the ScrubberProcessImpl class file scrubs all entries and puts valid entries in a scrubber valid group. The errors and transactions with expired accounts are put in scrubber error group and scrubber expired account group, respectively.
➔ The next step is the Demerger Sort step. The *compare* method in DemergerSortComparator class takes care of this which is similar to the Scrubber Sort step.
➔ This is followed by the Demerger step. This reads the error group documents then moves the original entries from the valid group to error group. The *performDemerger* method in

ScrubberProcess class takes care of this.

➔ Finally the *retrieveFilesToAggregate* method of ReportAggregationStep class aggregates all files and the *aggregateReports* method of ReportAggregatorService class writes them into the file.

Files Involved

- Bean definition in spring-gl.xml
- CreateBackupGroupStep.Java
- OriginEntryGroupService.Java
- PreScrubberService.Java
- ScrubberSortComparator.Java
- ScrubberProcess.Java
- DemergerSortComparator.Java
- ReportAggregationStep.Java

Dependencies

- Enterprise Feed Job
- Collector Job
- Nightly Out Job
- Pdp Extract Gl Transactions Step Job

Auto Disapprove Job

Purpose

This Job collects all the documents with document status in “Enroute” and cancels them on a specific date mentioned as a parameter (YEAR_END_AUTO_DISAPPROVE_DOCUMENT_RUN_DATE) in System Parameters. This job is normally run at year end.

NOTE: As of OLE v3.0, this job has been unscheduled as it was expected to run only during year end and not nightly. For more information, please refer to JIRA, OLE-7749.

Pseudo Code

➔ The Job first checks if the current date is the date mentioned in the System Parameter and proceeds only if the date matches in *processAutoDisapproveDocuments* method of AutoDisapproveDocumentsService class.

➔ It records the System User Id and also captures the annotation from System Parameter (YEAR_END_AUTO_DISAPPROVE_ANNOTATION) to set as note.

➔ It also checks if the document type is allowed to be auto disapproved from System Parameter.

➔ Finally it sets the user id, the note and uses the *superUserDisapproveDocumentWithoutSaving* method of DocumentService class to disapprove the document.

Files Involved

- Bean definition in spring-sys.xml
- AutoDisapproveDocumentsStep.Java
- AutoDisapproveDocumentsService.Java
- DocumentService.Java

Auto Close Purchase Order Job

Purpose

This job takes a list of all open purchase orders and checks for encumbrances, if they are fully disencumbered then the Purchase order is closed with relevant message.

Pseudo Code

- ➔ A list of all purchase order is taken and the document status is checked by the method *autoCloseFullyDisencumberedOrders* method in PurchaseOrderService class.
- ➔ Encumbrances are checked and if they are disencumbered then the purchase order is closed with relevant message.
- ➔ For potential documents, a change document is created and routing is set.

Files Involved

- Bean definition in spring-purap.xml
- AutoClosePurchaseOrdersStep.Java
- PurchaseOrderService.Java

PDP Nightly Load Payments Job

Purpose

The job processes payment files dropped into specific folders for processing as per nightly batch processing.

Pseudo Code

- ➔ A list of done payment files is returned based on the directory configured in *spring-pdp.xml.*
- ➔ The file so extracted is processed in the *processPaymentFile* method of PaymentFileService class.
- ➔ The *parsePaymentFile* method in PaymentFileService class validates the XML against the schema and parses.
- ➔ The *loadPayments* method in PaymentFileService class now loads the data from the XML and sends warnings as email.
- ➔ The *createOutputFile* method finally creates an output file with the status and other messages and descriptions.

Files Involved

- Bean definition in spring-pdp.xml
- LoadPaymentsStep.Java
- PaymentsFileService.Java
- BatchInputFileService.Java

Dependencies

- Disbursement Voucher Pre Disbursement Processor Extract Job
- Purchasing Pre Disbursement Extract Job

Auto Approve Payment Requests Job

Purpose

This job evaluates payment requests that are awaiting fiscal officer approval and auto approves them to FINAL status. Auto approval will not take place if any of the following 4 criteria are true:
1. Check if vendor is foreign.
2. Check to make sure the payment request isn't scheduled to stop in tax review.
3. Check for positive approval required indicator.
4. If it uses the payment request auto approve exclusion.

Pseudo Code

- ➔ Initially, a list of all payment request documents is made in *autoApprovePaymentRequests* method of PaymentRequestService class.
- ➔ The list is filtered to include only those documents with statuses - Awaiting Sub-Account Manager Approval, Awaiting Budget Approval, Awaiting Fiscal Officer Approval, Awaiting Chart Approval and Awaiting Payment Method Approval.
- ➔ The System Parameter, DEFAULT_POS_APRVL_LMT, has the value under which auto approval of payment request is allowed. This value is considered for auto approval.
- ➔ Finally the document is approved provided it doesn't match the above mentioned conditions.

Files Involved

- Bean definition in spring-purap.xml
- AutoApprovePaymentRequestsStep.Java
- PaymentRequestService.Java

PDP Load Payments Job

The PDP Load Payments Job serves the same purpose, follows the same steps and involves the same files as that of the PDP Nightly Load Payments Job.

Approve Line Item Receiving Job

Purpose

This job reviews the Line Item Receiving document that when submitted are in "Awaiting PO Open" status typically because an unordered item was received that is not on the purchase order and the purchase order is not available for amendment at the time of Line Item Receiving document submission. This job then reviews the purchase order to see if it is in the "Open" status. It then approves the the Line Item Receiving document.

Pseudo Code

- ➔ Initially, a list of all line item receiving documents with "Awaiting Purchase Order Open Status" is collected in *approveReceivingDocsForPOAmendment* method in ReceivingService class.
- ➔ Their respective Purchase Orders are retrieved and checked if the purchase order is open for processing in the *approveReceivingDoc* method of ReceivingService class.
- ➔ If purchase order is open for processing then the receiving document is approved with the "Approved by Batch Job" message.

Files Involved

- Bean definition in spring-purap.xml
- ApproveLineItemReceivingStep.Java
- ReceivingService.Java

Auto Close Recurring Orders Job

Purpose

This job looks at the end date on a recurring order (that is, a purchase order with a recurring payment type not equal to null). If this date is less than or equal to the date defined in the AUTO_CLOSE_RECURRING_PO_DATE parameter, it closes the purchase order and disencumbers any outstanding amounts.

Pseudo Code

- ➔ The "AUTO_CLOSE_RECURRING_PO_TO_EMAIL_ADDRESSES" system parameter for sending email report on closed POs and "AUTO_CLOSE_RECURRING_PO_DATE" system parameter for specifying the date used by batch job to close recurring POs are read in *autoCloseRecurringOrders* method of PurchaseOrderService class.
- ➔ A check is done comparing the date set in the system parameters and the date arrived by subtracting three months from current date. If the system parameter date is found to be greater the process ends.
- ➔ A list of recurring purchase orders with excluded vendor choice codes is taken.
- ➔ This list is further filtered to include only those documents with open status.
- ➔ This list is iterated and each PO document's end date is compared with the date arrived by subtracting three months from current date. If the end date is less then the PO is closed, email sent and a potential change document is routed.

Files Involved

- Bean definition in spring-purap.xml
- AutoCloseRecurringOrdersStep.Java
- PurchaseOrderService.Java

Close Job

The Close job is part of the Contracts and Grants module and hence is destined to be removed.

Account Temporary Restricted Notify Job

Purpose

This job checks if the status date of accounts marked with temporarily restricted status matches the current date and notifies the fiscal officer to either change the status or change the status date.

Pseudo Code

- ➔ A list of all accounts with Account Restricted Status Code set to "T- Temporarily Restricted" is collected.
- ➔ The Account Restricted Status Date is retrieved and compared with the current date.
- ➔ If the status date equals or is greater than the current date then the Fiscal Officer information

is retrieved from the Account Document.

➔ A notification is sent to the Fiscal Officer to either change the status or change the status date to a future date.

Files Involved

- Bean definition in spring-select.xml
- OleAccountTemporaryRestrictedStep.Java

Pdp Daily Report Job

Purpose

The job generates a report showing a summary of the number and amounts (by customer) of the payments that should be picked up the next time a format process is run. The report returns information for payments with pay dates equal to or earlier than today.

Pseudo Code

➔ The job collects all data pertaining to daily reports by filtering payment details based on the Payment Group Status code, whether it is for immediate processing and the payment date.

➔ The Job then creates a PDF file using method *openPdfWriter* in DailyReportServiceImpl class.

➔ Design related changes are set and the data collected earlier is written in this file.

Files Involved

- Bean definition in spring-pdp.xml
- DailyReportStep.Java
- DailyReportService.Java
- PaymentDetailDao.Java

NOTE: The date format used in creating the report is hardcoded in the *runReport* method of DailyReportServiceImpl class file. This may have to be modified as per the needs of the institution.

Dependencies

- Pdp Nightly Load Payments Job

Purge Job

Purpose

The purge job, as the name suggests, gets rid of old records from the system. It removes session documents, temporary files, ole lookup results and pending attachments.

Pseudo Code

➔ The job first deletes all pending attachments that are older than a configured time. This configured time is calculated from the System Parameter “MAX_AGE” in the *execute* method of PurgePendingAttachmentsStep class.

➔ The next step in this job is deleting old lookup results. The “MULTIPLE_VALUE_RESULTS_EXPIRATION_SECONDS” system parameter holds the value from which the expiration date is calculated in the *execute* method of PurgeOldLookupResultsStep class.

➔ The next step in this job is deleting old files from the temporary folder mentioned in the

build.properties file. This is done in the *execute* method of PurgeTempFilesStep class.

➔ Finally the purge job deletes all session document. To calculate the expiration date, the job uses the system parameter “NUMBER_OF_DAYS_SINCE_LAST_UPDATE”. Based on the value entered in the system parameter the expiration date is calculated in the *execute* method of PurgeSessionDocumentsStep class.

➔ The *purgeAllSessionDocuments* method of SessionDocumentService class clears all the session documents based on the expiration date.

Files Involved

- Bean definition in spring-sys.xml
- PurgeSessionDocumentsStep.Java
- SessionDocumentService.Java
- SessionDocumentDao.Java

Pdp Send Ach Advice Notifications Job

Purpose

The job generates emails to ACH (Automated Clearing House) payees based on the values defined as part of the PDP customer associated with the ACH payment.

Pseudo Code

➔ A list of all automated clearing house payments needing advice notification is collected in the *sendAdviceNotifications* method of AchAdviceNotificationService class.

➔ The customer profile is retrieved from the payment information.

➔ The body of the email is built and sent in the *sendAchAdviceEmail* method of PdpEmailServiceImpl class.

➔ The payment information is updated with the date on which the advice has been sent and saved.

Files Involved

- Bean definition in spring-pdp.xml
- SendAchAdviceNotificationsStep.Java
- AchAdviceNotificationService.Java
- PdpEmailService.Java

Pdp Clear Pending Transactions Job

Purpose

This job clears the temporary table in which G/L entries are stored. It should be run after the PDP Extract GL Transactions Step Job and the GL Nightly Out job run.

Pseudo Code

➔ A list of all General Ledger Pending Transactions with process indicator as true is retrieved.

➔ The list is then deleted and cache cleared in *clearExtractedTransactions* method of PendingTransactionDao class.

Files Involved

- Bean definition in spring-pdp.xml

- ClearPendingTransactionsStep.Java
- PendingTransactionService.Java
- PendingTransactionDao.Java

Dependencies

- Pdp Extract Gl Transactions Step Job
- Poster Job

Enterprise Feed Job

Purpose

The job uses a data file and reconciliation file to reconcile origin entry data.

Pseudo Code

- ➔ The *feed* method of FileEnterpriseFeederService class prepares the directories in the *prepareDirectories* method.
- ➔ The *feedOnFile* method of FileEnterpriseFeederHelperService class takes in the done file, data file, recon file as parameters and processes them.
- ➔ The *reconcile* method of ReconciliationService class reconciles the data from the data file and reconciliation data file and records the errors.

Files Involved

- Bean definition in spring-gl.xml
- EnterpriseFeedStep.Java
- EnterpriseFeederService.Java
- FileEnterpriseFeederHelperService.Java
- ReconciliationService.Java

Clear Pending Entries Job

Purpose

The clear pending entries job, as the name suggests, clears the pending ledger entries. It purges the transaction entry data in GL_PENDING_ENTRY_T table.

Pseudo Code

- ➔ The *execute* method of ClearPendingStep class calls the *deleteCopiedPendingLedgerEntries* method of the NightlyOutService class.
- ➔ This in turn calls the *deleteByFinancialDocumentApprovedCode* method of GeneralLedgerPendingEntryService class.
- ➔ A query is created and used to delete entries.

Files Involved

- Bean definition in spring-gl.xml
- ClearPendingStep.Java
- NightlyOutService.Java
- GeneralLedgerPendingEntryService.Java
- GeneralLedgerPendingEntryDao.Java

Dependencies

- Poster Job

Process PdP Cancels And Paid Step

Purpose

The job sends information back to the Purchasing/Accounts Payable module to update payment requests and credit memos with disbursement information. It also updates the Pre-Disbursement Processor Status tab on the Disbursement Voucher document.

Pseudo Code

- ➔ The *execute* method of ProcessPdPCancelsAndPaidStep class calls the *processPdpCancelsAndPaids* method of ProcessPdpCancelPaidService class.
- ➔ This in turn calls two methods viz. *processPdpCancels* and *processPdpPaids* of ProcessPdpCancelPaidService class.
- ➔ The *processPdpCancels* method processes cancelled documents and the *processPdpPaids* method processes paid documents.

Files Involved

- Bean definition in spring-pdp.xml
- ProcessPdpCancelsAndPaidStep.Java
- ProcessPdpCancelPaidService.Java

Poster Balancing Job

Purpose

The job generates the automated balancing report and could be run as part of an institution's nightly batch cycle. The first time the automated balancing process is initiated it populates entry and balance history tables. During the first run, the process builds the history tables that future runs of the process will use. Each run of the report generates statistics indicating the fiscal years included in the balancing along with summarized counts for each table verified and any comparison failures detected.

Pseudo Code

- ➔ The *getCustomBatchExecutor* method of PosterBalancingStep class calls the *runBalancing* method of BalancingService class.
- ➔ In the BalancingServiceImpl class, the poster file cache is first cleared by invoking the method, *clearPosterFileCache*.
- ➔ After the clearing of cache, the *runBalancing* method of BalancingServiceBaseImpl class is invoked.

Files Involved

- Bean definition in spring-gl.xml
- PosterBalancingStep.Java
- BalancingService.Java
- BalancingServiceBaseImpl.Java

Dependencies

- Poster Job

Schedule Job

Purpose

The Schedule job is responsible for running all the other jobs under the Job Group, Scheduled.

Pseudo Code

- ➔ The *reinitializeScheduledJobs* method of the SchedulerService class updates the status and adds the job to the list.
- ➔ The *execute* method then checks for the status check interval. This is a system parameter (STATUS_CHECK_INTERVAL) which defines intervals between which Schedule Jobs should check and execute waiting jobs.
- ➔ The *execute* method also checks the cut off time which is also mentioned in the system parameter (CUTOFF_TIME).
- ➔ After the checks, the *processWaitingJobs* method of SchedulerService class processes all the waiting jobs.

Files Involved

- Bean definition in spring-sys.xml
- ScheduleStep.Java
- SchedulerService.Java

Pdp Extract General Ledger (GL) Transactions Step Job

Purpose

The Pdp Extract General Ledger Transactions Step Job generates a file with the appropriate General Ledger entries for any payments processed through PDP. This file is called gl_glentry_pdp and is sent to the Origin Entry directory to be processed the next time the GL Nightly Out job is run.

Pseudo Code

- ➔ The *getCustomBatchExecutor* method of ExtractGlTransactionsStep class calls on the *extractGlTransactions* method of ExtractTransactionsService class.
- ➔ The unextracted transactions is retrieved by getting all the GL transactions where the extract flag is null in the *getUnextractedTransactions* method of PendingTransactionDao class.
- ➔ These transactions are iterated and their origin entry is recorded.
- ➔ Finally, the details are written into a file and placed in the relevant folder for access.

Files Involved

- Bean definition in spring-pdp.xml
- ExtractGlTransactionsStep.Java
- ExtractTransactionsService.Java
- PendingTransactionService.Java
- PendingTransactionDao.Java

Sufficient Funds Account Update Job

Purpose

The Sufficient Funds Account Update Job, as the name suggests, rebuilds and updates all necessary sufficient fund balances of the accounts.

Pseudo Code

- ➔ The *getCustomBatchExecutor* method of SufficientFundsAccountUpdateStep class calls the *rebuildSufficientFunds* method of SufficientFundsAccountUpdateService class.
- ➔ The *rebuildSufficientFunds* method retrieves all those data having the account financial object type code as object and changes them to account.
- ➔ Following this it retrieves all those data having the account financial object type code as account and processes them in the *calculateSufficientFundsByAccount* method of SufficientFundsAccountUpdateService class.

Files Involved

- Bean definition in spring-gl.xml
- SufficientFundsAccountUpdateStep.Java
- SufficientFundsAccountUpdateService.Java

Dependencies

- Nightly Out Job

Disbursement Voucher Pre Disbursement Processor Extract Job

Purpose

The Disbursement Voucher Pre Disbursement Processor Extract Job extracts all payments from a disbursement voucher with a status of “Approved” and uploads them as a batch for processing.

Pseudo Code

- ➔ The *execute* method of DvToPdpExtractStep class calls the *extractPayments* method of DisbursementVoucherExtractService class.
- ➔ The method retrieves the maximum number of note lines from the System Parameter data, MAX_NOTE_LINES.
- ➔ Finally, the *extractPaymentsForCampus* method of DisbursementVoucherExtractService class extracts all the outstanding payments from all the disbursement vouchers in approved status for a given campus and adds these payments to a batch file that is uploaded for processing.

Files Involved

- Bean definition in spring-fp.xml
- DvToPdpExtractStep.Java
- DisbursementVoucherExtractService.Java

Collector Job

Purpose

The collector job controls the collector process. It retrieves the files that need processing, parses each file into collector batch object using the collector digester rules, validates the contents in the CollectorService, stores origin group, GL entries and Id billings for each batch, sends email to workgroup listed in the batch file header with process results and finally cleans up the .done files.

Pseudo Code

- ➔ The *getCustomBatchExecutor* method of CollectorStep class first calls the *performCollection* method of CollectorService class.
- ➔ The *performCollection* method retrieves all the input files with .done extension from the folder.
- ➔ The *loadCollectorFile* method loads the files retrieved from the folder.
- ➔ The files are renamed with .data extension.
- ➔ Finally the execution statistics are added to the collector run.
- ➔ The *sendEmails* method of CollectorReportService class sends out emails about the validation and demerger of collector run and the *generateCollectorRunReports* method generates the report about a given collector run.
- ➔ Finally the *finalizeCollector* method of CollectorService class removes all the .done files.

Files Involved

- Bean definition in spring-gl.xml
- CollectorStep.Java
- CollectorService.Java
- CollectorReportService.Java
- CollectorHelperService.Java

Nightly Out Job

Purpose

The Nightly Out Job runs the process that prepares general ledger pending entries to be fed to the scrubber.

Pseudo Code

- ➔ The *getCustomBatchExecutor* method of NightlyOutStep class calls the *copyApprovedPendingLedgerEntries* method of NightlyOutService class.
- ➔ The *copyApprovedPendingLedgerEntries* method retrieves a list of approved pending ledger entries.
- ➔ It thens iterates through the list and processes and adds them to the nightly ledger summary report.

Files Involved

- Bean definition in spring-gl.xml
- NightlyOutStep.Java
- NightlyOutService.Java

Dependencies

- Auto Close Purchase Orders Job
- Auto Close Recurring Orders Job
- Process Pdp Cancels And Paid Job
- Asset Depreciation Batch Job

Purchasing Pre Disbursement Extract Job

Purpose

The Purchasing Pre Disbursement Extract Job extracts all eligible and approved Payment Request and Credit Memos eDocs into the Pre Disbursement Processor for payment.

Pseudo Code

- ➔ The *execute* method of ExtractPdpImmediatesStep class calls the *extractImmediatePaymentsOnly* method of PdpExtractService class.
- ➔ The *extractImmediatePaymentsOnly* method locks the document and calls the *extractPayments* method.
- ➔ Here the chart codes are retrieved and all the payments are extracted and a mail sent.

Files Involved

- Bean definition in spring-purap.xml
- ExtractPdpImmediatesStep.Java
- PdpExtractService.Java

Dependencies

- Auto Approve Payment Requests Job

Pdp Load Federal Reserve Bank Data Job

Purpose

The Pdp Load Federal Reserve Bank Data Job populates the Automated Clearing House (ACH) Bank table with values from an input file as defined in the System Parameter.

Pseudo Code

- ➔ The *execute* method of DownLoadFileViaHttpsStep class retrieves the source file URL and the target file name from the System Parameters, FEDERAL_ACH_BANK_FILE_URL and ACH_BANK_INPUT_FILE, respectively.
- ➔ The method makes a socket connection, reads the lines of the file and saves it in a output file.
- ➔ The *execute* method in LoadFederalReserveBankDataStep class retrieves the file saved by the earlier method and loads and saves them in the table.

Files Involved

- Bean definition in spring-pdp.xml
- DownLoadFileViaHttpsStep.Java
- LoadFederalReserveBankDataStep.Java
- AchBankService.Java

Procurement Card Document Job

Purpose

The Procurement Card Document job will validate and load the kuali procurement card XML file into the transaction table.

Pseudo Code

- ➔ The *execute* method of ProcurementCardLoadStep class calls the *cleanTransactionsTable* method of ProcurementCardLoadTransactionsService class to remove all the procurement card transaction rows from the transaction load table.
- ➔ A list of input files with .done extension is retrieved and each file parsed in the *loadProcurementCardFile* method.
- ➔ The *loadTransactions* method loads all the parsed XML transactions into the temporary transaction table.
- ➔ The *createProcurementCardDocuments* method of ProcurementCardCreateDocumentService class retrieves the list of credit card transactions and traverses through this list creating Procurement Card Document for each card.
- ➔ Next, the *routeProcurementCardDocuments* method of ProcurementCardCreateDocumentService class looks for Procurement Card Documents with a status of I (inbox) and routes them.
- ➔ Finally, the *autoApproveProcurementCardDocuments* method of ProcurementCardCreateDocumentService class finds documents that have been in route status past the number of allowed days, from the System Parameter AUTO_APPROVE_NUMBER_OF_DAYS, and auto approves the documents.

Files Involved

- Bean definition in spring-fp.xml
- ProcurementCardLoadStep.Java
- ProcurementCardCreateDocumentsStep.Java
- ProcurementCardRouteDocumentsStep.Java
- ProcurementCardAutoApproveDocumentsStep.Java
- ProcurementCardLoadTransactionsService.Java
- ProcurementCardCreateDocumentService.Java

Electronic Invoice Extract Job

Purpose

The Electronic Invoice Extract Job examines and validates invoices uploaded electronically by vendors. It creates payment request eDocs for valid invoices and electronic invoice reject eDocs for invalid invoices.

Pseudo Code

- ➔ The *execute* method of ElectronicInvoiceStep class calls on the *loadElectronicInvoices* method of ElectronicInvoiceHelperService class.
- ➔ The *loadElectronicInvoices* method identifies the files, loads the data and passes it on to the *processElectronicInvoice* method where payment request document is created in *createPaymentRequest* method.

- ➔ If there are rejects then based on the System Parameter, FILE_MOVE_AFTER_LOAD_IND, the rejected files are moved to a different folder.
- ➔ Finally the *deleteDoneFile* method deletes the .done files.

Files Involved

- Bean definition in spring-purap.xml
- ElectronicInvoiceStep.Java
- ElectronicInvoiceHelperService.Java

Poster Job

Purpose

The poster job process applies the pending ledger entries including any OLE-generated entries, to the General Ledger.

Pseudo Code

- ➔ The *getCustomBatchExecutor* method of PosterEntriesStep class calls the *postMainEntries* method of PosterService class.
- ➔ The *postMainEntries* method posts the scrubbed general ledger entries to the general ledger tables.
- ➔ The *postReversalEntries* method runs the poster service on reversals. It posts the reversal general ledger entries to the general ledger tables.
- ➔ Following this the *generateIcrTransactions* method runs the process that creates indirect cost recover transactions which the poster can later post. It reads the expenditure table and uses the data to generate Indirect Cost Recovery Transactions.
- ➔ The *postIcrEntries* method runs the poster service on indirect cost recovery entries. It posts the ICR general ledger entries to the general ledger tables.
- ➔ The *execute* method of the FileRenameStep class changes filenames stamping them with date.
- ➔ Finally the reports are generated in the *getCustomBatchExecutor* method of PosterSummaryReportStep class and ReportAggregationStep class.

Files Involved

- Bean definition in spring-gl.xml
- PosterEntriesStep.Java
- PosterService.Java
- ReportAggregationStep.Java
- PosterSummaryReportStep.Java
- FileRenameStep.Java
- PostericrGenerationStep.Java
- IcrSortStep.Java
- PosterIndirectCostRecoveryEntriesStep.Java

Dependencies

- Scrubber Job
- Sufficient Funds Account Update Job

Pdp Extract Canceled Checks Job

Purpose

The Pdp Extract Canceled Checks Job creates a XML file containing information about disbursements that have been canceled in Pdp. This file is available in the directory staging/PDP/Payment Extract (accessible via the Batch Filelookup function on the Administration menu tab). This XML file is meant to be sent to the bank to inform them of canceled checks.

Pseudo Code

- ➔ The *execute* method of the ExtractCanceledChecksStep class calls the *extractCanceledChecks* method of ExtractPaymentService class.
- ➔ The *extractCanceledChecks* method uses the *getCanceledChecks* method of PaymentGroupHistoryDao class to retrieve the canceled checks data and writes them into a file.

Files Involved

- Bean definition in spring-pdp.xml
- ExtractCanceledChecksStep.Java
- ExtractPaymentService.Java
- PaymentGroupHistoryDao.Java

Fax Pending Document Job

Purpose

The Fax Pending Document Job, as the name suggests, faxes pending documents. Currently it sends only Purchase Order documents set to pending fax status inside workflow.

Pseudo Code

- ➔ The *faxPendingPurchaseOrders* method in FaxBatchDocumentsServiceImpl class retrieves a list of all purchase orders with status pending fax.
- ➔ The *faxPurchaseOrderPdf* method of FaxService class creates a pdf document with data from the PO business object.
- ➔ The *faxPDF* method is where the transmission happens. This is yet to be implemented and will vary with different institutions.

Files Involved

- Bean definition in spring-purap.xml
- FaxService.Java
- FaxBatchDocumentsService.Java
- PurchaseOrderService.Java

Payment Request Batch Job

Purpose

The Payment Request Batch Job retrieves Payment Request documents (OLE_PREQ) in ‘enroute’ status and refreshes them.

Pseudo Code

- ➔ The *execute* method in PaymentRequestBatchProcessor class retrieves a list of all Payment Request Documents (OLE_PREQ) which have their document status as 'Enroute'.
- ➔ The documentId is sent as a parameter to the *refreshDocument* method of DocumentRefreshQueue class of Rice KEW module.
- ➔ The *refreshDocument* method retrieves the active nodes for the documentId. It then suppresses notification, deletes the existing request graph and trigger regeneration of requests.

FIles Involved

- Bean definition in spring-sys.xml
- PaymentRequestBatchProcessor.Java

Action List Batch Job

Purpose

The Action List Batch Job, as the name suggests, is a batch job that is used to execute pending documents in action list.

Pseudo Code

- ➔ The *execute* method in ActionListBatchProcessor class first retrieves data from System Parameters - ACTN_LIST_BATCH_PRNPL_NM, ACTN_LIST_BATCH_RQST_CD and ACTN_LIST_BATCH_DOC_TYP_CD for principal Id, request code and document type, respectively.
- ➔ It then retrieves the action items for the principal id. The retrieved list is further filtered to include only document type of the one mentioned in the System Parameter. It is further filtered on the basis of request code.
- ➔ Depending on the request code the relevant method (*complete, clearFyi, acknowledge, approve*) in WorkflowDocumentActionsService class of Rice KEW module is called with principal id and document id as parameters and the request is performed.

Files Involved

- Bean definition in spring-sys.xml
- ActionListBatchProcessor.Java

The following jobs are Unscheduled Jobs that can be run when necessary.

Weekly Email Job

Purpose

The Weekly Email Job is a batch step implementation for sending weekly email reminders. The email reminders are triggered by the state of the user's action list. This job is not part of the Scheduled jobs and may not be extensively used by all institutions.

Pseudo Code

- ➔ The *execute* method in the WeeklyEmailStep class calls the *sendWeeklyReminder* method of the ActionListEmailServiceImpl class.

- ➔ The *sendWeeklyReminder* method first checks if the System Parameter, SEND_EMAIL_NOTIFICATION_IND, is Y. This parameter is used as a flag to determine whether or not to send email notifications.
- ➔ Next the *getUsersWithEmailSetting* method is called. Here a set of users who have set their email preference as 'weekly' is retrieved. This is done by accessing the user option service and using the *retrieveEmailPreferenceUserOptions* method.
- ➔ The Action list service is then used and a collection of action lists for the user is retrieved using the *getActionList* method.
- ➔ The *sendPeriodicReminder* method is then called where the *filterActionItemsToNotify* method is called. This method returns a filtered collection of action items which are filtered according to the user's preference. If the user had opted to receive primary or secondary delegation emails then they will be included.
- ➔ The filtered action items are then passed to the *buildWeeklyReminderBody* method which adds the body content of the email with the action list requiring attention.
- ➔ The content along with the user email address, subject are sent to the *sendEmail* method which sends the email.

Files Involved

- Bean definition in spring-sys.xml
- WeeklyEmailStep.Java
- ActionListEmailServiceImpl.Java

Vendor Exclude Job

Purpose

The Vendor Exclude Job, as the name suggests, helps in excluding debarred vendors. The job has two parts to it - loading the EPLS (Excluded Parties List System) file and matching the debarred vendors with the list of vendors. This job is not part of the Scheduled jobs and may not be extensively used by all institutions.

Pseudo Code

- ➔ The *execute* method of the LoadEplsFileStep class calls the *loadEplsFile* method of the VendorExcludeServiceImpl class.
- ➔ The *loadEplsFile* method calls the *listInputFileNamesWithDoneFile* method of the VendorExcludeServiceImpl class. The method considers only the latest file based on the last modified attribute.
- ➔ From the file, the debarred vendor list is prepared.
- ➔ The *execute* method of the VendorMatchStep class calls the *matchVendors* function of the VendorExcludeServiceImpl class. The method matches the Vendors and the data stored in the database.

Files Involved

- Bean definition in spring-vnd.xml
- LoadEplsFileStep.Java
- VendorMatchStep.Java
- VendorExcludeServiceImpl.Java

Pdp Inactivate Payee Ach Accounts Job

Purpose

The Pdp Inactivate Payee Ach Accounts Job is a batch step to inactivate the payee ACH (Automated Clearing House) accounts for which the payee are inactive. Payee active status is obtained from the associated Person or Vendor depending on the Payee ID type. It also prepares a report about the inactivated records. The job is not part of the Scheduled jobs and may not be extensively used by all institutions.

Pseudo Code

- ➔ The *getCustomBatchExecutor* method of the InactivatePayeeAchAccountsStep class calls the *inactivatePayeeAchAccounts* method of the InactivatePayeeAchAccountsServiceImpl class.
- ➔ The method retrieves a list of active ach accounts using the *getActiveAchAccounts* method of the AchServiceImpl class.
- ➔ The Payee Id number is retrieved from active ach accounts using the *getPayeeIdNumber* method and the Person object is retrieved by passing this number to the *getPersonByEmployeeId* method.
- ➔ If the object returned is a null, in case the person doesn't exist anymore or if the person is not active, the account is set to inactive.
- ➔ In case the payee is an entity, the id is passed to the *getEntityDefaultInfo* method uses the Id to retrieve the entity object.
- ➔ In case the payee is a vendor, the id is passed to the *getVendorDetail* method uses the Id to retrieve the Vendor object.
- ➔ Similar to person, if the vendor object is null or if the vendor is not active, the account is set to inactive.
- ➔ A report is recorded with the following details
    - ◆ Total active accounts before running the job
    - ◆ Total accounts inactivated for employees
    - ◆ Total accounts inactivated for entities
    - ◆ Total accounts inactivated for vendors
    - ◆ Accounts inactivation date

Files Involved

- Bean definition in spring-pdp.xml
- InactivatePayeeAchAccountsStep.Java
- InactivatePayeeAchAccountsServiceImpl.Java

CFDA Job

Purpose

The CFDA (Catalog of Federal Domestic Assistance) job parses data from the government web page listing the valid CFDA codes. The codes are compared with the CFDA table in OLE. Codes that are set to be managed automatically are restored with those on the web page. Codes that are set to be managed manually are left untouched. An email containing the summary of work done is sent to members of the CG_CFDA_BATCH_NOTIFY workgroup. The job is not part of the Scheduled jobs and may not be extensively used by all institutions.

Pseudo Code

- ➔ The *execute* method of the CfdaBatchStep class calls the *update* method of CfdaServiceImpl class. The *getGovCodes* method is then called to retrieve the Government list.
- ➔ The URL where the CFDA codes are available is specified in the system parameter, SOURCE_URL. The file name is a fusion of the last two digits of a calendar year (Eg: 99 if year is 1999) and the three digit number depicting the day of the year. The extension is a 'csv'. The method returns a map with the code and CFDA object which has the code and the title.
- ➔ The *getKfsCodes* method returns the CFDA list from the local OLE instance. The method returns statistics on how many records were not updated because they were manual, records deactivated because no longer on Government list, records not updated for historical purposes, etc.
- ➔ Email addresses to whom the reports are to be sent is retrieved from the System Parameter, RESULT_SUMMARY_TO_EMAIL_ADDRESSES.
- ➔ The subject line is configured in the cg-resources.properties file under *cfda.updateEmail.subject*.

Files Involved

- Bean definition in spring-coa.xml
- CfdaBatchStep.Java
- CfdaBatchServiceImpl.Java

Modules Lock/Unlock Job

Purpose

The Modules Lock Job, as the name suggests, helps in locking and unlocking modules for the batch. The namespace code and lock/unlock boolean are passed as properties. The job is not part of the Scheduled jobs and may not be extensively used by all institutions.

Pseudo Code

- ➔ The *execute* method of the LockModuleStep class calls the *lockModule* method of the LockModuleServiceImpl class.
- ➔ Depending on the lock or unlock job, the lockModule property is set to true or false.
- ➔ The *lockModule* method retrieves the System Parameter, OLTP_LOCKOUT_ACTIVE_IND.
- ➔ The OLTP_LOCKOUT_ACTIVE_IND parameter acts as an indicator to show whether the module is locked or unlocked for transactions. Different namespace code for the parameter helps in differentiating the Parameter for the different modules in OLE.
- ➔ Depending on the lockModule Property mentioned in the bean definition, the parameter is updated.

Files Involved

- Bean definition in spring-sys.xml
- LockModuleStep.Java
- LockModuleServiceImpl.Java

PO Relink Job

Purpose

The PO relink Job helps in creating multiple item copies and link them to the Purchase Order (PO) when there is a mismatch. The job is not part of the Scheduled jobs and may not be extensively used by all institutions.

Pseudo Code

- ➔ The *execute* method of OLECopyRecordProcessor class calls the *createCopyRecords* method of the same class.
- ➔ The *createCopyRecords* method calls the *getPurchaseOrderFDocNoList* method to retrieve the list of documents. It uses a SQL to pull records from the pur_po_t and ole_copy_t tables.
- ➔ The *createCopies* method is called where the document numbers are passed and the PO documents are retrieved. The copies are created based on the quantity information, Item UUID, Instance Id, Location, et al are set and linked to the PO.

Files Involved

- Bean definition in spring-sys.xml
- OLECopyRecordProcessor.Java

Clear Cache Job

Purpose

The Clear Cache Job, as the name suggests, is a batch step to clear the system cache. The job is not part of the Scheduled jobs and may not be extensively used by all institutions.

Pseudo Code

- ➔ The *execute* method of ClearCacheStep class calls the *clearSystemCaches* method of CacheServiceImpl class.
- ➔ The *clearSystemCaches* method retrieves a list of registered cache managers from the cache manager registry.
- ➔ The list is iterated and the cache name is retrieved. The cache object is pulled using the cache name and cleared.

Files Involved

- Bean definition in spring-sys.xml
- ClearCacheStep.Java
- CacheServiceImpl.Java

Service Interface Design (SOAP/REST)

No Web services are available currently.

User Interface Design

The Batch Job Lookup document uses KRAD's UIF (User Interface Framework). A very good guide on the framework can be found here. The Spring Beans XML used in the case of Batch Job is the *BatchJobStatus.xml* under the *datadictionary* folder.

Data Importing

It is currently not possible to import a Batch Job directly into OLE.

Data Exporting (if applicable)

OLE uses a RDBMS backend and hence any data can be exported using simple SQL queries.

Workflow

Batch Process document doesn't have any workflow defined by default. However, if needed it can be defined at *OleBatchDocType.xml* file.

System Parameters

Namespace Code	Parameter Name	Description
OLE-DLVR	COURTESY_NOTICE_INTER	The interval between courtesy notices.
OLE-SYS	YEAR_END_AUTO_DISAPPROVE_DOCUMENT_RUN_DATE	This specifies the date on which the auto disapproval step should run.
OLE-SYS	YEAR_END_AUTO_DISAPPROVE_ANNOTATION	This document has been automatically disapproved as part of year-end closing. If these are valid transactions they should be recreated using a year-end closing document.
OLE-PURAP	DEFAULT_POS_APRVL_LMT	The default amount above which positive payment request approval is required.
OLE-PURAP	AUTO_CLOSE_RECURRING_PO_DATE	The date used by batch job to automatically close recurring POs prior to this date. Parameter is defaulted to "mm/dd/yyyy" when job shouldn't close any orders. User should set the date when ready to close orders.

OLE-PURAP	AUTO_CLOSE_RECURRING_PO_TO_EMAIL_ADDRESSES	The Email address used by batch job for automatically closing POs. Email is sent to this address upon successful completion of the job to inform user of the number of POs closed.
OLE-PURAP	AUTO_CLOSE_PO_TO_DATE	The Date used by batch job to automatically close POs up to this date. Parameter is defaulted to "mm/dd/yyyy" when job will close any orders irrespective of PO Created Date. User should set the date when they want to close the PO's created up to this date.
OLE-PURAP	AUTO_CLOSE_PO_FROM_DATE	The Date used by batch job to automatically close POs starting from this date. Parameter is defaulted to "mm/dd/yyyy" when job will close any orders irrespective of PO Created Date. User should set the date when they want to close the PO's created from this date.
OLE-PURAP	AUTO_CLOSE_PO_THREAD_POOL_SIZE	The parameter is used to define the size of thread for auto close PO executor class
OLE-SYS	MAX_AGE	Pending attachments are attachments that do not yet have a permanent link with the associated Business Object (BO). These pending attachments are stored in the attachments.pending.directory (defined in the configuration service). If the BO is never persisted, then this attachment will become orphaned (i.e. not associated with any BO), but will remain in this directory. The PurgePendingAttachmentsStep batch step deletes these pending attachment files that are older than the value of this parameter. The unit of this value is seconds. Do not set this value too short, as this will cause problems attaching files to BO's.
KR-NS	MULTIPLE_VALUE_RESULTS_EXPIRATION_SECONDS	Lookup results may continue to be persisted in the DB long after they are needed. This parameter represents the maximum amount of time, in seconds, that the results will be allowed to persist in the DB before they are deleted from the DB.
OLE-SYS	NUMBER_OF_DAYS_SINCE_LAST_UPDATE	Determines the age of the session document records that the the step will operate on, e.g. if this param is set to 4, the rows with a last update timestamp

		older that 4 days prior to when the job is running will be deleted.
OLE-SYS	STATUS_CHECK_INTERVAL	Time in milliseconds that the scheduleStep should wait between iterations.
OLE-SYS	CUTOFF_TIME	Controls when the daily batch schedule should terminate. The scheduler service implementation compares the start time of the schedule job from quartz with this time on day after the schedule job started running.
OLE-PDP	MAX_NOTE_LINES	The maximum number of note/text lines allowable on a PDP payment. Typically this is the maximum number of lines that can be printed on a check stub.
OLE-PDP	FEDERAL_ACH_BANK_FILE_URL	The URL of the file to download from the Federal ACH Bank Directory website. This file is then loaded into the ACH bank table.
OLE-PDP	ACH_BANK_INPUT_FILE	The name of the file to load into the Federal ACH Bank Directory File. This file is used to validate bank routing numbers.
OLE-FP	AUTO_APPROVE_NUMBER_OF_DAYS	Number of days a Procurement Card document is allowed to remain in "enroute" status before being auto-approved by the system. Used in conjunction with parameter, AUTO_APPROVE_IND.
OLE-FP	AUTO_APPROVE_IND	Flag that determines auto approval of the transactions (Y/N) in the Procurement Card document (batch process) to be turned on or off.
OLE-PURAP	FILE_MOVE_AFTER_LOAD_IND	Value indicating whether the loaded eInvoice cxml file shall be moved to the accept/reject directory.
OLE-DLVR	NOTICE_THREAD_POOL_SIZE	This parameter provides the number of parallel execution to be applied on the notice generation process
OLE-SYS	ACTN_LIST_BATCH_PRNPL_NM	Principal ID on whose action list actions need to be taken on. ATTN: really is Principal ID and not Name as suggested by the name of this parameter.
OLE-SYS	ACTN_LIST_BATCH_RQST_CD	Request Code for the action that needs to be taken (K-Ack, A-Approve, F-FYI, C-Complete)

OLE-SYS	ACTN_LIST_BATCH_DOC_TYP_CD	Document Type for the action to be taken
OLE-DLVR	NOTICE_THREAD_POOL_SIZE	This parameter provides the number of parallel execution to be applied on the notice generation process.
OLE-DLVR	OVERDUE_NOTICE_TO_DATE	The overdue notices will be send to the patron who are having the overdue date falling on or before the date specified in this parameter while running the notice job
OLE-DLVR	COURTESY_NOTICE_TO_DATE	The courtesy notices will be send to the patron who are having the courtesy date falling on or before the date specified in this parameter while running the notice job
OLE-DLVR	LOST_NOTICE_TO_DATE	The replacement fee will be generated and item status is updated to lost for the items have the lost date falling on or before the date specified in this parameter
KR-WKFLW	SEND_EMAIL_NOTIFICATION_IND	This parameter is a flag to determine whether or not to send email notifications
OLE-COA	SOURCE_URL	The CFDA (Catalog of Federal Domestic Assistance) URL is specified here.
OLE-COA	RESULT_SUMMARY_TO_EMAIL_ADDRESSES	A list of email addresses who are the recipients of CFDA batch process results.
OLE-COA	OLTP_LOCKOUT_ACTIVE_IND	The parameter acts as an indicator to show if the Chart of Account Module is currently unavailable for Online Transaction Processing (OLTP). ‘Y’ means the module is currently locked and cannot be used. ‘N’ means the module is unlocked and available for use.
OLE-SYS	OLTP_LOCKOUT_ACTIVE_IND	The parameter acts as an indicator to show if the System Module is currently unavailable for Online Transaction Processing (OLTP). ‘Y’ means the module is currently locked and cannot be used. ‘N’ means the module is unlocked and available for use.
OLE-VND	OLTP_LOCKOUT_ACTIVE_IND	The parameter acts as an indicator to show if the Vendor Module is currently unavailable for Online Transaction Processing (OLTP). ‘Y’ means the module is currently locked and cannot be used. ‘N’ means the module is unlocked and available for use.

OLE-AR	OLTP_LOCKOUT_ACTIVE_IND	The parameter acts as an indicator to show if the Accounts Receivable Module is currently unavailable for Online Transaction Processing (OLTP). ‘Y’ means the module is currently locked and cannot be used. ‘N’ means the module is unlocked and available for use.
OLE-LD	OLTP_LOCKOUT_ACTIVE_IND	The parameter acts as an indicator to show if the LD Module is currently unavailable for Online Transaction Processing (OLTP). ‘Y’ means the module is currently locked and cannot be used. ‘N’ means the module is unlocked and available for use.
OLE-BC	OLTP_LOCKOUT_ACTIVE_IND	The parameter acts as an indicator to show if the Budget Construction Module is currently unavailable for Online Transaction Processing (OLTP). ‘Y’ means the module is currently locked and cannot be used. ‘N’ means the module is unlocked and available for use.
OLE-CAB	OLTP_LOCKOUT_ACTIVE_IND	The parameter acts as an indicator to show if the Capital Asset Builder Module is currently unavailable for Online Transaction Processing (OLTP). ‘Y’ means the module is currently locked and cannot be used. ‘N’ means the module is unlocked and available for use.
OLE-CAM	OLTP_LOCKOUT_ACTIVE_IND	The parameter acts as an indicator to show if the CAM Module is currently unavailable for Online Transaction Processing (OLTP). ‘Y’ means the module is currently locked and cannot be used. ‘N’ means the module is unlocked and available for use.
OLE-PURAP	OLTP_LOCKOUT_ACTIVE_IND	The parameter acts as an indicator to show if the Purchasing and Accounts Payable Module is currently unavailable for Online Transaction Processing (OLTP). ‘Y’ means the module is currently locked and cannot be used. ‘N’ means the module is unlocked and available for use.
OLE-FP	OLTP_LOCKOUT_ACTIVE_IND	The parameter acts as an indicator to show if the Financial Processing Module is currently unavailable for Online Transaction Processing (OLTP). ‘Y’ means the module is currently locked and cannot

		be used. ‘N’ means the module is unlocked and available for use.
OLE-EC	OLTP_LOCKOUT_ACTIVE_IND	The parameter acts as an indicator to show if the EC Module is currently unavailable for Online Transaction Processing (OLTP). ‘Y’ means the module is currently locked and cannot be used. ‘N’ means the module is unlocked and available for use.
OLE-PDP	OLTP_LOCKOUT_ACTIVE_IND	The parameter acts as an indicator to show if the Pre Disbursement Processor Module is currently unavailable for Online Transaction Processing (OLTP). ‘Y’ means the module is currently locked and cannot be used. ‘N’ means the module is unlocked and available for use.
OLE-CG	OLTP_LOCKOUT_ACTIVE_IND	The parameter acts as an indicator to show if the Contracts and Grants Module is currently unavailable for Online Transaction Processing (OLTP). ‘Y’ means the module is currently locked and cannot be used. ‘N’ means the module is unlocked and available for use.
OLE-ENDOW	OLTP_LOCKOUT_ACTIVE_IND	The parameter acts as an indicator to show if the Endowment Module is currently unavailable for Online Transaction Processing (OLTP). ‘Y’ means the module is currently locked and cannot be used. ‘N’ means the module is unlocked and available for use.
OLE-GL	OLTP_LOCKOUT_ACTIVE_IND	The parameter acts as an indicator to show if the General Ledger Module is currently unavailable for Online Transaction Processing (OLTP). ‘Y’ means the module is currently locked and cannot be used. ‘N’ means the module is unlocked and available for use.
OLE-SELECT	OLTP_LOCKOUT_ACTIVE_IND	The parameter acts as an indicator to show if the Select Module is currently unavailable for Online Transaction Processing (OLTP). ‘Y’ means the module is currently locked and cannot be used. ‘N’ means the module is unlocked and available for use.

Roles and Permissions

Permissions are linked to roles which are in turn linked to Users to give them access to screens and functions.

Permission ID	Permission Name
164	Modify Batch Job
OLE132	Modify Batch Job KFS*
OLE256	Look Up Records BatchJobStatus

Role ID	Role Name	Permissions
63	Technical Administrator	164
OLE45	Operations	OLE132
OLE82	Batch Job Modifier	OLE256