Cúram 8.2

Batch Process Reference Guide

Note

Before using this information and the product it supports, read the information in Notices on page 73

Edition

This edition applies to Cúram 8.2.

Contents

Chapter 1 Batch process reference

The following batch processes are available by default. Use this reference to find more information about specific batch processes.

1.1 Core batch processes

A list of the batch process that provide core functions in the application.

Unless specified otherwise, you can run batch processes in any order.

Batch Parameters and Processing Date

Several batch jobs in the application accept processingDate as a parameter.

Unless specified otherwise, the following conditions apply to the batch jobs that accept the parameter:

- The processingDate parameter is optional.
- You can use the processingDate parameter to specify the effective date for which the batch job is run, which is the date that is returned from the `getCurrentDate()` API. Otherwise, the `getCurrentDate()` API returns the current system date as the effective date.

Some batch programs might use the processingDate parameter in other specific ways that are described in the batch program documentation.

Cúram Processing

MeetingManagement

This batch process processes responses returned to the configured email account and updates the meeting attendee response data accordingly. The MeetingManagement batch process connects to a configured email account and checks for meeting responses. If responses are found, attendee response details are updated and the email is removed from the account inbox.

Processing steps

Meeting Management initiates the following processing steps:

1. Connects to the email account using the username, password and host set by the following the environment properties:
    1. ENV_MEETING_REQUEST_REPLY_USERNAME
    2. ENV_MEETING_REQUEST_REPLY_PASSWORD
    3. ENV_MEETING_REQUEST_REPLY_HOST
2. Opens the mail folder.

3. Processes all the meeting response messages and updates the appropriate meeting attendee response data. The message must contain a body part of type text/calendar, the calendar body part must comply with the RFC 2445 standard for calendar data exchange and contain a UID that matches the ID of a Cúram activity. An exception is thrown if an invalid response method is received or if a message processing exception is encountered.

Class and method

MeetingManagement.processMeetingResponses

Parameters

MeetingManagement has no input parameters.

Case Management

GenerateCommunications

This batch process generates communications that are part of case management, e.g., case approval communications. An environment variable can be set such that case management communications are not processed automatically but go into a pending status. This batch process generates all of these pending communications.

It is expected that this program be run nightly, but other than a longer run time and the potential for case management communications to go unprocessed until the program is run again, no problems will result from running this less frequently. Running this program more than once a day has no effect.

This batch process takes no parameters.

Batch Process Class and Method The class and method for this batch process is `curam.core.intf.GenerateCommunications.generateAllCommunications`.

ProductDeliveryFinalClosure

This batch process is provided to close cases in the pending closure state, when the closure grace period has expired.

It is expected that this program be run nightly, but other than a longer run time and the potential for cases to have exceeded the closure grace period before they are closed, no problems will result from running this less frequently. Running this program more than once a day has no effect.

This batch process takes no parameters.

Batch Process Class and Method The class and method for this batch process is `curam.core.intf.CloseCasesPendingClosure.closeCasesPendingClosure`.

EvaluateCertificationGracePeriod

This batch process is provided to set cases to pending closure when they have been out of certification for the certification grace period.

It is expected that this program be run nightly, but other than a longer run time and the potential for cases to have exceeded the certification grace period before they are set to pending closure, no problems will result from running this less frequently. Running this program more than once a day has no effect.

This batch process takes no parameters.

Batch Process Class and Method The class and method for this batch process is `curam.core.intf.EvaluateCertificationGracePeriod.run`.

DetermineProductDeliveryEligibility

This batch process is provided to process newly approved cases, and record decisions in respect of these cases. Any ineligible cases will be set to pending closure by this process, and cases that are eligible will be set to active. If an error occurs processing a case, it will be suspended and the case owner notified by email or via a task.

It is expected that this program be run nightly, but other than a longer run time and the potential for approved cases to be unprocessed until the program is run again, no problems will result from running this less frequently. Running this program more than once a day, will process all approved cases on the system at that point in time; no problems will result from running the program this way.

This batch process takes the following parameters:

- `Product ID` - unique identifier used to run this process for cases of a particular product

  Where no product ID is specified, all cases are processed.
- `Batch Process Instance ID` - unique identifier used to allow multiple instances of the same batch process to run at the same time effectively

  For example, this process can run for multiple products. Where no instance ID is specified, only one instance of the batch process can run.

Batch Process Class and Method The class and method for this batch process is `curam.core.intf.DetermineProductDeliveryEligibility.process`.

DetermineProductDeliveryEligibilityStream

This batch process is provided to support streaming for the `Determine Product Eligibility` process. This batch process can only be run in conjunction with the `Determine Product Eligibility` process. Streaming for batch processes is designed to allow for concurrent execution, possibly on different machines, of batch programs to ensure that the database is used to its full capacity.

This batch process takes the parameter, `Batch Process Instance ID` . The instance ID is a unique identifier used to allow multiple instances of the same batch process to run at the same

time effectively. Where no instance ID is specified, only one instance of the batch process can run.

Streaming Multiple Instances of Determine Product Delivery Eligibility Multiple instances of the `Determine Product Delivery Eligibility` batch process can be run concurrently, by giving each instance a different `Batch Process Instance ID` . To start a stream for a particular instance of the `Determine Product Delivery Eligibility` batch process, you must link the `Determine Product Delivery Eligibility Stream` batch process (or multiple stream batch processes) to the particular batch process instance using the `Batch Process Instance ID` parameter. For example, where the `Batch Process Instance ID` is "eligibility_determination_1" for an instance of the `Determine Product Delivery Eligibility` batch process, you must also set the `Batch Process Instance ID` parameter for the `Determine Product Delivery Eligibility Stream` batch process (or multiple stream batch processes) to be "eligibility_determination_1". Any number of `Determine Product Delivery Eligibility Stream` batch processes can be linked to the same instance of the `Determine Product Delivery Eligibility` batch process.

Batch Process Class and Method The class and method for this batch process is `curam.core.intf.DetermineProductDeliveryEligibilityStream.process`.

ReassessOutstandingCases

The `ReassessOutstandingCases` batch process reads all the records on the `ScheduleReassessment` entity and reassesses each case in turn. The `ScheduleReassessment` entity contains entries for cases that are pending reassessment and should be read only in batch mode. Cases can also be reassessed in deferred mode but in this mode, `ScheduleReassessment` is not used.

The batch process accepts one parameter, which is processingDate:

- The processingDate parameter is optional.
- You can use the processingDate parameter to specify the effective date for which the batch job is run, which is the date that is returned from the `getCurrentDate()` API. Otherwise, the `getCurrentDate()` API returns the current system date as the effective date.

Batch Process Class and Method The class and method for this batch process is `curam.core.intf.ReassessOutstandingCases.reassess`.

FullPropagationToRuleObjects

This batch process performs full propagation of rate table database data to rule objects.

This process should be run whenever there is reason to believe that stored CER rule objects no longer accurately reflect their source rate table database data. Discrepancies can occur whenever incremental propagation of database data has been bypassed, e.g. if rate tables are updated outside of the control of the application.

Note: In general you should not need to run this program unless you have made large numbers of changes to rate tables outside of the application's APIs - if you have made a small number of changes then you can use the "Apply Changes" action in the online administration application.

A summary of discrepancies found is written to the standard application logs. For detailed logging of all processing, the log level should be set to verbose or ultra-verbose.

The batch process takes no parameters.

Batch Process Class and Method The class and method for this batch process is `curam.core.sl.infrastructure.propagator.intf.FullPropagationToRuleObjects.execute`.

RateCreateInitialRuleObjects

This batch process creates the initial CER rule objects corresponding to rate table entries in an environment where the rate table data has been populated outside of the application's APIs.

See the *Propagating Non Cúram Data For Cúram Express Rules* guide for more information.

This batch process takes the following parameter:

- `rateTableType` - The rate table to process (or blank to process all rate tables).

Batch Process Class and Method The class and method for this batch process is `curam.core.sl.infrastructure.rate.intf.RateCreateInitialRuleObjects .prop`

CREOLEBulkCaseChunkReassessmentByProduct

This batch process is provided to identify and perform full reassessment on a large number of "Active" CER cases of a given product type. For any cases where the determination changes as a result of this reassessment, the new determination will be stored and the old one superseded.

Important: As this process will cause reassessment of all cases of the specified type, it may cause a lot of unnecessary reassessments. Where appropriate, a new batch process should be written in order to more precisely identify the cases that require reassessment, especially when the cases are spread across a range of products. For a full explanation of how to write an appropriate batch process see the *Inside Cúram Eligibility and Entitlement Using Cúram Express Rules* guide.

You can run this program when you have made changes to the system that affect a large number of CER cases, and you want to force the system to reassess cases by product (rather than leaving the Dependency Manager batch suite to determine the order in which to reassess cases).

Note: See the *Inside Cúram Eligibility and Entitlement Using Cúram Express Rules* guide for full details on how to choose whether to use this batch process in addition to or instead of the Dependency Manager batch suite.

The type of changes that can affect a large number of CER cases are:

- publishing CER Rule Set Changes;
- publishing CER Product Configuration changes;
- publishing CER Data Configuration changes; and
- applying Rate Table changes.

This batch process takes the following parameters:

- `Product ID` - unique identifier used to run this process for cases of a particular product

  Where no product ID is specified, all cases are processed.
- `Batch Process Instance ID` - unique identifier used to allow multiple instances of the same batch process to run at the same time effectively.

  For example, this process can run for multiple products. Where no instance ID is specified, only one instance of the batch process can run.

Batch Process Class and Method The class and method for this batch process is `curam.core.sl.infrastructure.assessment.intf .CREOLEBulkCaseChunkReassess`

CREOLEBulkCaseChunkReassessmentStream

This batch process is provided to support streaming for the `CREOLE Bulk Case Chunk Reassessment By Product` process. This batch process can only be run in conjunction with the `CREOLE Bulk Case Chunk Reassessment By Product` process. Streaming for batch processes is designed to allow for concurrent execution, possibly on different machines, of batch programs to ensure that the database is used to its full capacity.

This batch process takes the parameter, `Batch Process Instance ID` . The instance ID is a unique identifier used to allow multiple instances of the same batch process to run at the same time effectively. Where no instance ID is specified, only one instance of the batch process can run.

Streaming Multiple Instances of CREOLE Bulk Case Chunk Reassessment By Product Multiple instances of the `CREOLE Bulk Case Chunk Reassessment By Product` batch process can be run concurrently, by giving each instance a different `Batch Process Instance ID` . To start a stream for a particular instance of the `CREOLE Bulk Case Chunk Reassessment By Product` batch process, you must link the `CREOLE Bulk Case Chunk Reassessment Stream` batch process (or multiple stream batch processes) to the particular batch process instance using the `Batch Process Instance ID` parameter. For example, where the `Batch Process Instance ID` is "batch_reassessment_1" for an instance of the `CREOLE Bulk Case Chunk Reassessment By Product` batch process, you must also set the `Batch Process Instance ID` parameter for the `CREOLE Bulk Case Chunk Reassessment Stream` batch process (or multiple stream batch processes) to be "batch_reassessment_1". Any number of `CREOLE Bulk Case Chunk Reassessment Stream` batch processes can be linked to the same instance of the `CREOLE Bulk Case Chunk Reassessment By Product` batch process.

Batch Process Class and Method The class and method for this batch process is `curam.core.sl.infrastructure.assessment .intf.CREOLEBulkCaseChunkReassess`

ApplyProductReassessmentStrategy

Checks each product delivery case for a product to see if the case's support for reassessment (e.g. support for reassessment when closed) has changed due to the change in the product's reassessment strategy.

For each product delivery case for the product:

- if the case was not reassessable under the old strategy but becomes reassessable under the new strategy, then an assessment is performed on the case to build up the dependency records for the case's determination result;
- if the case was reassessable under the old strategy but is no longer reassessable under the new strategy, then the dependency records for the determination result are removed;
- otherwise no action is performed on the case.

This batch process takes the following parameter:

- `Product ID` - unique identifier used to run this process for cases of a particular product

See the *`Inside Cúram Eligibility and Entitlement Using Cúram Express Rules`* guide.

Batch Process Class and Method The class and method for the chunker batch process is `curam.core.sl.infrastructure.assessment .intf.ApplyProductReassessmentStr` me>.

The class and method for the stream batch process is `curam.core.sl.infrastructure.assessment .intf.ApplyProductReassessmentStr`

Redetermine Translator

This batch is used to perform automatic redetermine translator processing when a change to the language skill of a user is made and the change will affect a large volumes of cases.

RedetermineTranslator initiates the following processing steps:

1. Checks all open cases for which a particular user is assigned as the case owner and compares the preferred language of each case participant in the case against the language skill of the user.
2. Updates the translator required indicator for the case participant if necessary.

Class and method

RedetermineTranslator.process

RedetermineTranslatorStream.process

Parameters

Table 1: Batch Parameters

Description of the parameters used to run the batch process.

Parameter name	Description
Instance Identifier	Instance ID. Allows multiple instances of the same batch process to run at the same time.
Processing Date	The date the batch program will be processed.
User Name	Used to specify the case owner whose language skills will be assessed.

Related Information:

The curam.cases.maxnocases.onlineautotranslatordetermination application property controls whether automatic re-determination of translator requirements will occur in batch mode or singly via online processing. If the number of open cases that require processing exceeds the value specified in the application property, re-determination will not occur in online mode, and must be executed in batch instead.

Dependency Manager

For full details on the batch processes included with the Dependency Manager, see the *Cúram Express Rules Reference Guide*.

SubmitPrecedentChangeSet

This batch process is the starting point for the Dependency Manager batch suite, containing a light-weight single-stream process that submits the currently-open batch precedent change set, and creates a new open batch precedent change set, which will be used to capture any subsequent precedent changes identified and queued for batch processing.

Batch Process Class and Method The class and method for this batch process is `curam.dependency.intf.SubmitPrecedentChangeSet.process`.

PerformBatchRecalculationsFromPrecedentChangeSet

This batch process is the heavyweight multiple-stream process that identifies the dependents which are potentially affected by the changes in the submitted precedent change set, and recalculates them.

This batch process must be run once for each dependent type registered with the Dependency Manager.

Batch Process Class and Method The class and method for the chunker batch process is `curam.dependency .intf.PerformBatchRecalculationsFromPrecedentChangeSet.p`

The class and method for the stream batch process is `curam.dependency .intf.PerformBatchRecalculationsFromPrecedentChangeSetSt`

CompletePrecedentChangeSet

The end point for the Dependency Manager batch suite, containing a lightweight single-stream process that completes the currently-submitted batch precedent change set.

Batch Process Class and Method The class and method for this batch process is `curam.dependency.intf.CompletePrecedentChangeSet.process`.

Dependency Manager Batch Tooling

Tooling has been provided in the form of Ant scripts to assist in the running of the Dependency Manager batch suite. Using the tooling will ensure that the batch jobs are run in the correct order, and will allow the user to set a property to manage the performance of the batch run. There are two parts to the tool, the first part produces an ant script specific to the system its being run on, while the second involves invoking this ant script to run the batch suite.

Prerequisites

The following environment variables must be set as per normal Cúram operation:

- SERVER_DIR
- CURAMSDEJ

Database drivers should be in the SERVER_DIR/drivers directory. The tool will pickup the Database connection details from the normal Bootstrap.properties file.

Producing the Specific Ant Script

Overview:

In this step, the tool examines the supported dependent types on the system, and then constructs an ant script to ensure each of these dependent types gets processed during the batch suite run. Additionally, certain configuration options can be specified in the conf/DependencyManagerBatch.properties file as follows:

- batch.default.threads=3 - This specifies the default number of threads to use in the generated ant script (If this property has a value of n, one chunker and n-1 streams are created for processing each dependent type).

Additionally, for each dependent type, the default can be overridden by specifying a property for that type as below:

- batch.CADETERRES.recalculation.threads=5 - This example would override the number of threads used to process the Case Determination Reassessment dependent type.

Running the Step:

To run the step, simply execute the 'ant' command from within the DependencyManagerBatch directory.

Output:

This step produces a file, DependencyManagerBatch.xml within the current directory.

Invoke the Script to run the Batch Suite

Overview:

The second step of the tool uses the generated output from the first. The script performs the following steps:

- Check if any dependents must be processed, if none then exit.
- If dependents require processing then submit the current batch precedent change set.
- For each dependent type, process the recalculations using the specified number of threads.
- Finally, complete the precedent change set.

Running the Step:

To run this step, simply execute the command 'ant -f DependencyManagerBatchBase.xml'.

Output:

If no dependents required processing, a message will be output stating this and the script will exit. Otherwise, an individual log file is written to the Logs folder for each batch process / thread executed.

Financial

GenerateInstructionLineItems

This batch process is provided to create instruction line items for all financial components that are due for processing. The input parameters for this process identify the Financial Components that are due for processing.

This batch process performs the following:

- Where one date is specified for "Date From" and "Date To", all Financial Components with a processing date earlier than this date are retrieved. Where no date is specified, the current system date is used.
- Similarly, where no delivery method is specified, all delivery methods are processed.
- Using the "Date To", this process checks the Financial Calendar to determine if there are subsequent days that should be included in the current processing of Financial Components.
- This process retrieves all Financial Components due for processing and groups each financial component by case.
- This process reassesses each case to determine if a change in circumstance may have changed Financial eligibility.
- For each remaining Financial Component, the amount and cover period is calculated and an instruction line item is created.
- This process rolls forward the processing date of the financial component and where this causes the financial component to logically reach the end of its lifetime, the financial component is expired.

This batch process takes the following parameters:

- `Batch Process Instance ID` - unique identifier used to allow multiple instances of the same batch process to run at the same time effectively

For example, this process can run for multiple delivery methods. Where no instance ID is specified, only one instance of the batch process can run.

- `Date From` - start date for identifying Financial Components to include for processing
- `Date To` - end date for identifying Financial Components to include for processing
- `Delivery Method` - method of delivery to be used for identifying Financial Components to include for processing

Batch Process Class and Method The class and method for this batch process is `curam.core .intf.GenerateInstructionLineItems.processAllFinancialComponen`

Note: This batch process should run to completion before attempting to run *`GenerateInstruments`*.

GenerateInstructionLineItemsStream

This batch process is provided to support streaming for the `Generate Instruction Line Items` process. This batch process can only be run in conjunction with the `Generate Instruction Line Items` process. Streaming for batch processes is designed to allow for concurrent execution, possibly on different machines, of batch programs to ensure that the database is used to its full capacity.

This batch process takes the parameter, `Batch Process Instance ID` . The instance ID is a unique identifier used to allow multiple instances of the same batch process to run at the same time effectively. Where no instance ID is specified, only one instance of the batch process can run.

Streaming Multiple Instances of Generate Instruction Line Items Multiple instances of the `Generate Instruction Line Items` batch process can be run concurrently, by giving each instance a different `Batch Process Instance ID` . To start a stream for a particular instance of the `Generate Instruction Line Items` batch process, you must link the `Generation Instruction Line Items Stream` batch process (or multiple stream batch processes) to the particular batch process instance using the `Batch Process Instance ID` parameter. For example, where the `Batch Process Instance ID` is "generate_instruction_line_items_1" for an instance of the `Generate Instruction Line Items` batch process, you must also set the `Batch Process Instance ID` parameter for the `Generate Instruction Line Items Stream` batch process (or multiple stream batch processes) to be "generate_instruction_line_items_1". Any number of `Generate Instruction Line Items Stream` batch processes can be linked to the same instance of the `Generate Instruction Line Items` batch process.

Batch Process Class and Method The class and method for this batch process is `curam.core.intf.GenerateInstructionLineItemsStream.process`.

GenerateInstruments

This batch process performs the following:

- Identifies Instruction Line Items that are to be processed
- Rolls up Instruction Line Items into Instructions
- Creates an Instrument (where appropriate)

This batch process takes the parameter, `Batch Process Instance ID` . The instance ID is a unique identifier used to allow multiple instances of the same batch process to run at the same time effectively. Where no instance ID is specified, only one instance of the batch process can run.

Batch Process Class and Method The class and method for this batch process is `curam.core.intf.GenerateInstruments.processInstructionLineItemsDue`.

Note: This process should run after the completion of the *`GenerateInstructionLineItems`* and before the *`IssueConcernPayments`* process.

GenerateInstrumentsStream

This batch process is provided to support streaming for the `Generate Instruments` process. This batch process can only be run in conjunction with the `Generate Instruments` process. Streaming for batch processes is designed to allow for concurrent execution, possibly on different machines, of batch programs to ensure that the database is used to its full capacity.

This batch process takes the parameter, `Batch Process Instance ID` . The instance ID is a unique identifier used to allow multiple instances of the same batch process to run at the same time effectively. Where no instance ID is specified, only one instance of the batch process can run.

Streaming Multiple Instances of Generate Instruments Multiple instances of the `Generate Instruments` batch process can be run concurrently, by giving each instance a different `Batch Process Instance ID` . To start a stream for a particular instance of the `Generate Instruments` batch process, you must link the `Generation Instruments Stream` batch process (or multiple stream batch processes) to the particular batch process instance using the `Batch Process Instance ID` parameter. For example, where the `Batch Process Instance ID` is "generate_instruments_1" for an instance of the `Generate Instruments` batch process, you must also set the `Batch Process Instance ID` parameter for the `Generate Instruments Stream` batch process (or multiple stream batch processes) to be "generate_instruments_1". Any number of `Generate Instruments Stream` batch processes can be linked to the same instance of the `Generate Instruments` batch process.

Batch Process Class and Method The class and method for this batch process is `curam.core.intf.GenerateInstrumentsStream.process`.

IssueConcernPayments

This batch process performs the following:

- Identifies Utility, Service Supplier, or Product Providers concerns that may be due for payment based on the details specified on input
- Retrieves any outstanding payment instructions for each concern and rolls these into one payment instrument for issue to the concern
- Updates the next payment date for the concern

This batch process takes the following parameters:

- `Date From` - start of the date range for which Concern Payments are to be processed
- `Date To` - end of the date range for which Concern Payments are to be processed
- `Method Of Payment` - method of delivery to be used for identifying Financial Components to include for processing

  Where this is not specified, all methods of payments are processed.
- `Concern Type` - one or either Utility, Service Supplier, or Product Provider, where the batch process is to be run for one concern type only

  Where this is not specified, it is run for the three concern types.

Batch Process Class and Method The class and method for this batch process is `curam.core.intf.IssueConcernPayments.issueConcernTypePayment`.

Note: This process should run after the completion of the *`GenerateInstruments`* and before the *`GeneratePayslips`* process.

GeneratePayslips

This batch process performs the following:

- Finds Payslips that have not yet been issued
- Retrieves the necessary data to output information in a readable format
- Updates the status of the Payslip to "Issued"

This batch process is a sample implementation to demonstrate that Payslips can be generated from Cúram. The output data (which currently generates one output file per recipient type) is not the definitive implementation for this batch process. For example, a customized implementation may provide one output file per recipient.

This batch process takes no parameters.

Batch Process Class and Method The class and method for this batch process is `curam.core.intf.GeneratePayslips.generateNewPayslips`.

Note: This process should run after the completion of the *`IssueConcernPayments`* process.

LoadServiceSupplierReturns

Service supplier returns can be stored in an input file. This batch process loads the service supplier returns from the input file.

This batch process takes the parameter, `File Name` . This is the directory where the input file resides.

Batch Process Class and Method The class and method for this batch process is `curam.core.intf.LoadServiceSupplierReturns.loadSupplierReturns`.

ProcessPaymentInstrumentTypes

This batch process performs the following:

- Retrieves payments that have not yet been issued
- Gathers the necessary data for payment not yet issued and outputs this to a file used to interface to an external system
- Updates the status of the payment to indicate that it has been issued

This batch process takes the parameter, `Delivery Method` . This is the method of delivery to be used for identifying un-issued payments. Where this is not specified, all methods of payment are processed.

Batch Process Class and Method The class and method for this batch process is `curam.core.intf.ProcessPaymentInstrumentTypes.processPmtInstrumentType`.

Note: This batch process is a sample implementation to demonstrate how the application may interface with payment processing systems. It may be run only for un-issued payments. Where the delivery method is specified, all un-issued payments for that delivery method are processed; if the delivery method is not specified, all un-issued payments are processed.

ExpirePayments

This batch process performs the following:

- Retrieves all payment instruments based on the delivery method specified on input, whose effective date is on or before the current date minus the expiry period
- Sets the status to expired for each of these payment instruments and outputs a record to the log file

  This log file may be used to communicate to the appropriate financial institution that the payments are not to be encashed.

This batch process takes the following parameters:

- `Expiry Period` - number of days which constitutes the expiry period
- `Delivery Method` - method of delivery to be used for identifying Payments that are due for expiry

  Where this is not specified, all methods of payment are processed.

Batch Process Class and Method The class and method for this batch process is `curam.core.intf.ExpirePayments.expirePaymentInstrument`.

PaymentReconciliation.

This batch process reconciles an account by comparing what was due to be paid with what was actually paid. Any discrepancies found in this comparison are generated in a report.

This batch process takes the following parameters:

- `File Path` - directory where the output file resides
- `File Name` - full name of the output file, including extension, that contains the details of the payments

When run, this batch process looks for the *`fileName`* specified in the *`filePath`* specified.

Batch Process Class and Method The class and method for this batch process is `curam.core.intf.PaymentReconciliation.reconcilePayments`.

ReconcileCaseAccount

This batch process performs the following:

- Reconciles all liability cases on which an underpayment has been applied
- Reconciles all liability cases on which an overpayment has been applied

This batch process takes no parameters.

Batch Process Class and Method The class and method for this batch process is `curam.core.intf.ReconcileCaseAccount.reconcileCaseAccount`.

Workflow

ScanTaskDeadlines

The processing performed by this batch process may also be accomplished by that invoked in the `ProcessTaskDeadlines` and `ProcessTaskDeadlinesStream` batch processes. This batch process performs the following:

- Searches for overdue tasks , i.e., tasks with a due date and time in the past
- If a deadline handler function has been specified in the associated workflow process definition for the deadline, then this handler is invoked.
- Otherwise, if the complete activity indicator has been set to true, then the workflow engine completes the activity associated with the deadline and progresses the workflow.
- The deadline that has been processed is removed to ensure it is not processed again.
- The data associated with the *Context_Deadline* workflow data object attribute is persisted.
- If there is a task associated with the deadline that has expired, a task history record is written detailing this fact.

This batch process takes no parameters.

Batch Process Class and Method The class and method for this batch process is `curam.core.facade.intf.ScanTaskDeadlines.scanDeadlines`.

Note: When executing this batch process from the command line the `batch.username` command line parameter must be supplied, and must refer to a valid user with appropriate privileges.

ProcessTaskDeadlines

This batch process performs the following:

- Searches for overdue tasks , i.e., tasks with a due date and time in the past
- If a deadline handler function has been specified in the associated workflow process definition for the deadline, then this handler is invoked.
- Otherwise, if the complete activity indicator has been set to true, then the workflow engine completes the activity associated with the deadline and progresses the workflow.
- The deadline that has been processed is removed to ensure it is not processed again.
- The data associated with the *Context_Deadline* workflow data object attribute is persisted.
- If there is a task associated with the deadline that has expired, a task history record is written detailing this fact.

This batch process takes no parameters.

Batch Process Class and Method The class and method for this batch process is `curam.core.facade.intf.ProcessTaskDeadlines.process`.

Note: When executing this batch process from the command line the `batch.username` command line parameter must be supplied, and must refer to a valid user with appropriate privileges.

ProcessTaskDeadlinesStream

This batch process is provided to support streaming for the `ProcessTaskDeadlines` process.

This batch process can only be run in conjunction with the `ProcessTaskDeadlines` process. Streaming for batch processes is designed to allow for concurrent execution, possibly on different machines, of batch programs to ensure that the database is used to its full capacity.

This batch process takes the parameter, `Batch Process Instance ID`. The instance ID is a unique identifier used to allow multiple instances of the same batch process to run at the same time effectively. Where no instance ID is specified, only one instance of the batch process can run.

Streaming Multiple Instances of Process Task Deadlines Multiple instances of the `Process Task Deadlines` batch process can be run concurrently, by giving each instance a different `Batch Process Instance ID` . To start a stream for a particular instance of the `Process Task Deadlines` batch process, you must link the `Process Task Deadlines Stream` batch process (or multiple stream batch processes) to the particular batch process instance using the `Batch Process Instance ID` parameter. For example, where the `Batch Process Instance ID` is "process_task_deadlines_1" for an instance of the `Process Task Deadlines` batch process, you must also set the `Batch Process Instance ID` parameter for the `Process Task Deadlines Stream` batch process (or multiple stream batch processes) to be "process_task_deadlines_1". Any number of `Process Task Deadlines Stream` batch processes can be linked to the same instance of the `Process Task Deadlines` batch process.

Batch Process Class and Method The class and method for this batch process is `curam.core.facade.intf.ProcessTaskDeadlinesStream.process`.

RestartTask

This batch process performs the following:

- Searches for tasks that have a status of `Deferred` and whose restart date time has passed the current date time.
- For each of those tasks, it sets the status to `Open` . It also sets the restart date time back to the zero date time.

This batch process takes no parameters.

Batch Process Class and Method The class and method for this batch process is `curam.core.facade.intf.RestartTask.restart`.

ExpireWaitListEntry

When a client is added to a wait list for a resource, an expiry date can be specified. The expiry date could be entered by the user, or created by the system. This batch process is used to expire a wait list entry if the client is neither allocated a resource nor removed from the list before the expiry date is reached. The process expires the Wait List entry if the Wait List expiry date is on or before the batch processing date, and if the Wait List entry is in the 'Open' state. If the Wait List requires the renumbering, then the system renumbers the Wait List by decrementing by 1 the position of all the Wait List entries in an 'Open' state, and with positions higher than or equal to the position of the expired entry. After the Wait List Entry is expired, the process raises a workflow event 'WAITLIST. WAITLISTENTRYEXPIRED'. It is expected that this batch process would be scheduled to run daily.

Batch Process Class and Method The class and method for this batch process is `curam.core.intf.ExpireWaitListEntry.expireWaitListEntry`.

WaitListReview

This batch process is used to raise a workflow event 'WAITLIST. WAITLISTENTRYSELECTEDFORREVIEW' to generate wait list review tasks for all the eligible wait list entries that are due for review. The eligible wait list entries are all the wait list entries with review dates on or before the batch processing date. If there is no review date set, then it is derived by subtracting the configured status review reminder period from its expiry date. The status review reminder period is configured by the system administrator using the 'curam.waitlist.statusreviewreminderperiod' property. For example, if the Wait List review date is on the 30th and the status review reminder period is set as 5, then the batch process 'WaitListReview' raises the workflow event to generate the wait list review task on the 25th. For successful generation of the review tasks, the property 'curam.batchlauncher.dbtojms.enabled' should be set to true and appropriate values must be defined for 'curam.batchlauncher.dbtojms.notification.host' and 'curam.batchlauncher.dbtojms.notification.port' properties. These properties are configured by the system administrator. The batch process would be scheduled to run daily.

Batch Process Class and Method The class and method for this batch process is `curam.core.sl.intf.WaitListReview.processWaitListEntriesDueForReview`.

Input Interfaces

LoadPaymentsReceived

This batch process performs the following:

- Validates the data for each record in the input file, *`PaymentReceivedFile.txt`*
- Loads the payment received record onto a concern's account where a concern is identified; otherwise, loads the payment received record into a suspense account
- Maintains a set of control totals and compares these to the control total record at the end of the input file
- Rolls back all processing if the running totals do not match the data from the input file

This batch process takes the following parameters:

- `File Path` - directory where the input file resides
- `File Name` - full name of the input file, including extension, that contains the details of the payments received.

When run, this batch process looks for the *`fileName`* specified in the *`filePath`* specified.

Batch Process Class and Method The class and method for this batch process is `curam.core.intf.LoadPaymentsReceived.loadPaymentReceivedFile`.

Output Interfaces

Calendar Export

This batch process performs the following:

- Uses the parameters to gather calendar activities
- Exports the activities from the application into an output file in vCalendar format.

  This output file can then be imported into an external calendar.

Parameter Requirements In order for this batch process to run, it is required that you enter either the `User Name` parameter, the `Organization ID` parameter, or both the `Start Date` and `End Date` parameters.

This batch process takes the following parameters:

- `Export File Path` - directory where the output file resides
- `User Name` - name of the user whose activities are exported

  If this parameter is set, the batch process exports only the activities of the specified user.
- `Organization ID` - unique identifier for the organization whose activities are exported

  If this parameter is set, the batch process exports the activities of the organization.
- `Start Date` - start date of the date range for activities to be exported to an output file
- `End Date` - end date of the date range for activities to be exported to an output file

  In order for the batch process to export activities for a specified date range, both the `Start Date` and `End Date` parameter must be set.

Batch Process Class and Method The class and method for this batch process is `curam.core.intf.CalendarExport.exportActivityDetails`.

Generate Ledger Interface

This gathers financial transactions for a specified date, or date range, and exports them from the application into an output file. The output file contains details of Instruction Line Items for the specified date or date range. This output file can then be imported into the General Ledger.

This batch process takes the following parameters:

- `Date From` - start of the date range for financial transactions to be exported to an output file

  If DateFrom is found to be a null date, i.e., not specified by a user, an error is thrown and the batch process will not succeed.
- `Date To` - end date of the date range for financial transactions to be exported to an output file

If DateTo is found to be a null date, i.e., not specified by a user, an error is thrown and the batch process will not succeed.

- `Creation Date Search Indicator` - indicates whether the extract is based on creation date or effective date

  If this indicator is set to true, a creation date range search is performed; otherwise, an effective date range search is performed.

Batch Process Class and Method The class and method for this batch process is `curam.core.intf.GeneralLedgerInterface.exportFinancialDetails`.

1.2 Provider Management batch processes

A list of the batch processes that provide Provider Management functions in the application.

You can typically configure these batch processes in the system administration application.

Unless specified otherwise, you can run batch processes in any order.

CreateRosters

This batch process generates rosters for service offerings.

CreateRoster initiates the following processing steps:

1. Checks every active service offering to see whether a service is required and creates a roster where required.
2. Sets date range for the roster based on the configured roster generation frequency for the service offering. The first date is set as the processing date.
3. Retrieves all provider offerings for a service offering and date range to determine whether a roster exists.
   - If a roster does not exist, a roster is created.
   - If a roster exists with a different date range, the roster is updated with the new date range.

Class and method

CreateRosters.process

Dependencies

You must run the CreateRoster and ProcessSALI batch processes in the following order:

1. CreateRoster
2. ProcessSALI

This sequence ensures that SALIs created by ProcessSALI are based on the latest service offering rosters.

Parameters

Description of the parameters that are used to run the batch process

Parameter	Description	Default value
Batch Processing Date	The date the batch program is processed.	n/a

CreateRostersStream uses batch streaming to distribute the processing load of CreateRosters.

Related reference

ProcessSALI on page 31
ProcessSALI processes service authorization line items to add them to rosters.

Process Service Authorization Line Item

This batch process processes service authorization line items to add them to rosters.

Process Service Authorization Line Item (ProcessSALI) initiates the following processing steps:

1. Generates all service authorization line items for which a matching service authorization exists for each case participant role on the processing date.
2. Obtains the date range for retrieval of the service authorization line items from the service offering and processing date.
3. Adds the service authorization line items to the correct roster as created by the CreateRosters batch process.

Note: The ProcessSALI batch process should always be executed immediately after the CreateRosters batch to ensure that the latest rosters are used to create the service authorization line items.

Class and method

ProcessSALI.process

Parameters

Table 2: Batch Parameters

Description of the parameters used to run the batch process.

Parameter name	Description	Default value
Batch Processing Date	The date the batch program will be processed. This is an optional parameter. If processing date is not specified, the current date is used.	n/a
Service Offering ID	The service offering ID.	n/a

ProcessSALIStream uses batch streaming to distribute the processing load of ProcessSALI.

Related reference

CreateRosters on page 30
Should be executed before the ProcessSALI batch process to ensure the latest rosters are used to generate service line items.

GenerateRostersWithDate

This batch process generates rosters and roster line items for a specific date for provider services for which service authorization exist.

GenerateRostersWithDate initiates the following processing steps:

1. Generates an attendance roster up to the specified date for each provider service for which service authorizations exists.
2. Generates a roster line item for every client authorized in the service authorization line item for the period of the roster.
3. Generates a roster for all the open or in-progress service authorization line items for the provider.

   Note: A roster is not generated if a roster already exists during the frequency period calculated from the processing date for the particular provider service.
4. Adds clients authorized for the provider service to the new roster as a roster line item. If an open roster line item entry already exists, the entry is extended to accommodate the new authorization values.

Class and Method

GenerateRostersWithDate.generateRostersWithDate

Parameters

Description of the parameters used to run the batch process.

Parameter name	Description	Default value
Processing Date	The date the batch program will be processed.	n/a

IssuePaperRosters

This batch process processes service offerings to issue blank paper rosters for all provider offerings for which a paper roster is required.

IssuePaperRosters initiates the following processing steps:

1. Checks all provider offerings for a given service offering to see if there is a need to issue a paper roster for the associated rosters.
2. Generates a paper roster for every service offering where one is required.

The date range for the paper roster is calculated based on the configured roster generation frequency value and the processing date.

When a paper roster is generated for a roster, events are raised. These events can be used to automate the issuance of the paper roster to the provider.

Class and method

IssuePaperRosters.process

Parameters

Description of the parameters used to run the batch process.

Parameter name	Description	Default value
Batch Processing Date	The date the batch program is processed.	n/a

IssuePaperRostersStream uses batch streaming to distribute the processing load of IssuePaperRosters.

StartEffectiveServiceDeliveries

Checks if any of the service delivery records with a state of 'Not Started' have a cover period start date which is effective based on the current date. Any service delivery satisfying this check will have its state updated to 'In Progress'.

StartEffectiveServiceDeliveries intitiates the following processing steps:

1. Searches for all service deliveries in system which are in 'Not Started' State.
2. Processes all qualifying service deliveries to start service delivery.
3. Updates the status to 'In Progress'
4. Updates the service start date for the cover period.

Class and method

StartEffectiveServiceDeliveries.startEffectiveServiceDeliveries.

Parameters

StartEffectiveServiceDeliveries has no input parameters.

Introduction

This chapter lists the batch processes which provide CPM functionality. The following sections list the Provider Management batch processes that are configurable in the system administration application.

Note: Unless specified otherwise, batch processes may be run in any order.

Batch Parameters and Processing Date

Quite a number of the batch jobs within the application take 'processingDate' as a parameter. When this is specified, it will be the date the batch job is run for, i.e., it will be the date returned from the getCurrentDate() API. When not specified, the batch job is run for the system date. This parameter is not typically listed as one of the parameters in the descriptions below.

RosterSubmissionDue

This batch process is used to raise an event 'ROSTER.ROSTER_NOT_SUBMITTED_AFTER_DUE_DATE' on the day after the roster submission due date, if the roster has not yet been submitted, i.e., the roster is in 'Normal' state. The event 'ROSTER.ROSTER_NOT_SUBMITTED_AFTER_DUE_DATE' is used to send the notifications. The submission due date is calculated by adding the roster grace period to the last day of the roster period. The roster grace period is configured by the administrator for a service offering. Due to the nature of the batch job, it is expected that this batch run would be scheduled to coincide with the roster generation frequency. For example, if the roster generation frequency is monthly, and there is no roster grace period, then this batch job could be scheduled to be run the day after the last day of the roster period, i.e. the 1st of every month. If a number of different roster generation frequencies are in use, the batch could be run daily.

Batch Process Class and Method The class and method for this batch process is `RosterSubmissionDue.generateRosterNotification`.

ProcessContractRenewalNotifications

This batch process is used to raise the workflow events 'ContractNotificationEvent.RENEWPROVIDERCONTRACT' and 'ContractNotificationEvent.RENEWPGCONTRACT' to send notifications to the user, a certain number of days prior to contract expiration. The number of days is configured by the system administrator using the 'curam.contracts.renewal.noofdaysevents' property. The batch process raises the workflow events when the difference between the contract end date and the current date is less than 'curam.contracts.renewal.noofdaysevents'. The workflow event 'ContractNotificationEvent.RENEWPROVIDERCONTRACT' is for the provider contract and the workflow event 'ContractNotificationEvent.RENEWPGCONTRACT' is for the provider group contract. This batch process run would be scheduled to run daily, or alternatively if an agency had a regular contract renewal date, the process could be scheduled to run a certain number of days prior to that date.

Batch Process Class and Method The class and method for this batch process is `ProcessContractRenewalNotifications.processContractRenewalNotifications`.

ProcessLicenseRenewalNotifications

This batch process is used to raise a workflow event 'LicenseNotificationEvent.RENEWLICENSE', that sends a notification to the user a certain number of days prior to license expiration. The number of days is configured by the system administrator using the 'curam.license.renewal.noofdaysevents' property. The batch process raises the workflow event when the difference between the license end date and the current date is less than 'curam.license.renewal.noofdaysevents'. This batch process run would be scheduled to run daily, or alternatively if an agency had a regular license renewal date, the process could be scheduled to run a certain number of days prior to that date.

Batch Process Class and Method The class and method for this batch process is `ProcessLicenseRenewalNotifications.processLicenseRenewalNotifications`.

ActivateCPMLiabilityCases

This batch process is used to activate the liability cases created as a part of CPM financial processing. The process retrieves all the liability cases created for the different financial product delivery cases in CPM (Invoice, Attendance, Contract and Placement), and sends them for processing so that they could be activated, if they are eligible for activation. This process then invokes the API 'ProductDeliveryActivationEligibility'. The API uses the method 'assessEligibilityForCase(CaseIDDetails)' to process the case for activation. This batch process would be scheduled to run daily and would be run before the financial batch processes 'GenerateInstructionLineItem' and 'GenerateInstruments' are run.

Batch Process Class and Method The class and method for this batch process is `ActivateCPMLiabilityCases.approveAndActivateCPMCases`.

CreatePlacementEvidence

This batch process is used to create placement evidence for placements in respect of services that are configured to be paid based on placement data. The batch process first identifies the eligible placements, i.e., those placements for which evidence needs to be created. If the placement end date is not specified, the batch process uses the placement payment frequency and calculates the next occurrence till the next payment date. If the placement end date is specified and if it happens to be before the next payment date, then the batch process considers the placement end date and calculates the next occurrence till the placement end date.

For example, if there is a placement from 1st March to 15th April and the payment date is on every 29th, then the system should create placement evidences from 1st March to 29th March and 30th March to 15th April. Where evidence doesn't already exist, then it is created and added to the CPM Placement product delivery case, which triggers reassessment of the case. The batch processes 'CreatePlacementEvidence' and 'ReassessOutstandingCases' have to be scheduled to run before running 'GenerateInstructionLineItem' and 'GenerateInstrument' batch processes

for placement based payments. The financials batch jobs 'GenerateInstructionLineItem' and 'GenerateInstruments' can then be used to generate payments to providers.

Batch Process Class and Method The class and method for this batch process is `CreatePlacementEvidence.createPlacementEvidence`.

ExpireReservation

This batch process is used to expire active reservations when their date of expiry is reached. The reservation expiry date is calculated by adding the reservation grace period to the start date of the reservation. If the reservation expiry date for an active reservation is earlier than the processing date, then the active reservation is expired. The reservation grace period is set for the provider and is configured by the user. If the reservation grace period is not set at the provider level, then the system considers the grace period set at the agency level. The grace period at the agency level is configured by the system administrator using the 'curam.cpm.reservation.agencygraceperiod' property. After the active reservation is expired, a reservation expiration notification is sent to the case owner of the case, stating that the reservation has expired. It is expected that this batch process would be scheduled to run daily.

The batch process takes no parameters.

Batch Process Class and Method The class and method for this batch process is `ExpireReservation.expireReservation`.

GenerateCPMPayslips

This batch process is used to generate pay slips for all the un-issued pay slips (Invoice based, Attendance based, Placement and Flat-Rate Contract) from the payslip table. The batch process uses the property 'curam.cpm.batch.generatepayslips.commitcountenabled' to determine whether the batch process must use the commit count for processing and the property 'curam.cpm.batch.generatepayslips.commitcount' to determine the number of records to be processed by the batch before the results are committed to the database. Both these properties are configured by the system administrator. This batch process would be scheduled to run after the financial batch processes 'GenerateInstructionLineItem' and 'GenerateInstruments' are run.

Batch Process Class and Method The class and method for this batch process is `GeneratePayslips.generateNewPayslips`.

GenerateRosters

This batch process is used to generate attendance rosters for providers for which service authorizations exist, i.e. they have clients expected to receive services from them in the upcoming period. The process will also generate a roster line item for every client authorized in the service

authorization line item for the period of the roster. The batch process would be scheduled to run based on the roster generation frequency, configured for the provider. For example, if the roster generation frequency is monthly, then the batch would be scheduled to run on the 1st of every month. For more information on entering attendance and processing of rosters, please refer to the Provider Management Guide.

Batch Process Class and Method The class and method for this batch process is `GenerateRosters.generateRosters`.

RosterSubmissionReminder

This batch process is used to generate reminder notifications which have not been generated previously for roster submission, by either using the XSL or the XML templates for the roster event RosterNotificationEvent.ROSTER_SUBMISSION_REMINDER. When the 'Roster Submission Reminder Required' indicator is set to true, the system sends the notifications to the relevant providers, a certain number of days (configured by the user) prior to the roster submission due date, only if the reminder has not already been sent.

The 'Roster Submission Reminder Required' indicator is configured by the system administrator using 'curam.cpm.attendance.service.submissionreminderrequired' property. When this indicator is set to true, then the system considers 'curam.cpm.attendance.service.rostersubmissionreminderperiod' property to get the number of days prior to which the reminders have to be sent. This property is also configured by the system administrator. For example, if this batch process is run on 18th and the property 'curam.cpm.attendance.service.rostersubmissionreminderperiod' is set to 2, then the notifications are sent for all the rosters that have submission due date as 20th. It is expected that this batch process would be scheduled to run daily.

Batch Process Class and Method The class and method for this batch process is `RosterSubmissionReminder.createNotification`.

ActivateProviderPaymentCorrectionCases

This batch process is used to process and activate the payment correction cases which are used to manage over and underpayments when CPM financial product deliveries (Invoice, Attendance, Contract and Placement) are configured not to use rolled up reassessment. The process retrieves all the payment correction cases created for the product delivery cases in each of the four financial product deliveries. The process then processes and activates the payment correction cases using the API 'paymentCorrection'. This batch process would be scheduled to run daily before the financial batch processes 'GenerateInstructionLineItem' and 'GenerateInstruments' are run.

Batch Process Class and Method The class and method for this batch process is `ActivateProviderPaymentCorrectionCases.processPaymentCorrectionCases`.

ImportTaxonomyData

This batch process is used to import a taxonomy xml file into the system. The process is run each time after a new or the updated taxonomy xml file is available. When importing, the system checks the version numbers of any previously imported taxonomy versions against the version number of the taxonomy file that is being imported. The version number of the file is derived from the release date and the country code. If a version of the taxonomy which is imported already exists in the system with a different language (i.e. the two files are translations of the same taxonomy version), then the translations from the file being imported are added to the existing taxonomy. When a later version of a taxonomy is imported, the system compares the existing and new versions, and identifies any changes. This process is described in more detail in the Provider Management Guide.

Batch Process Class and Method The class and method for this batch process is `ImportTaxonomyData.importTaxonomy`.

1.3 Decision Assist batch processes

A list of the batch processes that provide Decision Assist functions in the application

You can typically configure these batch processes in the system administration application.

Unless specified otherwise, you can run batch processes in any order.

Batch Parameters and Processing Date

Quite a number of the batch jobs within the application take 'processingDate' as a parameter. When this is specified, it will be the date the batch job is run for. When not specified, the batch job is run for the system date.

BatchUploadICDCodes

This is the batch process that is used to perform batch upload of the ICD Code files that contain the classification of the medical conditions.

This batch job populates ICDCODE and ICDCODEVERSION tables. Following are the input parameters that are passed to this batch :

Table 3: BatchUploadICDCodes Input Parameters

Input Parameter	Description
`classification`	The category of ICD code.
`fileName`	Name of the ICD codes file to be read.
`filePath`	Full path of the ICD codes file to be read.

Input Parameter	Description
`processingDate`	Processing date of the batch.
`versionNumber`	Version no. to be associated with the ICD Code.
`volume`	Volume of the ICD Code Version.

Batch Process Class and Method The class and method for this batch process is `ICDCodes.batchUploadICDCodes`.

1.4 Financial batch processes

Financial batch processes are used to create and maintain instruction line items and their associated financial instructions.

Introduction

This appendix provides an overview of each batch processes in the Financial Manager application. However, before a description of each one is given, a number of prerequisites to running the batch processes, as well as some other general information on the topic, are discussed. These are as follows:

1. Submitting to the batch queue
2. Running the Batch Launcher
3. Running a batch program from the Command Line
4. Financials batch suite
5. Business processing date
6. Outputs/emails/reports

Submitting to the batch queue

Each batch process to be executed must first be submitted to the batch queue via the System Administrator application.

This is done by selecting the *Execute* action for that batch process from the list. If parameters are required, the user is prompted to enter them on the Execute Batch Process page. The parameters vary depending on the batch process and some are optional. Once the user enters the parameter value(s), they should then click the *Execute* button to submit the process to the queue. For batch processes that do not require any parameters, the user must confirm the execution of the process. Additionally, the user has the option of canceling at this point.

On submission to the queue, the following entities are populated, BatchParamValue only being populated if parameters are required:

- BatchProcRequest
- BatchParamValue

One important thing to note about submitting processes to the queue is that they should be submitted in the order in which they must run, so if process A must run before process B, process A must be executed first. This is particularly important when creating financials.

Running the batch launcher

Processes that are submitted to the queue are picked up by the batch launcher. The batch launcher is part of the Server Development Environment (SDEJ).

When the batch launcher is run, the first thing it does is start the stand alone keyserver - this will be required if any batch process is performing inserts onto the database.

When all the batch processes have executed, the batch launcher will stop the stand alone keyserver.

Note that batch process streaming allows multiple instances of a single batch process to be run in parallel, making full use of the available processing.

For more information about batch process streaming, see the *Batch Streaming Developers Guide*.

Running a batch program from the command line

Run a command from the main project directory, *EJBServer*.

The following command can be run from the main project directory, *EJBServer*, substituting the appropriate values for <username>, <ClassName> and <OperationName> as well as any parameter name-value pairs:

```
build runbatch
-Dbatch.username=<username>
-Dbatch.program=curam.core.intf.<ClassName>.<OperationName>
-Dbatch.params="param1=param1value, param2=param2value"
```

Financials batch suite

To create the financial records, run four batch processes in sequence.

In order for the financials records to be created, the following batch processes must be run in this sequence:

1. DetermineProductDeliveryEligibility
   - Parameters - instanceID, processingDate and productID
   - Processes - Cases to FCs
   - Pre-run Status - Approved ('CS6')
   - Post-run Status - Active ('CS1') if Eligible, PendingClosure ('CS7') if Ineligible, Suspended ('CS2') if error occurred
   - Summary of Processing - Identifies cases with status of Approved, Activates and assesses case eligibility, and generates FCs
2. GenerateInstructionLineItems

- Parameters - deliveryMethod, instanceID, processingDate, processingDateFrom, processingDateTo and productID
- Processes - FCs to ILIs
- Pre-run Status - Live ('LIV')
- Post-run Status - Live ('LIV') if the FC is not expired; or Closed ('CLD') if nextProcessingDate > processingDate
- Summary of Processing - Reassesses each case being processed, generates ILIs for live FCs and rolls forward their nextProcessingDate. Expires FCs if nextProcessingDate is after processing date

3. GenerateInstruments; and optionally

- Parameters - instanceID and processingDate
- Processes - ILIs to Instructions and Instruments
- Pre-run Status - Unprocessed ('UNP')
- Post-run Status - Processed ('PRO')
- Summary of Processing - Picks up unprocessed ILIs and rolls them into instructions and instruments

4. GeneratePayslips

- Parameters - processingDate
- Processes - Payslips and PayslipInstructions
- Pre-run Status - Pending Issue ('PS2') or Created ('PS1')
- Post-run Status - Issued ('PS3')
- Summary of Processing - Picks up Payslips pending issue and generates the Payslip for them

Business processing date

The business processing date is an optional parameter that is common to all batch jobs.

The parameter is distinct from the system date. During batch execution the system date is always 'today' but the business processing date can be specified using the ProcessingDate parameter of the particular batch job. This gives us the ability to 'spoof' the date that a batch process was run. For example, if a financial batch run was scheduled to run on a Friday but failed. The organization could re-run it on Saturday but set the ProcessingDate to Friday's date. This means that all the calculations etc. would behave as though it was genuinely being run on the Friday. If the ProcessingDate parameter is not specified the business processing date defaults to the system date.

At a technical level, within any transactions initiated by the batch process, calls to *getCurrentDate()* will return the business processing date specified as the batch parameter. Any calls to *getSystemDate()* will return the system date, that is, 'today', as normal.

Output to logs, emails, and reports

When a batch process is being designed, the developer will usually want some sort of output containing summary information, such as the number of records processed, the execution time

and so on. If the information is written to a log file, this file is located in the *EJBServer/buildlogs* directory.

If 'from' and 'to' email addresses were specified during the installation process then an email, containing the same information as the log file above, will be sent to the 'to' email address.

Any reports from the batch process can be found in the same location as the log and will have a .dat file extension.

GenerateInstructionLineItems

GenerateInstructionLineItems is the batch process that produces Instruction Line Items (ILIs) from financial components (FCs). It identifies each financial component of a particular delivery method that reaches its next processing date within the dates specified.

GenerateInstructionLineItems is run directly after the DetermineProductDeliveryEligibility batch process, or at any time after a case is activated.

GenerateInstructionLineItems is run to generate Instruction Line Items that are eventually rolled up by the GenerateInstruments batch process to produce payment instructions or instruments and liability instructions or instruments.

How does GenerateInstructionLineItems work?

The following list outlines the parameters that the batch process looks for when the batch process is submitted to the queue from the application:

1. DeliveryMethod.
2. InstanceID.
3. ProcessingDate.
4. ProcessingDateFrom.
5. ProcessingDateTo.
6. ProductID.

Note: These parameters are not mandatory. If the dates are not entered, the dates default to the system null date. If DeliveryMethod or ProductID is not specified, all FCs are processed regardless of the delivery method and product. Ensure that the InstanceID is specified when you use the batch streaming architecture.

When the batch process runs, it determines the FCs to process based on the specified parameters.

- **DeliveryMethod**
  The following list outlines the FCs that are processed if you do not specify the DeliveryMethod:
  - Liability FCs.
  - Payment FCs.

- Recoupment FCs.
- Liability FCs, that is, FCs of category LBY are processed for the delivery method of Invoice.
- Payment FCs, that is, FCs of category CLM are processed for each delivery method that is listed in the `MethodOfDelivery` code table. Examples include Cash (CSH), Check (CHQ) and EFT (EFT).
- Recoupment FCs, that is, FCs of category RCP are processed for each delivery method that is listed in the `LibMethodOfDelivery` code table. Examples include Giro (GIR) and Invoice (INV).

Note: If the DeliveryMethod is specified as INV, that is, Invoice, then only Liability FCs are processed. Otherwise, Payment and Recoupment FCs are processed for the specified DeliveryMethod as, for example, Cash (CSH), Check (CHQ), Giro (GIR).

- **InstanceID**
  When you run the batch streaming architecture, ensure you specify the InstanceID.
- **ProcessingDate**
  When you run the batch process, ProcessingDate is the business date to use. For more information, see the *Business processing date* related link.
- **ProcessingDateFrom**
  If ProcessingDateFrom is not specified when you submit the batch process, it defaults to the system null date.
  If ProcessingDateFrom was incorrectly set to a future date, it defaults to the system null date when the batch process runs. Otherwise, it uses the specified date.
- **ProcessingDateTo**
  ProcessingDateTo defaults to the business processing date inside the batch process if the parameter defaulted to a null date on submission, that is, no date was specified. The reason ProcessingDateTo is not defaulted to the current date on submission to the queue if not specified because the client current date might not be synchronized with the server current date. In general terms, this can potentially lead to problems when the batch process runs.
  If ProcessingDateTo was set incorrectly to a future date, it defaults to the business processing date when the batch process runs. Otherwise, the date that is specified is used.
- **ProductID**
  If ProductID was not specified, cases for all products are processed.

What can I configure or customize?

You can configure the system to control whether Cúram Express Rules (CER) cases are reassessed before you generate payments. The following steps outline how to configure the system in this way:

1. Log in as a system administrative user.
2. Click **Application** > **Miscellaneous**.
3. Set the ENV_GENERATEINSTRUCTIONLINEITEMS_DONT_REASSESS_CASE application property to **YES**. The default value is **NO**.
4. To publish the property change, click **Publish**.

GenerateInstruments

GenerateInstruments is the batch process that identifies unprocessed instruction line items (ILIs), that is, instruction line items with a status of UNP, and processes each one.

GenerateInstruments is run directly after the GenerateInstructionLineItems batch process.

GenerateInstruments is run to generate the payment instructions or instruments and liability instructions or instruments that create the respective payments and bills that are issued to a participant.

How does GenerateInstruments work?

The following list outlines the parameters the batch process looks for when the batch process is submitted to the queue from the application:

1. InstanceID.
2. ProcessingDate.

Note: The parameters are not mandatory.

- **InstanceID**
  When you run the batch streaming architecture, ensure you specify the InstanceID.
- **ProcessingDate**
  When you run the batch process, ProcessingDate is the business date to use. For more information, see the *Business processing date* related link.
  The following list outlines the order that ILIs are processed when the batch job runs:
  - Product delivery ILIs due, that is, all payment and liability Instruction Line Items.
  - Tax ILIs due, that is, rolls up Instruction Line Items of category TAX to issue payments to the tax authority.
  - Applied deduction ILIs due, that is, Instruction Line Items of category DED that are applied to a liability.
  - Repayment ILIs due, that is, Instruction Line Items of category PRV.
  - Third-party deduction ILIs due, that is, rolls up Instruction Line Items of category DED to issue payments to third parties.

GeneratePayslips

GeneratePayslips is the batch process that identifies payslips that are pending issue. GeneratePayslips is run directly after the GenerateInstruments batch process.

Why is GeneratePayslips run?

`GeneratePayslips` is run to generate the payslips that are sent to the participants to reflect the breakdown of their payments or bills. The breakdown is given at the Instruction Line Item (ILI) level. The following list outlines the different types of payslip:

- Client.
- Case nominee.
- Participant.
- Third party.

How does GeneratePayslips work?

When the batch process is submitted to the queue from the application, it looks for the parameter ProcessingDate. The parameter ProcessingDate is not mandatory.

Use the business date ProcessingDate when you run the batch process. For more information, see the *Business processing date* related link.

When the batch job runs, it processes payslips with a status of pending issue (PS2). As it processes the payslips, the batch job decides what type of payslip it is based on its components' recipient type. The batch job opens an output file for the payslip type and writes all payslips of this type into this file. Output data files for a particular type are opened only after each run of the batch process.

LoadPaymentsReceived

`LoadPaymentsReceived` is the batch process that loads payment received details from an external flat file onto the system for persistent storage.

When the batch process runs, it opens an input file and processes each record listed in the file. The data in this input file is provided in tab delimited format.

The batch process is run whenever the input file becomes available to the organization, for example, from a bank, to reflect payments that have already been received. The data needs to be entered onto the system to reflect these payments.

How does LoadPaymentsReceived work?

When the batch process is submitted to the queue from the application, it looks for the parameters FilePath, FileName, and ProcessingDate.

The parameters FilePath and FileName are mandatory. The parameter FilePath is typically in the form *<Drive>:/Curam/svr/run*, that is, the directory where the input file resides.

The parameter FileName is the full name of the input file, including extension, that contains details of the payments received.

Use the ProcessingDate business date while executing the batch process. For more information, see the *Business processing date* related link. When run, the batch process looks for the FileName specified in the FilePath specified.

Note: When the `curam.participant.enableibanfunctionality` application property is set to **True**, then the batch input file also requires IBAN and BIC values.

IssueConcernPayments

`IssueConcernPayments` is the batch process that issues payments to persons, employers, external parties, information providers, service suppliers, product providers, utilities, and representatives.

`IssueConcernPayments` identifies persons, employers, external parties, information providers, service suppliers, product providers, utilities, and representatives who are due to be paid and issues their respective payments. The payment method and frequency for each of the participant types are held on the respective entities and are set during the registration process.

The times at which the process is run dependents on the next processing dates of the relevant participants. It is conceivable that the process be run daily, as the next processing dates of the participants, that is, persons, employers, external parties, information providers, service suppliers, product providers, utilities, and representatives, might cover the entire week for all types. Typically, the dates are held on the organization calendar.

How does IssueConcernPayments work?

The following list outlines the parameters that the batch process looks for when the batch process is submitted to the queue from the application:

1. ConcernTypeCode.
2. MethodOfPayment.
3. NextPaymentDateFrom.
4. NextPaymentDateTo.
5. ProcessingDate.

Note: The parameters are not mandatory. If the dates are not entered, the dates default to the system null date.

- **ConcernTypeCode**

  If ConcernTypeCode is not specified, all concern role types that are listed in the `ConcernRoleType` code table are processed. The parameter applies to persons, employers, external parties, information providers, service suppliers, product providers, utilities, and representatives. Nothing is picked up for prospect person or prospect employer.

  All records of the specified type are processed even if the type does not exist, that is, the batch process never fails if an incorrect type is specified. Payments are issued only for participants of type person (RL1), employer (RL2), external party (RL17), information provider (RL5), service supplier (RL3), product provider (RL4), utility (RL6), or representative (RL13).

- **MethodOfPayment**
  If MethodOfPayment is not specified, all delivery methods that are listed in the `MethodOfDelivery` code table are processed. Otherwise, processing is performed for the method of payment that is specified only.
- **NextPaymentDateFrom and NextPaymentDateTo**
  If NextPaymentDateFrom or NextPaymentDateTo is not specified when submitting the batch process, they default to the system null date.
  NextPaymentDateFromNextPaymentDateTo are not defaulted to the current date on submission to the queue because if the user does not specify them the client date might not be synchronized with the server current date. In general terms, this can potentially lead to problems when batch processes run.
  Inside the batch process, two processing date parameters, dateFrom and dateTo, are set based on the values of NextPaymentDateFrom and NextPaymentDateTo. The following list outlines the checks that are performed:
  - If both NextPaymentDateFrom and NextPaymentDateTo are null dates, that is, not specified, dateFrom and dateTo are set to the current system date.
  - If NextPaymentDateFrom was specified and NextPaymentDateTo was not, both dateFrom and dateTo are set to the NextPaymentDateFrom value.
  - Similarly, if NextPaymentDateTo was specified and NextPaymentDateFrom was not, both dateFrom and dateTo are set to the NextPaymentDateTo value.
  - If NextPaymentDateFrom and NextPaymentDateTo are specified, dateFrom is set to the NextPaymentDateFrom value with dateTo being set to NextPaymentDateTo.
- **ProcessingDate**
  When you run the batch process, use the business date ProcessingDate. For more information, see the *Business processing date* related link.

ExpirePayments

`ExpirePayments` is the batch process that expires payments that are not cashed after a certain length of time.

`ExpirePayments` identifies payment instruments on the system that have a `reconcilStatusCode` of 'issued' (ISS) and expires them if they are on the system for a certain length of time.

The purpose of `ExpirePayments` is to expire payments of a specified delivery method. Typically, these are of type check (CHQ) that were not cashed after some days. Checks usually have a six-month lifespan and cannot be cashed after this. The organization wants to run the batch process to expire the necessary payments.

How does ExpirePayments work?

The following list outlines the parameters that the batch process looks for when the batch process is submitted to the queue from the application:

- DeliveryMethod.

- ExpiryPeriod.
- ProcessingDate.

Enter the `ExpiryPeriod` in days, that is, the number of days the payments must be on the system to be picked up by the batch process. The ProcessingDate is the business date to use when you run the batch process. For more information, see the *Business processing date* related link.

ProcessPaymentInstrumentTypes

`ProcessPaymentInstrumentTypes` is the batch job that processes all payment instrument records that are due for issue and writes their details into an output file.

`ProcessPaymentInstrumentTypes` identifies the payment instruments that are to be issued, populates an output file with these instruments' details and updates their status to issued (ISS). Running this program and creating the output file is the equivalent of issuing payments. For example, if the batch process is run for DeliveryMethodType of EFT, the output file is sent to the bank where the payments are transferred to the participant accounts.

Run `ProcessPaymentInstrumentTypes` to provide a list of payments due to the financial institution or institutions that provide the payments to the participant or participants.

How does ProcessPaymentInstrumentTypes work?

The following list outlines the parameters that the batch process looks for when the batch process is submitted to the queue from the application:

- DeliveryMethodType.
- ProcessingDate.

The parameters are not mandatory. If the delivery method type is not provided, all delivery method types are processed. Otherwise, the one provided is processed. Use the ProcessingDate is the business date to use when you run the batch process. For more information, see the *Business processing date* related link.

Payment reconciliation

Payment reconciliation is the batch process that reconciles an account by comparing what was due to be paid with the amount paid, reporting any discrepancies.

When the batch process runs, it compares the intended amount to be paid to a participant against the amount paid. The input file to the process contains details of the payments that are received by the participants. The file comes from the institution that made the payments, for example, a bank. The payments in this flat file are compared to the payments issued by the system. Any discrepancies found are generated in a report.

Run the payment reconciliation batch to identify discrepancies, if there are any, in amounts that are paid against the intended amounts to pay.

How does payment reconciliation work?

When the batch process is submitted to the queue from the application, it looks for the parameters FilePath, FileName, and ProcessingDate.

The parameters FilePath and FileName are mandatory. The parameter FilePath is typically in the form `<Drive>:/Curam/svr/run`, that is, the directory where the input file resides.

The parameter FileName is the full name of the input file, including extension, that contains details of the payments received.

Use the ProcessingDate business date while you run the batch process. For more information, see the *Business processing date* related link. When run, the batch process looks for the FileName specified in the FilePath specified.

GeneralLedgerInterface

`GeneralLedgerInterface` is the batch process that gathers financial transactions for a specified date, or date range, and exports them from the application into an output file that can then be imported into the general ledger.

The output file contains details of financial transactions at the lowest level of granularity. In the application, financial transactions at this level are defined as instruction line items. The output file is a sample of what is sent to a third party who then uses the file to update the general ledger.

The `GeneralLedgerInterface` is run whenever the organization wants to create a financial transaction output file that can be imported into the general ledger.

The `GeneralLedgerInterface` is run to provide an output file of credit and debit transactions for a specified date, or date range, that can be imported into a general ledger interface.

How does GeneralLedgerInterface work?

WThe following list outlines the parameters that the batch process looks for when the batch process is submitted to the queue from the application:

1. CreationDateSearchInd.
2. DateFrom.
3. DateTo.
4. ProcessingDate.

Note: The parameters are not mandatory. If the dates are not entered, they default to the system null date.

- **CreationDateSearchInd**
  CreationDateSearchInd indicates whether the extract is based on the creation date or effective date. If CreationDateSearchInd was set to True by the user, a creation date range search is performed. Otherwise, an effective date range search is used.

When specifying the value for CreationDateRangeInd, it must be either True or False. It cannot be 1 or 0, or Yes or No. Validation is performed on the client to prevent entries of this type.

- **DateFrom and DateTo**
  The user does not specify DateFrom or DateTo when submitting the batch process. The parameters default to the system null date.
  DateFrom and DateTo are not defaulted to the current date on submission to the queue if the user does not specify them because the client current date might not be synchronized with the server current date. In general terms, this can potentially lead to problems when batch processes run.
  The following list outlines the date checks that are performed inside the batch process:
  - If DateFrom is found to be a null date, that is, not specified by a user, produces an error and the batch process fails.
  - Similarly, if DateTo is found to be a null date, that is, not specified by a user, produces an error and the batch process fails.
- **ProcessingDate**
  Use the ProcessingDate business date while you run the batch process. For more information, see the *Business processing date* related link.

ReconcileCaseAccount

`ReconcileCaseAccount` is run to reconcile liability cases where a liability overpayment or liability underpayment occurs. It works only for liability over billing and liability under billing. It does not reconcile benefit overpayment or payment corrections.

Use `ReconcileCaseAccount` to balance a single case account. During the lifetime of a liability case, liability transactions might be added to the case. Over allocation might occur due to one or more of these for reserving the money for this case. So, the reconcile case batch process does take the over allocations and see whether they can be allocated to another liability transactions on the same case. When all liabilities are paid, if any money remains it is made available for allocation to other cases. However, the allocation to other cases is not automated.

The following list outlines the batch process performs two separates pieces of processing:

- It reconciles all liability cases where an underpayment is applied.
- It also reconciles all liability cases where an overpayment is applied.

Run the `ReconcileCaseAccount` to reconcile both overpayment and underpayment liability cases.

How does ReconcileCaseAccount work?

When the batch process is submitted to the queue from the application, it looks for the ProcessingDate.

> **Note:** The parameter is not mandatory.

- **ProcessingDate**
  Use the ProcessingDate business date while you run the batch process. For more information, see the *Business processing date* related link.
  The process reconciles both overpayment and underpayment liability cases. The following list outlines the order in which liability cases are performed:

  1. Overpayment liability cases.
  2. Underpayment liability cases.

  The processing for reconciling overpaid liability cases retrieves the list of instruction line items for the overpaid liability case. If the instruction line items represent an overpayment and have a status of processed, the overallocation is converted into an unallocated payment received record. If the case the overpayment relates to has any outstanding liabilities, then the newly created payment received instruction is allocated against them.
  The processing for reconciling underpaid liability cases retrieves the list of instruction line items for the underpaid liability case. If the instruction line items represent an underpayment and have a status of processed, an adjustment line item record is created that is allocated against outstanding liability instruction line items.

1.5 Funded Program Management batch processes

A list of the batch process that provide Funded Program Management functions in the application.

You can typically configure these batch processes in the system administration application.

Unless specified otherwise, you can run batch processes in any order.

ModifyFundFiscalYearStatus

This is the batch process that modifies the status of all the fund fiscal years to INACTIVE for the fiscal years whose end date is before the batch processing date. If the processing date is not entered by the user, the current system date will be used as the processing date. This batch should be run daily in order to make sure that the fund fiscal year status is accurate.

Batch Process Class and Method The class and method for this batch process is `ModifyFundFiscalYearStatus.modifyFundFiscalYearStatus`.

1.6 BatchUploadICDCodes

BatchUploadICDCodes is used to batch upload the ICD Code files that contain the classification of the medical conditions into the ICDCODE and ICDCODEVERSION tables.

BatchUploadICDCodes initiates the following processing steps:

1. Creates the ICD code version entry for the input details.
2. Parses the ICD code file content and stores the data in the ICDCode table.

Class and method

ICDCodes.batchUploadICDCodes

Parameters

Parameter	Description
classification	ICD code category
fileName	Name of the ICD codes file to be read
filePath	Full path of the ICD codes file to be read
processingDate	Processing date of the batch
versionNumber	Version number to be associated with the ICD code
volume	Volume of the ICD code version

1.7 CitizenWorkspacePurgeNonReferencedDataProcess

The CitizenWorkspacePurgeNonReferencedDataProcess batch process removes all data that is 'unreachable' by the client regardless of whether the client is a generated, standard, or linked user.

Data can become unreachable if, for example, a browser window is closed abruptly and an in progress application or modal is deleted without being submitted.

Execute the CitizenWorkspacePurgeNonReferencedDataProcess batch process while the system is offline. The CitizenWorkspacePurgeNonReferencedDataProcess batch process removes the following data from the database:

1. CitizenScriptInfos in the OPEN state (CSS01)
2. CitizenScriptInfos in the CANCELLED state (CSS04)
3. IEG Executions associated with those CitizenScriptInfos
4. Data stores associated with the CitizenScriptInfos provided those data stores aren't associated with any other CitizenScriptInfos, for example, IN-PROGRESS scripts (CSS02).

Class and method

CitizenWorkspacePurgeNonReferencedDataProcess.process

1.8 CitizenWorkspacePurgeNonReferencedMotivationsProcess

The CitizenWorkspacePurgeNonReferencedMotivationsProcess batch process removes all motivations data that is inaccessible by the client, regardless of whether the client is a generated, standard, or linked user.

Data can become inaccessible by the client if, for example, a browser window is closed abruptly and an in progress application or modal is deleted without being submitted.

Run the CitizenWorkspacePurgeNonReferencedMotivationsProcess batch process while the system is offline. The CitizenWorkspacePurgeNonReferencedMotivationsProcess batch process removes the following data from the database:

- Motivations with a state of STARTED or CANCELLED unless the motivation is associated with a specific IntakeApplication through an entry in the MotivationIntakeAppLink table
- IEGExecutionState records that are related to the previous motivation
- MotivationResult records that are associated with the previous motivation
- Linked string records that are associated with the previous MotivationResult
- Data stores that are associated with the previous motivation if they are not referenced by another Motivation, IntakeApplication or CitizenScriptInfo, for example, IN-PROGRESS scripts.

Class and method

CitizenWorkspacePurgeNonReferencedMotivationsProcess.process

1.9 CitizenWorkspacePurgeOldDataProcess

The CitizenWorkspacePurgeOldDataProcess batch process is a direct replacement for the CitizenPortalPurgeData batch job. Its purpose is to provide equivalent functionality in a way that is more scalable.

Execute the batch process while the system is offline. The CitizenWorkspacePurgeOldDataProcess batch process deletes the following data from the database:

1. CitizenScriptInfo entities older than the purge date in the SUBMITTED state (CSS03) associated with IntakeApplications (OST03, OST04)
2. CitizenScriptInfo entities older than the purge date in the CANCELLED state (CSS04)
3. IEG Execution State entities associated with CitizenScriptInfo.
4. Data stores associated with CitizenScriptInfo provided they are not associated with an IntakeApplication.

Class and method

CitizenWorkspacePurgeOldDataProcess.process

Parameters

Parameter	Description	Default value
purgeDate	purge date in the format YYYYMMDD. For example, entering 20131204 will remove data before the date 2013-12-04.	N/A

1.10 CitizenWorkspacePurgeDataProcess

The CitizenWorkspacePurgeDataProcess batch process removes all Citizen Workspace related transient data from the system.

Transient data is data that is created by users who are not registered on the system, for example, users who do not have an account.

Execute the batch process while the system is offline. The CitizenWorkspacePurgeDataProcess batch process removes the following transient data from the system:

1. All generated users in the ExternalUser user table
2. CitizenScriptInfo entities that belong to the generated users
3. IEGExecutionState entities that belong to the users
4. All DataStore entities belonging to generated users provided those data store entities are not associated with a submitted application.
5. All entries in the PagePlayerState table.
6. All entries in the TriageResult table except for TriageResults that are associated with submitted Life Events.

Class and method

CitizenWorkspacePurgeDataProcess.process

1.11 CreateRosters

This batch process generates rosters for service offerings.

CreateRoster initiates the following processing steps:

1. Checks every active service offering to see whether a service is required and creates a roster where required.
2. Sets date range for the roster based on the configured roster generation frequency for the service offering. The first date is set as the processing date.
3. Retrieves all provider offerings for a service offering and date range to determine whether a roster exists.
   - If a roster does not exist, a roster is created.
   - If a roster exists with a different date range, the roster is updated with the new date range.

Class and method

CreateRosters.process

Dependencies

You must run the CreateRoster and ProcessSALI batch processes in the following order:

1. CreateRoster
2. ProcessSALI

This sequence ensures that SALIs created by ProcessSALI are based on the latest service offering rosters.

Parameters

Description of the parameters that are used to run the batch process

Parameter	Description	Default value
Batch Processing Date	The date the batch program is processed.	n/a

CreateRostersStream uses batch streaming to distribute the processing load of CreateRosters.

Related reference

ProcessSALI on page 31
ProcessSALI processes service authorization line items to add them to rosters.

1.12 DiscrepancyBatchProcess

This batch process compares scheduled versus actual attended hours for cash assistance clients for all active outcome plans to determine whether the client has met the required participation.

1.13 DynamicEvidenceConfigurationExtractor

This batch process extracts dynamic evidence configuration artefacts.

Parameters

Parameter	Description	Default value
Codetable Directory	Name of the directory where codetable (CTX) files are written.	codetable
Component	The name of the destination component where the extractor output is written, for example, custom.	n/a
Data Manager Directory	Name of the directory where datamanger (DMX, blob, clob) files will be written.	data
DMX Directory	Name of the directory where DMX files will be written.	initial
Evidence Type Code Prefix	Specifies that only dynamic evidence types which evidence type codes begin with this prefix (e.g. DET) will be extracted.	n/a
Evidence Type Extraction List	Lists dynamic evidence type logical names to specify which dynamic evidence types will be extracted.	n/a

Parameter	Description	Default value
Extract All	A value of true specifies that all dynamic evidence type configurations on the system will be extracted.	n/a
Extract Case Links	A value of true specifies that dynamic evidence links to cases other than products and integrated cases will be extracted.	n/a
Extract IC Link	A value of true specifies that dynamic evidence links to integrated cases will be extracted.	n/a
Extract Product Links	A value of true specifies that dynamic evidence links to products will be extracted	n/a
Lower Key	The lower key in the replacement primary key range.	n/a
Preserve Range Keys	The list of range keys that should not be changed.	n/a
Server Directory	The location of the server folder within the destination component.	EJBServer
Tab Directory	Name of the directory where section configuration (SEC) files are written.	tab
Upper Key	The upper key in the replacement primary key range.	n/a

1.14 ExportUsersToLDIF

This batch process exports the contents of the active internal users defined in Cúram to an LDIF file. Once the export from Cúram is complete, this file can be directly imported into an LDAP server.

ExportUsersToLDIF initiates the following processing steps:

1. Search the USERS table for all active users (Users.searchActive).
2. Write the user details to an LDIF file.

This batch job should be executed while the system is offline.

Note: The organization structure is not included in the export. The export provides a flat listing of users in the LDIF file.

Class and method

ExportUsersToLDIF.export

Parameters

Parameter	Description	Default value
File Name	The full name of the file.	n/a
File Path	The file path of the file.	n/a
Processing Date	The date the batch program will be processed.	n/a

1.15 GenerateRostersWithDate

This batch process generates rosters and roster line items for a specific date for provider services for which service authorization exist.

GenerateRostersWithDate initiates the following processing steps:

1. Generates an attendance roster up to the specified date for each provider service for which service authorizations exists.
2. Generates a roster line item for every client authorized in the service authorization line item for the period of the roster.
3. Generates a roster for all the open or in-progress service authorization line items for the provider.

   Note: A roster is not generated if a roster already exists during the frequency period calculated from the processing date for the particular provider service.
4. Adds clients authorized for the provider service to the new roster as a roster line item. If an open roster line item entry already exists, the entry is extended to accommodate the new authorization values.

Class and Method

GenerateRostersWithDate.generateRostersWithDate

Parameters

Description of the parameters used to run the batch process.

Parameter name	Description	Default value
Processing Date	The date the batch program will be processed.	n/a

1.16 IssuePaperRosters

This batch process processes service offerings to issue blank paper rosters for all provider offerings for which a paper roster is required.

IssuePaperRosters initiates the following processing steps:

1. Checks all provider offerings for a given service offering to see if there is a need to issue a paper roster for the associated rosters.
2. Generates a paper roster for every service offering where one is required.

The date range for the paper roster is calculated based on the configured roster generation frequency value and the processing date.

When a paper roster is generated for a roster, events are raised. These events can be used to automate the issuance of the paper roster to the provider.

Class and method

IssuePaperRosters.process

Parameters

Description of the parameters used to run the batch process.

Parameter name	Description	Default value
Batch Processing Date	The date the batch program is processed.	n/a

IssuePaperRostersStream uses batch streaming to distribute the processing load of IssuePaperRosters.

1.17 MeetingManagement

This batch process processes responses returned to the configured email account and updates the meeting attendee response data accordingly. The MeetingManagement batch process connects to a configured email account and checks for meeting responses. If responses are found, attendee response details are updated and the email is removed from the account inbox.

Processing steps

Meeting Management initiates the following processing steps:

1. Connects to the email account using the username, password and host set by the following the environment properties:
    1. ENV_MEETING_REQUEST_REPLY_USERNAME
    2. ENV_MEETING_REQUEST_REPLY_PASSWORD
    3. ENV_MEETING_REQUEST_REPLY_HOST
2. Opens the mail folder.
3. Processes all the meeting response messages and updates the appropriate meeting attendee response data. The message must contain a body part of type text/calendar, the calendar body part must comply with the RFC 2445 standard for calendar data exchange and contain a UID that matches the ID of a Cúram activity. An exception is thrown if an invalid response method is received or if a message processing exception is encountered.

Class and method

MeetingManagement.processMeetingResponses

Parameters

MeetingManagement has no input parameters.

1.18 ModifyFundFiscalYearStatus

This batch process sets the status of all fund fiscal years to 'Inactive' for the fiscal years whose end date is before the batch processing date. If the processing date is not entered by the user, the current system date is used as the processing date. This batch should be run daily in order to make sure that the fund fiscal year status is accurate.

ModifyFundFiscalYearStatus initiates the following processing steps:

1. Checks if the batch processing is set, if it is not, the current date is used.
2. Searches all the active fund fiscal years for which the end date is before the given processing date.
3. Modifies all the business status of the returned results to INACTIVE state.

Class and method

ModifyFundFiscalYearStatus.modifyFundFiscalYearStatus

Parameters

Parameter	Description
Batch Processing Date	The date the batch program is processed.

1.19 PDMIdentifyCases

This batch process assists customers in meeting federal regulations for annual renewal processing and notice generation. The process ensures that the projected eligibility and notice generation processes consider all HCR cases.

For non-annual renewals periodic data matching runs, inserting evidence by using the PDMEvidenceMaintanence API ensures that the case is processed by the projected eligibility and notice generation processes.

Class and method

curam.hcr.pdm.sl.intf.PDMIdentifyCases.process

Parameters

Parameter	Description	Default value
runID	Mandatory. The run ID value that you specified in the periodic data matching run configuration in the Administration application.	A different value is required for each run.

Parameter	Description	Default value
instanceID	Optional. An optional instance ID that used by the batch infrastructure to stop or restart a batch process.	'BPN26006'
processingDate	Optional.	The current date.

1.20 ProcessDefinitionImportDirectory

This batch process imports process definitions from a specified directory.

Parameters

Parameter	Description	Default value
Directory Name	The full path of the directory containing the process definitions to import	n/a
Overwrite Existing Process Definitions	Set this indicator to true to overwrite existing process definitions on the database with those being imported.	true

1.21 ProcessDefinitionImportFile

This batch process imports a process definition from a specified file.

Parameters

Parameter	Description	Default value
File Name	The full path of the file containing the process definition to import.	n/a
Overwrite Existing Process Definition	Set this indicator to true to overwrite the existing process definition on the database with the one being imported.	n/a

1.22 ProcessRenewalData

This batch process is used to run the projected eligibility and notice generation processes for a specified list of cases as part of periodic data matching or annual renewals.

For periodic data matching runs, inserting evidence by using the PDMEvidenceMaintanence API automatically ensures that the case is processed by periodic data matching projected eligibility and notice generation processes. For annual renewals, there can also be cases for which no evidence was received, but still must be processed as part of an annual renewal.

Class and method

curam.hcr.pdm.sl.intf.ProcessRenewalData.process

Parameters

Parameter	Description	Default value
runID	Mandatory. The run ID value that you specified in the periodic data matching run configuration in the Cúram Administration application.	A different value is required for each run.
instanceID	Optional. An optional instance ID that used by the batch infrastructure to stop or restart a batch process.	'BPN26006'
processingDate	Optional.	The current date.

1.23 Process Service Authorization Line Item

This batch process processes service authorization line items to add them to rosters.

Process Service Authorization Line Item (ProcessSALI) initiates the following processing steps:

1. Generates all service authorization line items for which a matching service authorization exists for each case participant role on the processing date.
2. Obtains the date range for retrieval of the service authorization line items from the service offering and processing date.
3. Adds the service authorization line items to the correct roster as created by the CreateRosters batch process.

Note: The ProcessSALI batch process should always be executed immediately after the CreateRosters batch to ensure that the latest rosters are used to create the service authorization line items.

Class and method

ProcessSALI.process

Parameters

Table 4: Batch Parameters

Description of the parameters used to run the batch process.

Parameter name	Description	Default value
Batch Processing Date	The date the batch program will be processed. This is an optional parameter. If processing date is not specified, the current date is used.	n/a

Parameter name	Description	Default value
Service Offering ID	The service offering ID.	n/a

ProcessSALIStream uses batch streaming to distribute the processing load of ProcessSALI.

Related reference

CreateRosters on page 30

Should be executed before the ProcessSALI batch process to ensure the latest rosters are used to generate service line items.

1.24 Purge Non-Referenced Old Data

Purge Non-Referenced Old Data is a configurable batch process that removes data records that are older than a specified date. The batch process applies to all client types, including generated, standard, and linked user clients.

Purged data

The Purge Non-Referenced Old Data batch process purges the following entity records based on the values that you specify for the scriptType, scriptStatus, and purgeData parameters:

- CitizenScriptInfo
- Datastore Entity
- Datastore Entity role
- LifeEvent
- LifeEventRmtLink
- IEGExecutionState
- TriageResult

Executing the batch process

Because the batch process executes a data purge that is irreversible, it is advisable to back up your data first.

To execute the batch process from the command line, enter a command to run the `curam.citizenworkspace.facade.intf.CitizenScriptInfoPurgeBatchProcess.process` process while the system is offline. The following table shows the process parameters:

Table 5: `CitizenScriptInfoPurgeBatchProcess.process` process parameters

Process parameter	Description	Default value
purgeDate	Mandatory purge date in the format YYYYMMDD, where the purge date specifies the date that data was last modified. For example, enter 20131204 to remove data before the date 2013-12-04.	Not applicable. You must specify a purge date.

Process parameter	Description	Default value
scriptStatus	For valid code values, see the `CT_CitizenScriptStatus` code table.	If you do not specify a valid `CT_CitizenScriptStatus` code table value, the scriptStatus parameter is not applied.
scriptType	For valid code values, see the `CT_OnlineScriptType` code table.	If you do not specify a valid `CT_OnlineScriptType` code table value, the scriptType parameter is not applied.

Select the **OK** button.

The following example shows a sample Ant command for running the batch process:

```
ant -f $CURAMSDEJ/bin/app_batchlauncher.xml
-
Dbatch.program=curam.citizenworkspace.facade.intf.CitizenScriptInfoPurgeBatchProcess.process
-Dbatch.parameters="purgeDate=20170216"
```

The corresponding stream batch process is `CitizenScriptInfoPurgeBatchProcessStream.process`. The following example shows a sample Ant command for running the stream batch process:

```
ant -f $CURAMSDEJ/bin/app_batchlauncher.xml
-
Dbatch.program=curam.citizenworkspace.facade.intf.CitizenScriptInfoPurgeBatchProcessStream.process
```

Alternatively, execute the batch process through the application user interface while the system is offline by using the following steps:

1. Log on to Cúram as a system administrative user, for example, `sysadmin`.
2. Click **System Configurations**.
3. In the Shortcuts panel, click **Batch** > **Processes**.
4. Search for the `Purge Non-Referenced Old Data` batch process.
5. Specify values for the scriptType, scriptStatus, and purgeData parameters. You must specify valid values for scriptType and scriptStatus for the parameters to be applied. For information about the parameters, see the previous `CitizenScriptInfoPurgeBatchProcess.process` process parameters table.
6. Execute the batch process.

Process results

The following table outlines the results of running the Purge Non-Referenced Old Data batch process, based on the parameter values that you specify.

Table 6: Results of running the Purge Non-Referenced Old Data process based on specified parameter values

purgeDate	scriptStatus	scriptType	Process result
Purge date in the format YYYYMMDD	Not specified or invalid value	Not specified or invalid value	Removes all records based on the value of purgeDate, regardless of the values of scriptStatus and scriptType.
Purge date in the format YYYYMMDD	Valid `CT_CitizenScriptStatus` code table value	Not specified or invalid value	Removes all records based on the values of purgeDate and scriptStatus, regardless of the value of scriptType.
Purge date in the format YYYYMMDD	Valid `CT_CitizenScriptStatus` code table value	Valid `CT_OnlineScriptType` code table value	Removes all records based on the values of purgeData, scriptType, and scriptStatus.
Purge date in the format YYYYMMDD	Not specified or invalid value	Valid `CT_OnlineScriptType` code table value	Removes all records based on the values of purgeData and scriptType, regardless of the value of scriptStatus.

Viewing the process results

When you execute the batch process, the purge batch process generates a report that displays information about the processed records.

Customizing the Purge Non-Referenced Old Data batch process

The Purge Non-Referenced Old Data batch process purges only the entities that are listed in the previous *Purged data* section. The following methods are available in a default installation for customizing the Purge Non-Referenced Old Data batch process:

- Use the following method in customized scenarios for any type of preremoval processing for purging records. Enter the code on one line.

```
curam.citizenworkspace.facade.impl.CitizenScriptInfoPurgeBatchHook
  .preCitizenScriptInfoPurgeBatchRecords(CitizenScriptInfoPurgeBatchDetails)
```

- Use the following method in customized scenarios for any type of post removal processing for purging records. Enter the code on one line.

```
curam.citizenworkspace.facade.impl.CitizenScriptInfoPurgeBatchHook
  .postCitizenScriptInfoPurgeBatchRecords(CitizenScriptInfoPurgeBatchDetails)
```

For example, you can create custom implementations of the previous methods to purge more custom entities that are related to citizen script information.

1.25 Redetermine Translator

This batch is used to perform automatic redetermine translator processing when a change to the language skill of a user is made and the change will affect a large volumes of cases.

RedetermineTranslator initiates the following processing steps:

1. Checks all open cases for which a particular user is assigned as the case owner and compares the preferred language of each case participant in the case against the language skill of the user.
2. Updates the translator required indicator for the case participant if necessary.

Class and method

RedetermineTranslator.process

RedetermineTranslatorStream.process

Parameters

Table 7: Batch Parameters

Description of the parameters used to run the batch process.

Parameter name	Description
Instance Identifier	Instance ID. Allows multiple instances of the same batch process to run at the same time.
Processing Date	The date the batch program will be processed.
User Name	Used to specify the case owner whose language skills will be assessed.

Related Information:

The curam.cases.maxnocases.onlineautotranslatordetermination application property controls whether automatic re-determination of translator requirements will occur in batch mode or singly via online processing. If the number of open cases that require processing exceeds the value specified in the application property, re-determination will not occur in online mode, and must be executed in batch instead.

1.26 ScanMilestoneDeliveryStartDateBatch

Use the ScanMilestoneDeliveryStartDateBatch process to monitor the expected start date and the status of milestone deliveries on the system and to raise the configured event when the milestone delivery is not started.

About this task

Complete the steps to run the ScanMilestoneDeliveryStartDateBatch process.

Procedure

1. Optional: Turn notifications on or off for each milestone delivery record that fails to raise the expected start date event due to an error. To generate a notification for each skipped milestone delivery record, set the `curam.batch.scanmilestonedeliverystartdatebatch.notifyonskip` environment property to **YES**.
2. Optional: Configure the batch process by applying the environment variables that are listed in the table to the appropriate values for the environment in which the batch process is running.

   Table 8: List of environment variables to apply.

Application property	Description	Default value
`curam.batch.scanmilestonedeliverystartdatebatch.chunksize`	The property specifies the number of records to process in each chunk.	5
`curam.batch.scanmilestonedeliverystartdatebatch.dontrunstream`	The property specifies whether the batch process sleeps while it is waiting for the processing to complete.	**No**
`curam.batch.scanmilestonedeliverystartdatebatch.chunkkeywaitinterval`	The property specifies the interval in milliseconds that the batch process waits before it retries to read the chunk key table.	1000
`curam.batch.scanmilestonedeliverystartdatebatch.unprocessedchunkwaitinterval`	The property specifies the interval in milliseconds that the batch process waits before it retries to read the chunk table for unprocessed chunks.	1000
`curam.batch.scanmilestonedeliverystartdatebatch.processunprocessedchunk`	The property specifies whether the batch process attempts to implement any unprocessed chunks that are found after all streams are completed.	**No**

3. Run the batch process.

   To ensure that you use the database to its full capacity, use the ScanMilestoneDeliveryStartDateBatchStream. The batch stream process supports running in multiple streams to allow for concurrent execution on one or more computers. For more information about the batch stream process, see the *ScanMilestoneDeliveryStartDateBatchStream* related link.

Related concepts

ScanMilestoneDeliveryStartDateBatchStream on page 67
The ScanMilestoneDeliveryStartDateBatchStream process supports batch streaming to scan the milestone delivery start date. The batch process can run only with the ScanMilestoneDeliveryStartDateBatch process.

ScanMilestoneDeliveryStartDateBatchStream

The ScanMilestoneDeliveryStartDateBatchStream process supports batch streaming to scan the milestone delivery start date. The batch process can run only with the ScanMilestoneDeliveryStartDateBatch process.

Class and method for ScanMilestoneDeliveryStartDateBatch

The class and method for the batch process is `curam.core.sl.impl.ScanMilestoneDeliveryStartDateBatch.process`.

Parameters for ScanMilestoneDeliveryStartDateBatch

The following table lists the parameters for the batch process.

Table 9: Parameters for the batch process.

Parameter	Description	Default value
Batch process instance ID	The unique identifier that allows multiple instances of the same batch process to run at the same time effectively. When no instance ID is specified, only one instance of the batch process can run.	Not applicable.
Processing date	The dates that are used in comparison against the milestone delivery when the batch process determines whether the expected start date is reached.	Current system date.

Starting a stream for an instance of the ScanMilestoneDeliveryStartDateBatch process

To start a stream for an instance of the ScanMilestoneDeliveryStartDateBatch process, use the Batch Process Instance ID parameter to link the batch process, or multiple stream batch processes, to the particular batch process instance.

For example, where the Batch Process Instance ID is batch_reassessment_1 for an instance of the ScanMilestoneDeliveryStartDateBatch process, you must also set the Batch Process Instance ID parameter for the ScanMilestoneDeliveryStartDateBatchStream batch process, or multiple stream batch processes, to batch_reassessment_1.

No limit applies to the number of ScanMilestoneDeliveryStartDateBatchStream processes that you can link to the same instance of the ScanMilestoneDeliveryStartDateBatch process.

Class and method for ScanMilestoneDeliveryStartDateBatchStream

The class and method for the batch process is `curam.core.sl.impl.ScanMilestoneDeliveryStartDateBatchStream.process`.

Parameters for ScanMilestoneDeliveryStartDateBatchStream

The following table lists the parameters for the batch process.

Table 10: Parameters for the batch process.

Parameter	Description	Default value
Batch process instance ID	The unique identifier that allows multiple instances of the same batch process to run at the same time effectively. When no instance ID is specified, only one instance of the batch process can run.	Not applicable.

1.27 ScanMilestoneDeliveryEndDateBatch

Use the ScanMilestoneDeliveryEndDateBatch process to monitor the expected end date and the status of milestone deliveries on the system and to raise the configured event when the milestone delivery is not completed.

About this task

Complete the steps to run the ScanMilestoneDeliveryEndDateBatch process.

Procedure

1. Optional: Turn notifications on or off for each milestone delivery record that fails to raise the expected end date event due to an error. To generate a notification for each skipped milestone delivery record, set the `curam.batch.scanmilestonedeliveryenddatebatch.notifyonskip` environment property to **YES**.
2. Optional: Configure the batch process by applying the environment variables that are listed in the table to the appropriate values for the environment in which the batch process is running.

Table 11: List of environment variables to apply.

Application property	Description	Default value
`curam.batch.scanmilestonedeliveryenddatebatch.chunksize`	The property specifies the number of records to process in each chunk.	5
`curam.batch.scanmilestonedeliveryenddatebatch.dontrunstream`	The property specifies whether the batch process sleeps while it is waiting for the processing to complete.	**No**
`curam.batch.scanmilestonedeliveryenddatebatch.chunkkeywaitinterval`	The property specifies the interval in milliseconds that the batch process waits before it retries to read the chunk key table.	1000
`curam.batch.scanmilestonedeliveryenddatebatch.unprocessedchunkwaitinterval`	The property specifies the interval in milliseconds that the batch process waits before it retries to read the chunk table for unprocessed chunks.	1000

Application property	Description	Default value
curam.batch.streamprocessunprocessedchunk	The property specifies whether the batch process attempts to process any unprocessed chunks that are found after all streams are completed.	No

3. Run the batch process.

   To ensure that you use the database to its full capacity, use the ScanMilestoneDeliveryEndDateBatchStream. The batch stream process supports running in multiple streams to allow for concurrent execution on one or more computers. For more information about the batch stream process, see the *ScanMilestoneDeliveryEndDateBatchStream* related link.

Related concepts

ScanMilestoneDeliveryEndDateBatchStream on page 69
The ScanMilestoneDeliveryEndDateBatchStream process supports batch streaming to scan the milestone delivery end date. The batch process can run only with the ScanMilestoneDeliveryEndDateBatch process.

ScanMilestoneDeliveryEndDateBatchStream

The ScanMilestoneDeliveryEndDateBatchStream process supports batch streaming to scan the milestone delivery end date. The batch process can run only with the ScanMilestoneDeliveryEndDateBatch process.

Class and method for ScanMilestoneDeliveryEndDateBatch

The class and method for the batch process is `curam.core.sl.impl.ScanMilestoneDeliveryEndDateBatch.process`.

Parameters for ScanMilestoneDeliveryEndDateBatch

The following table lists the parameters for the batch process.

Table 12: Parameters for the batch process.

Parameter	Description	Default value
Batch process instance ID	The unique identifier that allows multiple instances of the same batch process to run at the same time effectively. When no instance ID is specified, only one instance of the batch process can run.	Not applicable.
Processing date	The dates that are used in comparison against milestone delivery when the batch process determines whether the expected end date is reached.	Current system date.

Starting a stream for an instance of the ScanMilestoneDeliveryEndDateBatch process

To start a stream for an instance of the ScanMilestoneDeliveryEndDateBatch process, use the Batch Process Instance ID parameter to link the batch process, or multiple stream batch processes, to the particular batch process instance.

For example, where the Batch Process Instance ID is batch_reassessment_1 for an instance of the ScanMilestoneDeliveryEndDateBatch process, you must also set the Batch Process Instance ID parameter for the ScanMilestoneDeliveryEndDateBatchStream batch process, or multiple stream batch processes, to batch_reassessment_1.

No limit applies to the number of ScanMilestoneDeliveryEndDateBatchStream processes that you can link to the same instance of the ScanMilestoneDeliveryEndDateBatch process.

Class and method for ScanMilestoneDeliveryEndDateBatchStream

The class and method for the batch process is `curam.core.sl.impl.ScanMilestoneDeliveryEndDateBatchStream.process`.

Parameters for ScanMilestoneDeliveryEndDateBatchStream

The following table lists the parameters for the batch process.

Table 13: Parameters for the batch process.

Parameter	Description	Default value
Batch process instance ID	The unique identifier that allows multiple instances of the same batch process to run at the same time effectively. When no instance ID is specified, only one instance of the batch process can run.	Not applicable.

1.28 StartEffectiveOutcomePlanActions

This batch process starts actions that have not already been started that have a start date in the past or a future-dated end date.

1.29 StartEffectiveServiceDeliveries

Checks if any of the service delivery records with a state of 'Not Started' have a cover period start date which is effective based on the current date. Any service delivery satisfying this check will have its state updated to 'In Progress'.

StartEffectiveServiceDeliveries intitiates the following processing steps:

1. Searches for all service deliveries in system which are in 'Not Started' State.
2. Processes all qualifying service deliveries to start service delivery.
3. Updates the status to 'In Progress'
4. Updates the service start date for the cover period.

Class and method

StartEffectiveServiceDeliveries.startEffectiveServiceDeliveries.

Parameters

StartEffectiveServiceDeliveries has no input parameters.

1.30 TransferInstructionLineItems

The TransferInstructionLineItems batch process selects unprocessed financial instruction line items and transfers each one to the configured ERP financial system.

Processing steps

TransferInstructionLineItems initiates the following processing steps:

1. Generates a rolled up list of un-processed ILIs to be transferred to the ERP system. This includes payment ILIs , tax ILIs, utility ILIs, deduction ILIs and surcharge ILIs which are generated for any liabilities that are due for processing.

   Note: Unapplied deduction ILIs will not be transferred.
2. Transfers each ILI in the list to the ERP system, setting the status on the ILI to 'Transferred'.

Note: The Transfer Instruction Line Item batch process should always be executed immediately after the Generate Instruction Line Items batch to ensure that the latest ILIs are copied to the ERP system.

Class and method

TransferInstructionLineItems.transferInstructionLineItemsDue

TransferInstructionLineItemsStream.process

Parameters

Parameter	Description	Default value
Instance Identifier	ID that allows multiple instances of the same batch process to run at the same time.	n/a
Processing Date	The date the batch program will be processed.	n/a

Related information

TransferInstructionLineItemsStream uses batch streaming to distribute the processing load of TransferInstructionLineItems. When TransferInstructionLineItems is streamed, the application property curam.workflow.gentransferilifailedticket determines whether a notification is generated when an instruction line item does not transfer automatically.

1.31 WorkParticipationBatchProcess

This batch process loads the Business Intelligence tables for Income support reports.

Notices

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the Merative website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of Merative

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of Merative.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

Merative reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by Merative, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

MERATIVE MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Merative or its licensors may have patents or pending patent applications covering subject matter described in this document. The furnishing of this documentation does not grant you any license to these patents.

Information concerning non-Merative products was obtained from the suppliers of those products, their published announcements or other publicly available sources. Merative has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-Merative products. Questions on the capabilities of non-Merative products should be addressed to the suppliers of those products.

Any references in this information to non-Merative websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those

websites are not part of the materials for this Merative product and use of those websites is at your own risk.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

The licensed program described in this document and all licensed material available for it are provided by Merative under terms of the Merative Client Agreement.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to Merative, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. Merative, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. Merative shall not be liable for any damages arising out of your use of the sample programs.

Privacy policy

The Merative privacy policy is available at https://www.merative.com/privacy.

Trademarks

Merative ™ and the Merative ™ logo are trademarks of Merative US L.P. in the United States and other countries.

IBM®, the IBM® logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

Adobe™, the Adobe™ logo, PostScript™, and the PostScript™ logo are either registered trademarks or trademarks of Adobe™ Systems Incorporated in the United States, and/or other countries.

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Microsoft™, Windows™, and the Windows™ logo are trademarks of Microsoft™ Corporation in the United States, other countries, or both.

UNIX™ is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.