Cúram 8.1.2

Operations Guide

Note

Before using this information and the product it supports, read the information in Notices on page 41

Edition

This edition applies to Cúram 8.1, 8.1.1, and 8.1.2.

Contents

1 Administering operations for a production system

Use this information to complete operational administration tasks for a runtime production or production-like system.

1.1 Advanced Setup

Overview

This chapter provides advanced setup steps you may wish to perform on your production Cúram system. The following is a list of the main topics covered:

- Storing XSL Templates for use in the application
- Creating the Production Database in the application
- Data Conversion Issues
- Creating your Organization in the application

If you are installing the application for demonstration or investigation purposes, then this chapter can be skipped.

Storing XSL Templates for Use in the Application

The application uses the Cúram XML Server to format and print pro forma correspondence letters. The Cúram XML Server formats these letters by merging a letter template (stored in XSL format) with any free text entered by the user and data retrieved from the database.

Sample XSL templates are inserted onto your Database Setup Utility described in the *Cúram Runtime* chapter of the *Cúram Installation Guide* for your platform type. You may create your own XSL templates using your preferred XSL editing tool for your hardware platform.

Note: The application does not provide any XSL editing tools.

In order to make a template available to the Cúram XML Server, you must follow the steps below:

1. Log on to the System Administration application using your username and password.

   Note: "Out of the box", the application provides a default administration username of "sysadmin" with a password "password".
2. The home page is displayed. Click on the *System Configurations* section.
3. Click on the *Communications - XSL Templates* link in the ShortCuts panel.
4. The Templates page is displayed, showing a list of XSL templates stored in the system.
5. Click the *New* button to add a new template.

6. The Create Template page is displayed, requesting details of your template. Click the *Browse* button to locate the template on your file system. Enter the details of your template and click *Save*.

Note: The version of XSL supported is XSL-FO version 1.0 W3C Recommendation.

Creating the Production Database in the Application

The *Cúram Runtime* chapter of the *Cúram Installation Guide* for your platform type describes how to create a basic database for your installation of the application.

However, in a production environment, it is likely that you will need to have far greater involvement in creating a database in the application that is in line with your established database administration practices. Below are suggested steps to create a database suitable for use with a production installation of the application:

1. At a command prompt, change directory to the root of the runtime installation directory.
2. Run the following command:

   ```
   build database
   ```
3. Review the contents of the *build/datamanager* subdirectory within your installation directory. This directory contains a number of generated scripts which define the structure and initial data of your database.
4. Refine or refactor the DDL/SQL commands in line with your requirements, e.g., you may wish to:
   - Specify the creation of your physical database, e.g., implement a partitioning strategy
   - Share the physical database with other applications you may have
   - Specify the physical attributes of tables in the application, e.g., expected growth rate
   - Customize the initial data required by the application, e.g., change the function privileges for your security roles
   - Omit foreign key constraints as they are not supported in a production environment

     Note: The Cúram Reference Application enforces referential integrity (RI) in the application and as such using database-enforced RI will degrade performance. It is strongly recommended that in a production environment you do not create foreign key constraints on your database.
   - Load data from your existing system(s) (NB see Data Conversion Issues on page 10 below)
   - Refine the indexing strategy, e.g., implement hashed indexes
5. Use your refined/refactored script(s) (in addition to any established administrative procedures you may have) to create your production database in the application.

Data Conversion Issues

In a production environment, it is likely that you will need to migrate data from your existing system(s) to your Cúram database.

The specification of such a migration exercise is outside the scope of this document since it requires in-depth knowledge of the following:

- Cúram Reference Model, together with any customizations you may have made
- Structure and integrity of your existing data
- Any requirements you may have for the ongoing synchronization of data between Cúram and your other systems
- Any existing migration procedures you may have

Notwithstanding the above, the following suggestions are worth making here:

- Sample initial data that is required to start the Cúram online server is provided. This initial data contains such items as an administrative user (admin). The initial data may be customized before loading into your database, or may be customized through the application itself (before "going live").
- Demonstration data is provided and it is suggested that you do not load this data into your production database.
- You may wish to pre-sort your data in line with your data clustering strategy.
- The Cúram Reference Model includes database foreign key constraints which help maintain the integrity of the Cúram data. If these constraints are applied before loading your migrated data, then constraint violations may occur if your data is not loaded in "parent-child" order

  Note: Often in a relational database a "parent" entity is associated with zero, one or more "child" entities; each of these child entities bears the key of its parent, and as such, the parent must be created first so that its key is available (in order to set the parent's key on the child) when the child is subsequently stored.

  ; therefore, it is suggested that foreign key constraints are applied after your migrated data has been loaded

  Note: It is recommended that you create foreign key constraints to identify integrity problems in your converted data, and then (once any problems have been resolved) drop these constraints as they are not supported for production databases.

  Any constraint that is rejected by the database will indicate an integrity problem in your migrated data.
- The Cúram Reference Model includes indexes to support all SQL queries used from within the Cúram Server Application. You may wish to drop some of these indexes in order to improve the performance of database write operations, at the expense of degrading the performance of some rarely-used queries. This exercise can only be undertaken once you have an understanding of which transactions in the application will be used online heavily at your installation and which will be used rarely online or solely in batch.

Creating Your Organization in the Application

The application database needs to contain information about your Organization and its organization units and users. Sample Organization data is provided that will allow you to start the Cúram Online Server Application and logon from a Cúram Client. This sample Organization data includes a sample organization structure. An organization structure consists of a hierarchy of organization units where each unit can have any number of assigned positions and each position can have any number of assigned users.

The setting up of data pertaining to your Organization is described in *the Cúram Administration* guide.

1.2 Running the Cúram Online System

Overview

This chapter describes how to run and administer the Cúram online system.

Starting the System

This section describes how to start the Cúram online system. The Cúram Server Application should be started after the Cúram XML Server if your runtime configuration includes the Cúram XML Server.

Cúram XML Server

To start the Cúram XML Server, run the following command from the *XMLServer* subdirectory within your installation directory:

```
ant -f xmlserver.xml
```

For Microsoft Windows On Microsoft ® Windows, you can also start the Cúram XML Server by clicking on the shortcut **Start** > **Programs** > **Cúram** > **XML Server** > **Start XML Server**.

Please refer to the *Cúram XML Infrastructure Guide* for further information.

Cúram Server Application

To start the Cúram Server Application, run the following command from your Cúram Runtime installation directory:

```
build startserver -Dserver.name=curam1
```

where `curam1` is the value of the `curam.server.name` property.

For Microsoft Windows The following additional options are available to start the Cúram Server Application:

- Click on the shortcut **Start** > **Programs** > **Cúram** > **Application Server** > **Online** > **Start Server**
- From your Cúram Runtime installation directory, run the following command:

  ```
  StartServer.bat
  ```

Please refer to the Cúram deployment guide for your chosen application server for further information.

Stopping the System

This section describes how to stop the Cúram online system. Note that the Cúram Server Application should be stopped before the Cúram XML Server.

Cúram Server Application

To stop the Cúram Server Application, run the following command from your Cúram Runtime installation directory:

```
build stopserver -Dserver.name=curam1
```

where `curam1` is the value of the `curam.server.name` property.

For Windows The following additional options are available to stop the Cúram Server Application:

- Click on the shortcut **Start** > **Programs** > **Cúram** > **Application Server** > **Online** > **Stop Server**
- From your Cúram Runtime installation directory, run the following command:

  ```
  StopServer.bat
  ```

Note: If you intend to stop your Cúram Server Application, you should advise any online users to complete their work before stopping the server.

Cúram XML Server

To stop the Cúram XML Server, enter ^C.

For Windows To stop the Cúram XML Server, you can also click "X" to close the console window.

Please refer to the *Cúram XML Infrastructure Guide* for further information.

System Logs

IBM® WebSphere® Application Server

By default, the online logs are stored in the directory *<WebSphere installation directory>\AppServer\profiles\AppServ01\logs\<Server name>* where *<Server Name>* is the WebSphere® Application Server added for the Cúram EARs.

WebSphere® Application Server Liberty

Kubernetes

By default, the WebSphere® Liberty logs are stored in *server.config.dir/logs/*, where *server.config.dir* is the WebSphere® Liberty property that represents a server-specific configuration directory.

Oracle WebLogic Server

By default, the online logs are stored in the directory *<BEA Install directory/user_projects/domains/<DomainName>/servers/<ServerName>* where *<Server Name>* is the name of the server.

1.3 Running batch processes in the application

You can run batch processes in the application. Learn about the characteristics of batch processes, the Batch Launcher, and running standalone batch processes.

Batch processes in the application have the following characteristics:

- Each batch process is specified in and generated from the Cúram model.
- At runtime, each batch process accepts configuration parameters. The way the parameters are passed to the batch process depends on whether the batch process is started by the Batch Launcher or as a "standalone" process.
- Each batch process stores a log of its progress (one log is stored per each invocation of a batch process).
- Each log is automatically emailed to a nominated user.

For more information about modeling batch processes in the application or the parameters common to all batch processes, .

The Batch Launcher

The Batch Launcher provides a deferred request mechanism. Online application users can record requests for batch processes to be run. The batch operator can run all the batch processes requested by online users.

Note: The Batch Launcher executes the batch processes in the order that requests were received from users. If you have a requirement for batch processes to run in a particular order, for example, if there are functional dependencies between them, then the Batch Launcher might be unsuitable for running these processes, see Running stand-alone batch Processes on page 17.

Batch Process Requests

To record a request for a batch process to be executed, you must follow these steps:

1. Log on to the application as a system administrator.
2. The home page is displayed. Click the **Administration** shortcut.
3. The **Administration** home page is displayed. Click the **System Configurations** section.
4. Click the **Batch Processes** tab.
5. The **Batch Processes** page is displayed, showing a list of available batch processes.
6. To execute a batch process, click the processes **Execute** button.
7. The **Execute Batch Process** page is displayed, showing a list of the parameters that are accepted by the batch process. Enter any parameters needed for your selected batch process.
8. Click **Execute** to queue your batch process for execution. The system records the request to run the batch process.

Running the Batch Launcher

The Batch Launcher is ideally run when the Cúram online application is "down"; however, if there is no danger that programs run from the Batch Launcher will lock out the application database (e.g., with large numbers of uncommitted updates) for an extended period, then the Batch Launcher may be run whilst the Cúram online application is "up".

To run the Batch Launcher, run the following command from the root directory of your runtime installation.

```
build runbatch
```

For Microsoft Windows You can also run the Batch Launcher by clicking on the shortcut **Start** > **Programs** > **Curam** > **Runtime** > **Run Batch Launcher**.

Cúram XML Server

To start the Cúram XML Server, run the following command from the *`XMLServer`* subdirectory within your installation directory:

```
ant -f xmlserver.xml
```

Configuring the Online Interface of Batch Processes

The online Batch Processes list may be customized by:

- Maintaining the batch process list
- Maintaining batch process groups

Maintaining the Batch Process List

Batch processes may be added to or removed from the list displayed to an online application user.

Note: Initially this list contains all the batch processes available in the application.

This may be useful if, for example, there are batch executables which you require to run as stand-alone programs rather than allowing a user to request that the batch process be run; such programs may be removed from the list.

Viewing the List of Batch Processes

To view the list of batch processes, navigate as follows:

1. Log on to the application as a system administrator.
2. The home page is displayed. Click **System Configurations** > **Batch Processes**.
3. The **Batch Processes** page is displayed, showing a list of available batch processes.

Adding a New Batch Process to the List

To add a new program to the list, navigate as follows:

1. Navigate to the list of batch processes as described above.
2. Click the New button.
3. A page appears requesting details of the new batch process. Enter the details and click Save.

Updating a Batch Process Already in the List

To update an existing batch process in the list, navigate as follows:

1. Navigate to the list of batch processes as described above.
2. Click Edit action next to your chosen batch process.
3. A page appears showing details of the new batch process. Change the details and click Save.

Removing a Batch Process from the List

To remove a batch process from the list, navigate as follows:

1. Navigate to the list of batch processes as described above.
2. Click Remove action next to your chosen batch process.

Maintaining Batch Process Groups

The application supports grouping batch processes into process groups, to aid categorization and navigation.

Note: For example, you may wish to create a group containing statistical reports, another containing financial processing, etc.

Note: Each batch process may belong to zero, one, or more batch groups.

Viewing the List of Batch Process Groups

To view the list of batch process groups, navigate as follows:

1. Click the Batch - Process Groups link in the ShortCuts Panel.
2. A list of Batch Groups is displayed.

Adding a New Batch Group to the List

To add a new group to the list, navigate as follows:

1. Navigate to the list of Process Groups as described above.
2. Click on the New button.
3. A page appears requesting details of the new batch process group. Enter the details and click Save.

Removing a Batch Process Group from the List

To remove a group from the list, navigate as follows:

1. Navigate to the list of batch process groups as described above.
2. Click Remove action next to your chosen batch process group.

Adding a Batch Process to a Group

To add a batch process to a group, navigate as follows:

1. Navigate to the list of batch process groups as described above.
2. Click View next to your chosen batch process group.
3. The details of the batch process group are displayed. Click the Add button.
4. A list of batch process is displayed. Choose a batch process and click Save.

Removing a Batch Process from a Group

To remove a batch process from a group, navigate as follows:

1. Navigate to the list of batch process groups as described above.
2. Click View next to your chosen batch process group.
3. A list of batch processes in the batch process group is displayed.
4. Click Remove next to the batch process you wish to remove from the batch process group.

Executing a Batch Process from a Group

Once in a group, a batch process may be executed from the group tree as follows:

1. Navigate to the list of batch process groups as described above.
2. Click View next to your chosen batch process group.
3. A list of batch processes in the batch process group is displayed.
4. Click Execute action next to the batch process you wish to execute.

Running stand-alone batch Processes

Each batch process can be run in a "stand-alone" mode.

Batch processes are typically run as stand-alone programs if:

- A batch process is being used for demonstration or testing purposes.
- There is a functional dependency or order between a suite of batch processes, that is, one particular batch process must successfully complete before another particular batch process is allowed to begin.
- A batch process (or suite of programs) is run on a regular basis, such as nightly, and it is cumbersome or inconvenient to use the Batch Launcher to record a request to run the batch process every time it is needed.

Command Format

Each batch process accepts input parameters passed on the command line. The parameters required vary between batch processes.

Note: The parameters accepted by a batch process may be viewed in the online Cúram Application prior to scheduling it for execution.

To execute a batch process in stand-alone mode, run the following (one-line) command from the runtime directory:

```
build runbatch
-Dbatch.program=
curam.intf.
<program class name>.<program operation name>
-Dbatch.username=<your Cúram batch user>
-Dbatch.parameters=
"<comma-separated list of parameter name=
value pairs>"
```

An example of this type of command is:

```
build runbatch
-Dbatch.program=
curam.intf.PersonExtract.extractPersonDetails
```

```
-Dbatch.username=superuser
-Dbatch.parameters="extractFilePath=c:\testfile.dat"
```

Note: If you are repeatedly running only *one* batch process (e.g., for testing purposes), it may be convenient to remove the "-D" parameters from the command line and place them in your *project*`\properties\Bootstrap.properties` file within your Cúram Runtime installation directory.

Scheduling of Batch Processes

If you require to schedule a batch process (or suite of batch processes) to run at a particular time, then you may use your preferred choice of scheduling tool. In order to configure your scheduling tool, batch processes return a status value as specified in your `BatchErrorCodes` codetable.

Note: No scheduling tools are included with the application.

System Logs

The batch logs are stored in *`buildlogs`* subdirectory within your Cúram Runtime installation directory and are named *`<batch process name><date and timestamp>.log`*.

A log is created for each invocation of a batch process executable and shows the progress of the batch process. The contents of the log is automatically emailed to the email address configured in your *`project\properties\Bootstrap.properties`* file or within the Properties Administration area of the Administration page in the application.

1.4 Core Batch Processes

Overview

This chapter lists the batch processes which provide core functionality in the application.

Note: Unless specified otherwise, batch processes may be run 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

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.

Related concepts

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`.

1.5 Cúram Configuration Settings

Introduction

This chapter describes the configuration settings for the

- Cúram XML Server
- Cúram Server Application

The properties listed in this Chapter are the complete set of properties for the entire Cúram Business Application Suite. Hence not all properties listed here will be relevant to your application.

Cúram XML Server

The *XMLServer* subdirectory of your installation directory contains the configuration file *xmlserverconfig.xml*.

See the *Cúram XML Infrastructure Guide* for details of the configuration settings in this file.

Cúram Server Application

Managing Configuration Settings

When the database is built, the Application.prx file is used to set up the initial configuration settings on the database. Once the database is built, all settings can be administered online as part of application administration (see the *Cúram System Configuration* guide).

Configuration settings can either be dynamic or static. Any changes made to dynamic configuration settings are automatically applied to the runtime application once these changes are published at runtime. Any changes made to static configuration settings will require a reboot to the system in order for these settings to be applied to the runtime application.

Publishing Configuration Settings New configuration setting information will not take effect until you publish these changes or reboot. To publish these changes, press Publish on the Properties page.

Changing configuration settings

To change configuration settings for the Cúram Server Application, you must follow these steps:

1. Log on to the application using your username and password.

   Note: "Out of the box", the application provides a default administration username of "sysadmin" with a password "password".
2. The home page is displayed. Click on the System Configurations section.
3. Click on the Application Data - Property Administration link in the ShortCuts panel.

4. The Properties Administration page is displayed with an option to filter the search by locale or category.
5. Select the appropriate filter and Click the Search button.
6. On the required property, click on Edit action.
7. The Edit Property page for the property is displayed.
8. Make the necessary changes to the property.
9. Press Save to save your changes.
10. The system records the new information.

Note: For more information on the Properties Administration area of the Cúram Admin Client, see the *Cúram System Administration* guide.

Available Configuration Settings

The Cúram Server Application is made up of these logical components:

- Cúram Server Infrastructure
- Core Cúram Server Application
- Customized Cúram Server Application

Application.prx lists the settings used by each of these components.

Cúram Server Infrastructure Configuration Settings

The settings for the Cúram Server Infrastructure are described in the *Cúram Server Developer's Guide*.

Customized Cúram Server Application

Contact your development team for details of settings required by any customizations of the core Cúram Server Application.

1.6 Monitoring Cúram processes

Use the following views to monitor and troubleshoot problems with process instances and to see process instance errors.

About this task

Use these views to see workflow processes and see specific errors in workflow and deferred processes. Plan to monitor the information in the following locations regularly for potential errors or exceptions. You can troubleshoot problems by steps such as suspending process instances or overriding event waits, or by retrying or aborting failed workflow process instances.

Monitoring Workflow process instances

Use the **Process Instances** view to see the status of each workflow process instance. By searching and filtering, you can see the current process instances and their status. Generally, the complete or in-progress processes are of most interest.

About this task

For troubleshooting, you have the following options:

- You can suspend a process instance that is in progress. You must resume the process instance before any further activities can run.
- You can stop a process instance that is in progress. Once aborted, a process instance cannot be resumed.
- All activities that wait for events to be raised have a failure mode where the event they are waiting on is raised before the activity runs. To progress such process instances, you can override the event wait.

Procedure

1. Log in as the admin user.
2. Select **Administration Workspace** > **Process Monitoring** > **Process Instances**
3. Use the search and filtering options to see the current workflow processes on the system.

Process Instance Errors

The Workflow Engine records information about errors that occur during the lifetime of a workflow process instance. You can use this information for troubleshooting problems with the process instance.

This troubleshooting includes retrying or aborting failed workflow process instances.

Retrying a failed process instance instructs the Workflow Engine to re-enact the workflow process instance from where it failed.

Aborting stops the process instance and its activities and closes any tasks that are associated with manual activities in the process instance. Depending on where the process was aborted, some manual steps might be required before the process is fully stopped.

Monitoring Process Instance Errors

Use the **Process Instance Errors** view to find Workflow process or Deferred Process errors.

About this task

Plan to monitor the **Process Instance Errors** view regularly for potential operational errors or exceptions. You can abort or retry failed workflow process instances.

Procedure

1. Log in as the admin user.
2. Select **Administration Workspace** > **Process Monitoring** > **Process Instance Errors**

3. Use the search and filtering options to find process instance errors.
4. Click the error details for more information.

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.