Merative Social Program Management 8.1

Batch Processes Developer Guide

Note

Before using this information and the product it supports, read the information in Notices on page 25

Edition

This edition applies to Merative™ Social Program Management 8.0.0, 8.0.1, 8.0.2, 8.0.3, and 8.1.

Contents

1 Developing batch processes

Use this information to learn how to specify, write, manage, configure, and run batch processes. The batch processing framework allows an external task scheduler to execute process class operations without user intervention. Users can request that certain batch processes be ran on their behalf. When started by a task scheduler, the batch launcher processes these requests and start the relevant batch processes with the parameters specified by the user.

1.1 Overview

This section describes the batch processing functionality of the Merative™ Social Program Management Server Development Environment (SDEJ). You will learn how to specify, write, manage, configure, and execute batch processes.

The SDEJ Batch Processing framework allows an external task scheduler to execute process class operations without user intervention.

Client application users can request that certain batch processes be executed on their behalf. A separate program, the Batch Launcher, when started by a task scheduler, will process these requests and start the relevant batch processes with the parameters specified by the user. This is particularly useful for such operations as report generation.

Prerequisites

You should be familiar with the *Cúram Modeling Reference Guide* and the Server Development Environment (SDEJ).

1.2 Batch Operations

This section details how to define batch stereotyped operations for your Merative™ Social Program Management Business Process Objects (BPOs).

Creating batch Operations

To create a batch process executable, simply designate an operation of a process stereotyped class in your UML model to have the stereotype of batch and develop the operation as you would normally do for any process class operation. The generator will produce SQL which will allow this operation to be submitted for execution by the Batch Launcher.

Restrictions

There are some restrictions on the use of the batch stereotype:

- There can be only one operation in a process class that uses the batch stereotype.

- The operation can take only one parameter that must be a struct.
- The struct parameter must be “flat”, that is, it must not aggregate any other structs.
- The operation return type should be void. Non-void return types are ignored, that is, treated as void.

Available Data Types

The table below describes the string representations of the basic server types that can be used in the struct parameter passed to a batch process operation.

Table 1: String Representations of the Basic Server Types

Type	String Representation
SVR_CHAR	The first character in the string is accepted as the character value.
SVR_INT8	A number in the range of -128 to 127, for example, -100, or 78.
SVR_INT16	A number in the range of -32768 to 32767, for example, 25601.
SVR_INT32	A number in the range of -2147483648 to 2147483647, for example, 40101.
SVR_INT64	A number in the range of -9223372036854775808 to 9223372036854775807, for example 3456789012.
SVR_FLOAT	A single precision floating point number. The maximum positive value is 3.402823466e+38 and the minimum positive value is 1.175494351e-38. For negative values, the magnitudes are the same. The numbers can be expressed in exponential form, -3.78123e3 or as a decimal -3781.23.
SVR_DOUBLE	A double precision floating point number. The maximum positive value is 1.7976931348623158e+308 and the minimum positive value is 2.2250738585072014e-308. For negative values, the magnitudes are the same. The numbers can be expressed in exponential or decimal form.
SVR_STRING	As is.
SVR_DATETIME	A date in ISO 8601:1988 format yyyymmddThhMMss. For example, 6:49:02pm on December 9, 1999, would be represented as 19991209T184902.
SVR_DATE	A date in ISO8601:1988 format yyyymmdd. For example, the previous date would be represented as 19991209.
SVR_MONEY	Same as for SVR::Double.
SVR_BOOLEAN	One of `true` or `false`. The value is case insensitive.

Batch Process Output

It is the responsibility of the batch operation to generate the output of the batch process. There is, however, no restriction on what output a batch operation may have: for example it might send its output to a printer, a database table, or generate an output file in a specified location.

1.3 The Batch Launcher

The Batch Launcher is responsible for handling batch requests. It is a separate program that does not require an entire application to be running and so may be started by a task scheduler.

During normal operation batch requests are made using the Batch Administration interface and are listed on the database. The Batch Launcher, when run, will process all these requests, in the order in which they were added.

The Batch Launcher can also be configured to run a single request by passing command line arguments to the Batch Launcher as detailed in Starting the batch launcher on page 11.

Since the Batch Launcher does not require the application server to be running, it does not perform any application level authentication. Instead it uses the data-source parameters to connect to the database.

Starting the batch launcher

Start the batch launcher by using Apache Ant to build the target **`runbatch`**. For example, on Microsoft® Windows use **`build runbatch`**.

You can also start the batch launcher by using Ant to build the default target in *`%SERVER_DIR%/build.xml`*.

In the default operation, the batch launcher reads the batch requests that are listed on the database. Alternatively, the parameters that are listed in the following table can be passed to the batch launcher to force it to start a single batch program as a Java argument that uses `-D`.

Table 2: Properties for starting a single batch process

Property name	Function
batch.program	*batch.program* is the fully qualified name of a batch operation. For more information, see the proceeding section.
batch.parameters	*batch.parameters* is the optional parameters for the batch process. For example, structField1=param1, structField2=param2, structField3=param3, and so on. Ensure that the value does not include white space.
batch.username	*batch.username* is a valid application username. *batch.username* is an optional parameter. For more information, see Security on page 13 .

Specifying the fully qualified name of a batch operation for the *batch.program* property

Note: The proceeding database tables are part of the internal infrastructure and are subject to change without notice.

The fully qualified name of a batch operation can be expressed as *appname.codepackage.intf.classname.operationname* for operations in code packages or *appname.intf.classname.operationname* for operations that are not in a code package where the following criteria applies:

- `appname` is the application name. Typically, the application name is `curam`.
- `codepackage` is the code package of the class that contains the operation.
- `classname` is the name of the class that contains the operation.
- `operationname` is the name of the operation.

You can obtain `appname`, `classname` and `operationname` from the BatchProcDef database table. You can identify the code package from the FunctionIdentifier table. The key for looking up this table is file name. The table is constructed as `classname.operationname`.

If a batch program fails, an email is sent to the recipient that is specified by the property `curam.batchlauncher.erroremail.recipient` where the property is set. The batch request remains on the queue and the batch launcher stops immediately, that is, the batch launcher does not try to process any other pending batch requests.

Note: Batch programs that run on IBM® z/OS® require that IBM® WebSphere® Application Server for z/OS is installed.

Running batch programs from the command line

You can also run batch programs from the command line. By substituting the appropriate values for `<username>`, `<ClassName>`, and `<OperationName>`, and any parameter name-value pairs, you can run the following command from the main project directory EJBServer:

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

Application Properties for the Batch Launcher

The following properties must be set in the *`Application.prx`* file:

Table 3: Mandatory application properties specific to the Batch Launcher

Property Name	Purpose
curam.batchlauncher.erroremail.recipient	The email address of the recipient for error messages from each batch job.

Property Name	Purpose
curam.mail.smtp.serverhost	The host name of the Internet email server to use for sending error emails.

In addition, it is necessary to specify data source parameters to enable the Batch Launcher to connect to the database. For more information on data source parameters, see the *Cúram Server Developer's Guide*.

The following properties are optional in the *`Application.prx`* file:

Table 4: Optional application properties specific to the Batch Launcher

Property Name	Purpose
curam.batchlauncher.erroremail.nostacktrace	Prevents the stack trace from being included in the email, which is sent if an un-handled exception occurs. If set to `true`, only the top level description of the exception is included in the body of the email.
curam.batchlauncher.default.error.code	Specifies a default return code for the Batch Launcher when no code is found in the BatchErrorCodes database table. If not specified the value defaults to 1.

Security

Since the Batch Launcher does not require the application server to be running, it does not perform any application level authentication or authorization. It must only authenticate against the database. The same credentials as used by the application server (located in *`Bootstrap.properties`*) are used by the Batch Launcher to connect to the database and run batch programs.

The property batch.username can be used to specify the user name for the operations run by the Batch Launcher. Setting this property will affect the user name recorded in the audit trails, the effective locale for the batch operation, and the result of the `TransactionInfo.getProgramUser()` method.

The effective locale for the batch operations is the default locale for the Merative™ Social Program Management server. If the batch.username property is specified, the effective locale is the default locale for the user specified.

If the batch.username property is not specified, the result of the `TransactionInfo.getProgramUser()` method will be null.

Batch Error Codes

Like most applications, the Batch Launcher returns an integer value to the operating system upon ending. Typically a return value of zero indicates success, and other value denotes an error

condition. By default the return code from the Batch Launcher in the event of an error is `1`, or the value specified by property curam.batchlauncher.default.error.code.

To give greater flexibility in error handling, it is also possible to map individual application error messages to different error codes. This would enable a script which runs the batch launcher to take different actions depending on the return value from the Batch Launcher.

For example to map a run time exception with message curam.util.message.infrastructure.ID_RECIP_EMAIL_ERROR to a return code `22` you simply need to add a record to table BatchErrorCodes, containing `infrastructure.ID_RECIP_EMAIL_ERROR` and `22` in fields ErrorCodeID and ErrorCode respectively.

This table can be administered using the Batch Administration interface.

If an error other than a subclass of `AppException` or `AppRuntimeException` occurs during a Batch Launcher run this will be wrapped in a curam.util.message.infrastructure.ID_UNHANDLED to allow for it to be customized on the BatchErrorCodes database table.

Batch Launcher Output Directory

Since the Batch Launcher is a stand-alone Java® program, its “current” or “base” directory is determined by the location from which the Java VM is launched. This in turn determines where any outputs produced by batch programs will be written to.

If required, this base location can be specified by setting the batch.base.dir property when running the Batch Launcher, that is,

`build runbatch -Dbatch.base.dir=<Directory>`

Where <*`Directory`*> is the new location required.

The default location from which the VM is launched is *`%SERVER_DIR%\buildlogs`*. Where SERVER_DIR is your platform environment variable (e.g., %SERVER_DIR% on Windows).

Output from the Batch Launcher is captured in the output directory via the Ant `<record>` task in files that are named *`BatchLauncheryyyyMMddHHmmss.log`*; where " `yyyy` " maps to the year, " `MM` " to the month, " `dd` " to the day, " `HH` " to the hour, " `mm` " to the minute, and " `ss` " to the second. Thus, each output log file is named based on when the Ant `runbatch` target is invoked and batch jobs that start at the same time will write to the same file.

Debugging Batch Programs

The Batch Launcher can also be used as a way of running/debugging or testing batch programs in your IDE.

Add a new Java class in EJBServer project which wraps the `BatchLauncher` class in the CuramSDEJ project. This newly added class should have a `main()` method, which delegates straight through to the `main()` method of the `curam.util.impl.BatchLauncher` class in the CuramSDEJ project. For example:

```
public static void main(String[] args) {

try {
     curam.util.impl.BatchLauncher.main(args);
```

```
    } catch (org.apache.tools.ant.ExitException e) {
     System.exit(e.getStatus());
  }
}
```

This class takes up to three arguments which are listed in order in the table in Starting the batch launcher on page 11. If no arguments are passed, this class will run all batch programs enqueued in table BatchProcRequest.

The advantages of using the Batch Launcher rather than a handcrafted test harness to launch your program are:

- Database transactions are correctly handled.
- Other transaction information such asBusiness Date is correctly setup. For more information on theBusiness Date see the *Cúram Modeling Reference Guide*

1.4 Administration and Scheduling

Batch Requests need to be submitted to be processed by the Batch Launcher whenever it is run. This is done via the Batch Administration interface.

All you need to do then, is schedule the Batch Launcher to run when you want the batch requests to be handled. You can use any third-party task scheduling tool of your choice. None is provided with the Merative™ Social Program Management SDEJ.

Administration Interface

The SDEJ provides an Administration interface for configuring and modifying the batch administration database tables. This Administration interface is listed below and more information on its methods and parameters can be found in the JavaDoc of class `curam.util.administration.intf.BatchAdmin`

The database tables that this interface modifies are detailed in the following sections.

Batch Process Definitions and Descriptions

When the model is generated, an XML file called `<ProjectName>_batch.xml` will be generated which contains the definitions of your batch processes. In order for users to issue batch process requests, this information must be loaded into the database using the Data Manager.

Once the information has been loaded it is possible to use the Administration interface to add further details about the batch processes and their parameters. The information that should be added is as follows:

- A description of each of the batch processes;
- The type of each batch process;
- A description of each of the parameters to each of the batch process. These are actually the fields of the respective batch operations struct fields;
- A default value for each of the parameters to each of the batch processes. This value should represent a valid input.

This information is contained in a number of database tables as shown in the following diagram. These tables are described in more detail below.

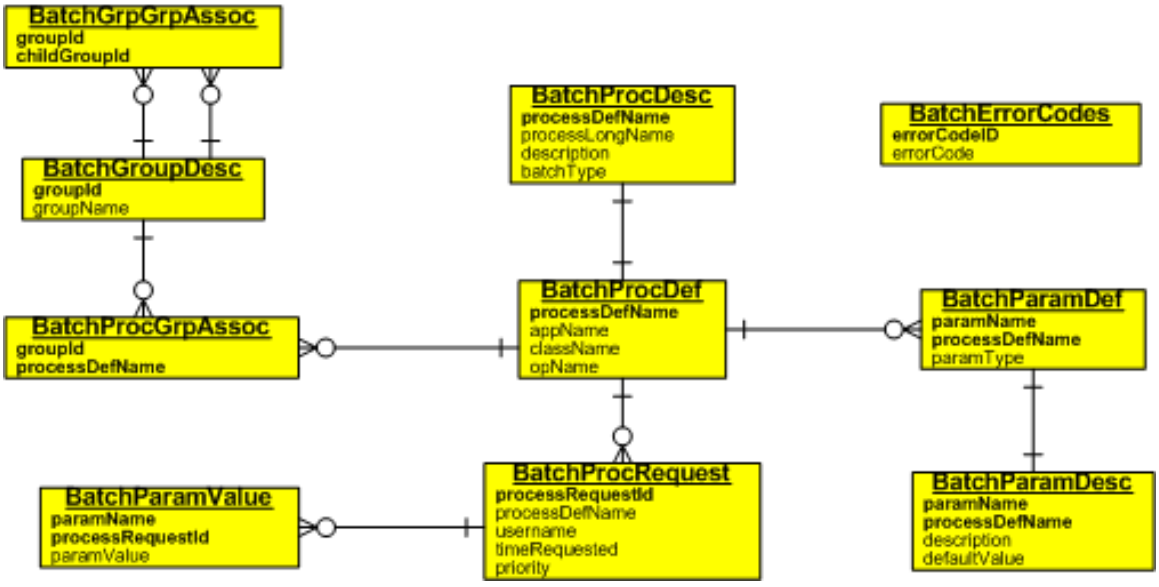

Figure 1: Batch database tables

- BatchProcDef

  The Batch Process Definition table contains the definitions of the batch processes as output from the model generators. The information in this table is loaded from the SQL file mentioned above. The fields in this table include:

  - processDefName - The name of the batch process.
  - appName - The name of your application. This is required to enable the fully qualified name of the batch operation to be determined.[1]
  - className - The name of the class containing the batch processing operation.
  - opName - The name of the operation that performs the batch processing.
- BatchParamDef

  The Batch Parameter Definition table contains the definitions of the parameters associated with the different batch processes. The information on this table is loaded from the SQL file mentioned above. The fields on this table include:

  - paramName - The name of the parameter.
  - processDefName - The name of the batch process.
  - paramType - The datatype of the parameter.
- BatchProcDesc

  The Batch Process Description table contains a user friendly description of the batch process. The information should be added using the Batch Administration interface. The fields on this table include:

[1] The fully qualified name of a batch operation is composed of the application name, the class name, and the operation name, separated by dots. For example, `curam.intf.Billing.generate`

- processDefName - The name of the batch process.
- processLongName - The descriptive name of the batch process.
- description - A description of the batch process.
- batchType - The type of batch process.

- BatchParamDesc

  The Batch Parameter Description table contains a user friendly description of the parameters associated with a batch process. The information should be added using the Batch Administration interface. The fields on this table include:

  - paramName - The name of the parameter.
  - processDefName - The name of the batch process.
  - description - The description of the parameter.
  - defaultValue - The default value for the parameter.

Batch Process Groups

You may also set up batch process groups and assign the processes to these groups. The groups will be presented to the user in a tree widget so that related batch process can be easily identified. For example, you might create a `Reports` group to hold your reporting processes.

There are a few rules and restrictions associated with these administration functions:

- A description must be added for a batch process before it can be added to a group;
- When a batch process description is added, a set of default batch parameter descriptions is added automatically. This is to ensure that the description and definition tables always correspond. These parameter descriptions can then be modified;
- A batch process can be added to multiple groups, but it cannot be added to the same group twice.

The batch grouping information is contained in a number of database tables:

- BatchGroupDesc

  The Batch Group Description table contains the descriptions of the batch process. The fields in this table include:

  - groupId - The ID of the batch group.
  - groupName - A descriptive name of the batch group.

- BatchProcGrpAssoc

  The Batch Process Group Association table contains the mapping between batch processes and batch groups. The fields in this table include:

  - groupId - The ID of the batch group.
  - processDefName - The name of the batch process.

- BatchGrpGrpAssoc

  This table is reserved for future use.

Batch Process Request

Batch Process Requests can be added using the Batch Administration interface and the information in contained in two database tables for the Batch Launcher to process.

- BatchProcRequest

  The Batch Process Request table contains the requests awaiting execution. The fields on this table include:

  - processRequestId - The unique ID for the batch process request.
  - processDefName - The name of the batch process to execute.
  - username - The name of the requesting user.
  - timeRequested - The time the batch process request was made.
  - priority - The priority of the batch process request.
- BatchParamValue

  The Batch Parameter Value table contains the parameter values that should be passed to a batch process request. The fields on this table include:

  - paramName - The name of the parameter.
  - processRequestId - The unique ID for the batch process request.
  - paramValue - The actual value of the parameter for the batch process request.

Batch Error Code Mapping

The Batch Error Codes table contains mappings from error codes to integers.

See Batch Error Codes on page 13 for more information about this feature.

1.5 Accessing Application Server Functionality

Merative™ Social Program Management batch programs are designed to run as Java (rather than Java EE) applications as this simplifies deployment and reduces licensing costs. However on occasion it is useful for a batch program to initiate functionality that is typically associated with online programs - such as starting workflows and deferred processes.

In an online application these rely on the presence of JMS (Java Message Service) which is not available to batch programs. However batch programs do have access to the database, so for those parts of workflow and deferred processing which put messages onto queues, a database table is used to temporarily store the messages. At some later stage a notification/trigger is sent to the online application. This triggers the online application to transfer the messages from the DB to the JMS queue. Therefore this mechanism is known as DB-to-JMS (short for Database-to-JMS).

How it works

DB-to-JMS works by intercepting messages sent to the Merative™ Social Program Management JMS queues in batch processing mode, and instead writing them to a database table. This means

that the standard transactional behavior of messages (they are not delivered until the originating transaction completes) is maintained.

Once the batch program transaction has been committed[2], the application server must be triggered to transfer the messages from the database table to their JMS queue(s).

The application server can be triggered by a call from within the application server, or by a call from a source outside the application server i.e. from the Batch Launcher or a batch program. The call from outside the application server consists of a single HTTP/HTTPS request which is handled by a servlet deployed in the server. This means that the external source must know the host name of the application server and the port number. This information is specified using application properties which are detailed below.

The application server does not need to be running in order for batch programs to use workflow or deferred processes. However it does need to be running in order to trigger the transfer of messages from the database table to the JMS queue(s).

This triggering can be performed in a number of ways:

- Automatically by the batch launcher at the end of each, or all batch programs. This is controlled by property curam.batchlauncher.dbtojms.notification.batchlaunchermode which provides fine grain control on triggering and is explained below.

  Depending on how this property is configured, this means that the application server must be running when a batch job completes execution, or when the batch launcher finishes processing all queued batch jobs.
- By a call from a batch program. By calling method `curam.util.resources.DBtoJMS.beginTransfer()`.

  Any work completed by the batch program must be committed and the application server must be running at the time of calling this method.
- By a call from an online program. By calling method `DBtoJMS.beginTransfer()`.

  Since this is an online program, the application server is guaranteed to be already running and will therefore be capable of being triggered.

Once the application server has been triggered, it will start a deferred process to transfer all messages from the database table to their target queues. To ensure that the deferred process does not take too long and cause a transaction timeout when processing a large number of messages, it will only process a fixed number of messages per transaction. If, at the end of the transaction there are still messages remaining, it will automatically start another deferred process to handle these, and so on. The optimal number of messages which can be processed within the duration of an EJB transaction is dependent on many factors such as hardware configuration, machine load, etc and is therefore specified by means of an application property: curam.batchlauncher.dbtojms.messagespertransaction. If server tracing is switched on, messages will be written to the server log showing the activities of the DB-to-JMS transactions complete with timings, which can be used to determine the optimal number of messages per transaction.

Only one triggering is required to cause all messages to be processed. Multiple triggering will result in multiple threads attempting to convert the same records which is harmless apart from wasting resources. In the event of a two threads attempting to process the same message, one of the threads will proceed, the other will skip (or back-off from) the record and, if tracing is enabled, a message will be written to the application log to this effect.

---

2 See the *Cúram Server Developer's Guide* for more information on Transaction Control.

Properties

The following properties are used by DB-to-JMS:

Table 5: Properties used by DB-to-JMS

Property Name	Type	Description
curam.batchlauncher.dbtojms.enabled	BOOLEAN	Default value is `false`. When this value is set to `true`, batch programs can use deferred processing and workflow.
curam.batchlauncher.dbtojms.notification.host	STRING	Specifies the name of the host on which the application server is running. This name is the same as the port on which theMerative™ Social Program Management client is listening. Batch programs require this information to trigger the DB-to-JMS conversion by calling `DBtoJMS.beginTransfer()`.
curam.batchlauncher.dbtojms.notification.port	INTEGER	Specifies the port number on which the application server is listening. This port number is the same as the port on which the client is listening. See property curam.batchlauncher.dbtojms.notification.host for more details.
curam.batchlauncher.dbtojms.notification.ssl	BOOLEAN	Default value is `true`. Specifies that the client application is listening on SSL (that is, uses HTTPS). This property determines whether the Batch Launcher uses HTTP or HTTPS to notify the application server to begin a transfer. Note that if an HTTPS notification fails, an HTTP notification is automatically attempted. This is to simplify switching between production mode, which uses SSL and development mode, which does not, without having to change the value of this property. See property curam.batchlauncher.dbtojms.notification.host for more details.
curam.batchlauncher.dbtojms.notification.ssl.protocol	STRING	Default value is SSL. Specifies the SSL protocol name (for example, SSL, TLS, and so on), which depends on the support that is provided by your JDK and application server. See the relevant documentation for protocol names valid in your environment. For this property to be used, the `curam.batchlauncher.dbtojms.notification.ssl` property must be set to `true`.
curam.batchlauncher.dbtojms.notification.encoding	STRING	Specifies the character encoding that is used on the application server. This property is only required if the application server is using a different character encoding to that of the batch launcher, and the batch launcher or batch program triggers the DB-to-JMS conversion by calling `DBtoJMS.beginTransfer()`.

Property Name	Type	Description
curam.batchlauncher.dbtojms.notification.batchlaunchermode	INTEGER	Default value is 0 (zero). Specifies the DB-to-JMS notification mode for the batch launcher. The following values are valid: • **0** No DB-to-JMS notification is performed by the batch launcher. • **1** One DB-to-JMS notification is performed by the batch launcher after all batch programs run, or if a single stand-alone batch program was ran by specifying the batch.program property. • **2** A DB-to-JMS notification is performed by the batch launcher after each batch program is run, or if a single stand-alone batch program was ran by specifying the batch.program property. Note that if this property is set to 1 or 2 then properties curam.batchlauncher.dbtojms.notification.host and curam.batchlauncher.dbtojms.notification.port must be set.
curam.batchlauncher.dbtojms.notification.disabledforstandalone	BOOLEAN	Default value is `false`. This property specifies that the batch launcher does not perform a DB-to-JMS notification when run in stand-alone mode that is, when the batch launcher is used to start a stand-alone operation by specifying property batch.program.
curam.batchlauncher.dbtojms.messagespertransaction	INTEGER	Default value is 512. Specifies the maximum number of messages to be processed per transaction when transferring pending messages from the database table to their JMS queues.
curam.batchlauncher.dbtojms.notification.test.stubbed	BOOLEAN	Default value is `false`. When this property is set to `true`, calls by batch programs or the Batch Launcher to `DBtoJMS.beginTransfer()` are stubbed out and take no effect. That is, this property prevents the method from attempting to contact an application server. This property is to enable debugging of batch programs if an application server becomes unavailable.

Property Name	Type	Description
curam.custom.deferredprocessing.dpcallback	STRING	Mandatory. Specifies the name of a custom callback class that implements interface `curam.util.deferredprocessing.impl.DPCallback` DB-to-JMS uses deferred processing, and this property must be set if deferred processing is used anywhere in the application. Method `dpHandleError` of this class is called whenever the deferred process used by DB-to-JMS fails. The developers implementation of this method must take appropriate action if the deferred process fails, such as sending an email or assigning a new task to a user.

Note: The property curam.test.stubdeferredprocessing, which is used internally by the SDEJ, can interfere with the operation of DB-to-JMS and must not be set. For more information about the Offline Unit-Testing of Deferred Processes, see the *Cúram Server Developer's Guide*.

Note: When you switch the application between a production and deployment environment, this switch can affect the host and port on which the application listens. For example, WebSphere Application Server can use a different port than Apache Tomcat. Therefore, you might need to change the curam.batchlauncher.dbtojms.notification.host and curam.batchlauncher.dbtojms.notification.port properties each time you change environment.

Deferred processing configuration required

Since DB-to-JMS runs as a deferred process, the deferred process `DB_TO_JMS` must be registered by adding an entry in the DPProcess table. This can be done by either an SQL statement or by a Data Manager file.

Examples of each are shown below.

```
-- Register the DB-to-JMS deferred process:
  INSERT INTO DPPROCESS
    (PROCESSNAME, INTERFACENAME, METHODNAME, TICKETTYPE, SUBJECT)
  VALUES
    ('DB_TO_JMS',
    'curam.util.internal.deferredprocessing.intf.DBtoJMSbpo',
    'continueTransfer', 'INF',
    'Transfers messages from database to JMS queues.');
```

Figure 2: Using an SQL statement to setup the DB-to-JMS deferred process

```
<!--  Register the DB-to-JMS deferred process: -->
  <table name="DPPROCESS">
    <column name="PROCESSNAME" type="text" />
    <column name="INTERFACENAME" type="text" />
    <column name="METHODNAME" type="text" />
    <column name="TICKETTYPE" type="text" />
    <column name="SUBJECT" type="text" />
    <row>
      <attribute name="PROCESSNAME">
```

```
      <value>DB_TO_JMS</value>
    </attribute>
    <attribute name="INTERFACENAME">
      <value>
      curam.util.internal.deferredprocessing.intf.DBtoJMSbpo
      </value>
    </attribute>
    <attribute name="METHODNAME">
      <value>continueTransfer</value>
    </attribute>
    <attribute name="TICKETTYPE">
      <value>INF</value>
    </attribute>
    <attribute name="SUBJECT">
      <value>Transfers messages from database to JMS queues.
      </value>
    </attribute>
  </row>
</table>
```

Figure 3: Using the data manager to setup the DB-to-JMS deferred process

Security Considerations

The Batch Launcher or batch programs can optionally trigger the application server to begin a DB-to-JMS transfer. This involves logging in and invoking a method on the server, which in turn requires a valid Cúram username and password.

By default the DB-to-JMS transfer operation uses user 'DBTOJMS', so this account must exist on the Cúram 'Users' table and must be enabled and assigned the role 'SYSTEMROLE'.

It is also possible to configure a different user name and password for DB-to-JMS. This is done by providing values for the following properties:

- `curam.security.credentials.dbtojms.username` - The username that Cúram uses to process JMS messages created by batch programs.
- `curam.security.credentials.dbtojms.password` - The password that Cúram uses to process JMS messages created by Batch programs. This password must be encrypted.

The above credentials must exist on the Cúram '`Users`' table, must be enabled, and should be assigned the role 'SYSTEMROLE'.

The previous mechanism for providing custom DB-to-JMS credentials, via the `curam.omega3.DBtoJMSCredentialsIntf`, APIs has been deprecated.

Related information

Limitations of DB-to-JMS

The DB-to-JMS mechanism has the following limitations:

- It is only used to put messages onto queues, not to take them off queues and process them.
- Messages can be seen only by the application server after the batch program transaction is committed.
- The application server does not poll the table for messages, it must be notified to do so.

- It is not a JMS implementation, that is, it works only for specific Merative™ Social Program Management queues.

1.6 Summary of Key Points

- Batch processes can be specified by adding an operation with a stereotype of batch to a process class in the Merative™ Social Program Management application UML model.
- The batch operation may take only one parameter of some struct type whose fields are of the simple server types.
- The return type of the batch operation should be `void`.
- The code generator will produce an XML file for input into the Data Manager to load the definitions of the batch operations into the database. Once there, descriptions need to be added to the operations and their parameters, and the operations can be added to groups through the Batch Administration interface.
- The Batch Launcher is responsible for processing batch requests made by users of the system. The launcher is configured via a combination of Java arguments passed in and properties on the database.
- A limited set of deferred processing and workflow features are available to batch programs (through DB-to-JMS, see 1.5 Accessing Application Server Functionality on page 18) even though these features normally require an application server and batch programs do not run within an application server.

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.