JBEAM Developer’s Guide

Table of Contents

[1. Getting started 4](#_Toc400984514)

[1.1. System Design & Architecture 5](#_Toc400984515)

[1.1.1. Core 5](#_Toc400984516)

[1.1.2. Monitor 7](#_Toc400984517)

[1.1.3. Communication 8](#_Toc400984518)

[1.2. Communication flow diagrams 8](#_Toc400984519)

[1.2.1. Start / Restart Batch [MONITOR to CORE] 9](#_Toc400984520)

[1.2.2. Stop Batch [MONITOR to CORE] 10](#_Toc400984521)

[1.2.3. Log Feed [CORE to MONITOR] 10](#_Toc400984522)

[1.2.4. Batch Execution Flow 12](#_Toc400984523)

[2. Working with JBEAM source code 13](#_Toc400984524)

[2.1. Core 13](#_Toc400984525)

[2.2. Core Communication 15](#_Toc400984526)

[2.3. Monitor Communication 16](#_Toc400984527)

[3. Build instructions 17](#_Toc400984528)

[3.1. Building components 17](#_Toc400984529)

[3.1.1. JBEAM Core components 17](#_Toc400984530)

[3.1.2. Building JBEAM UI 18](#_Toc400984531)

[10.1. Deploying components 20](#_Toc400984532)

[10.2. Configuring the Components 20](#_Toc400984533)

[10.3. Starting the Components 22](#_Toc400984534)

[11. Automated Testing 23](#_Toc400984535)

[12. Code snippets - Useful code samples 26](#_Toc400984536)

[12.1. Create a Java Batch Job 26](#_Toc400984537)

[12.2. Creating a Batch Execution Handler 28](#_Toc400984538)

[12.3. A more complex batch job 29](#_Toc400984539)

[13. Debugging - Helpful tips 32](#_Toc400984540)

[13.1. Refer following table for some common issues - 32](#_Toc400984541)

[13.2. JBEAM TIPS 34](#_Toc400984542)

[*Tip 1*: Future Run Restriction 34](#_Toc400984543)

[*Tip 2*: Terminate the Batch in case a critical batch job fails 35](#_Toc400984544)

[*Tip 3*: Create your project specific jars using ANT or Maven or any other build tool. 35](#_Toc400984545)

[*Tip 4*: Speed up the waiting time between the Processor and Listener 36](#_Toc400984546)

[14. Appendix 36](#_Toc400984547)

[14.1. Database Design & Table Structure 36](#_Toc400984548)

[14.2. Create a new communication message 55](#_Toc400984549)

[14.3. Create new / Override email content 56](#_Toc400984550)

[14.4. Create / Override batch job type / execution handler 56](#_Toc400984551)

[14.5. Enhance text logging 56](#_Toc400984552)

[14.6. Create new PRE / POST events 57](#_Toc400984553)

# Getting started

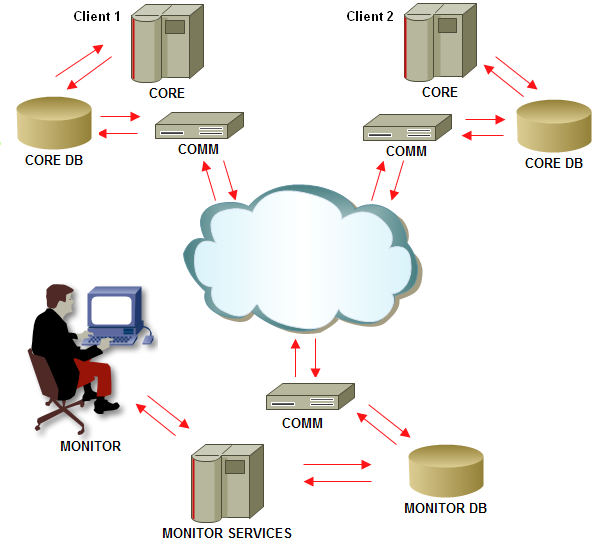
JBEAM stands for Java Batch Execution and Monitoring. It acts like a server for batch processing.

JBEAM mainly consists to two systems, Monitor and Core, and a UI. Core component is mainly responsible for execution of the batches. Monitor component is capable of monitoring more than one Core system and is responsible for interacting with UI.

The UI talks to the monitor for making any requests. The monitor in turn forwards these request to appropriate JBEAM Core instance where the processing takes place.

Details about these systems can be found in the discussion below.

## System Design & Architecture



The diagram to depict the overall architecture of system is illustrated above. The entire system comprises of the following sub systems.

## Core

The CORE contains the execution code that forms the processing unit of the system. All other sub systems are dependent on the results or the output of this core unit. The output of the CORE is dumped into the logging database. The CORE comprises of four important classes.

1. Processor Component

The processor component is the processing component that is scheduled. PRE schedules the execution of this component as set and works as the entry point into the entire system. The processor component reads from the configurations (and or the database) to configure the CORE system.

1. Assignment Component

The Assignment Component, as the name suggests, is responsible for assigning of work (through various internal cycles) and to schedule the listener objects into the PRE. The PRE, in turn, as requested would instantiate, initialize and execute the Listener Component.

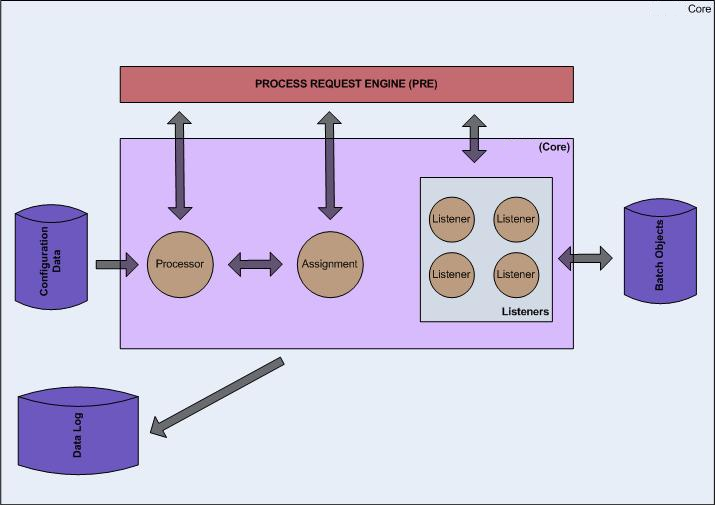
1. Listener Component

The listener components are essentially the worker objects that do the actual execution of the batch objects as assigned to them by the Assignment Component.

1. Listener Handler

Listener handlers are extensions to the listeners for specialized execution. The batch objects could comprise of either database objects (PLSQL), event parser objects (PLSQL objects that have to be used in conjunction with Event Parser system) or Java class. The handlers essentially specializes in execution of one of these object types.

The diagram below depicts the architecture diagram for the CORE system.



## Monitor

The MONITOR system is the user interface that would be used by the administration or the operation users to monitor and direct the proceedings of the batch. The user interface would allow the administration or the operational personnel to view the proceedings of the batch and also to instruct or direct the batch execution. The MONITOR system has two components.

1. UI – The User Interface or the presentation layer that would display the batch proceedings, statistics and other information.
2. Services – The services that the UI would need to display the data and or instruct a batch.

## Communication

The COMMUNICATION system, as the name suggests, is used to deliver and receive request messages between the CORE and the MONITOR. The system consists of a server and client components at each end. The server components contains of all the web services. The client component would have the client call (and the required stubs for those calls) to the server components.

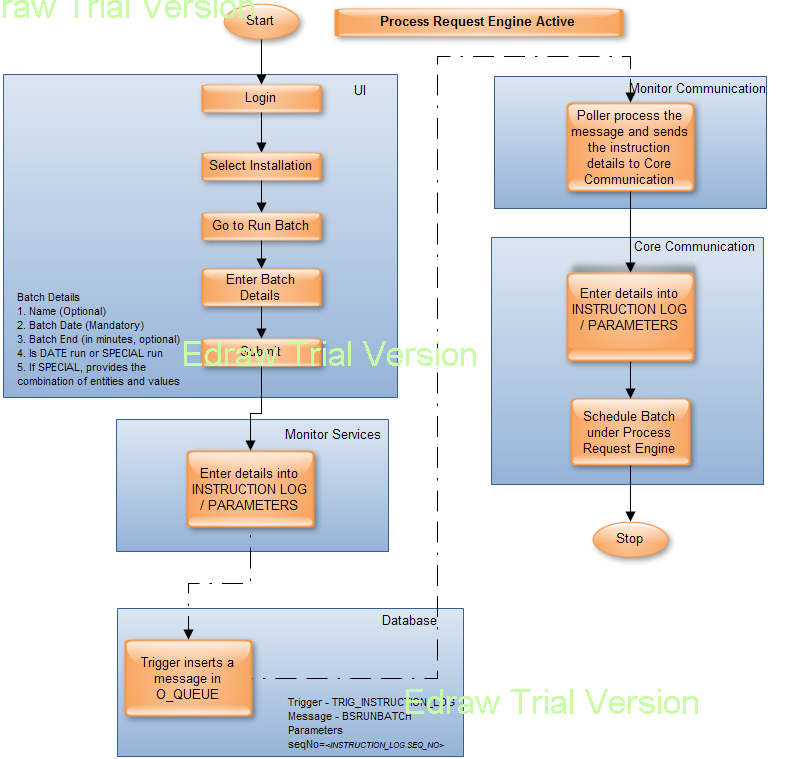
1. CORE-COMM – The communication piece at the CORE end.
2. MONITOR-COMM – The communication piece at the MONITOR end.

Each piece publishes the services needed by the other piece. At the same time each piece caches the client of the services published by the other piece, thereby making the communication possible.

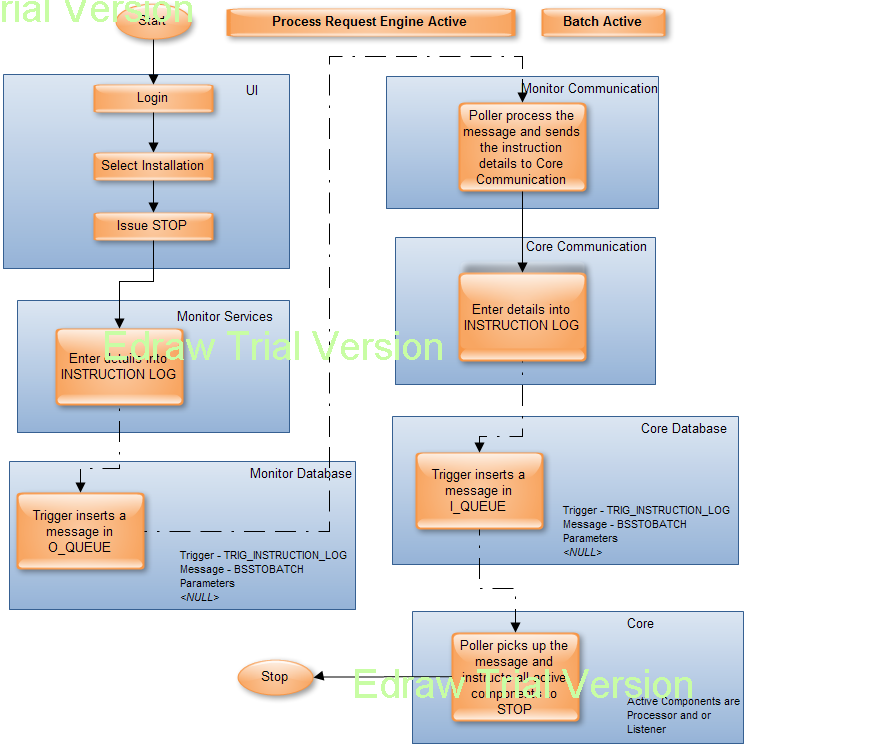
## Communication flow diagrams

Communication flow diagrams depicts how different type of messages flow through the system,

## Start / Restart Batch [MONITOR to CORE]

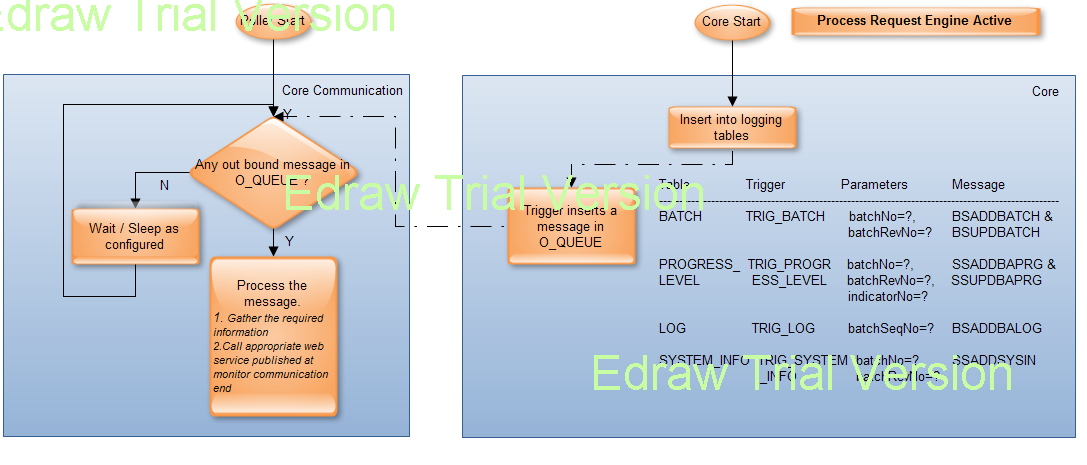


## Stop Batch [MONITOR to CORE]

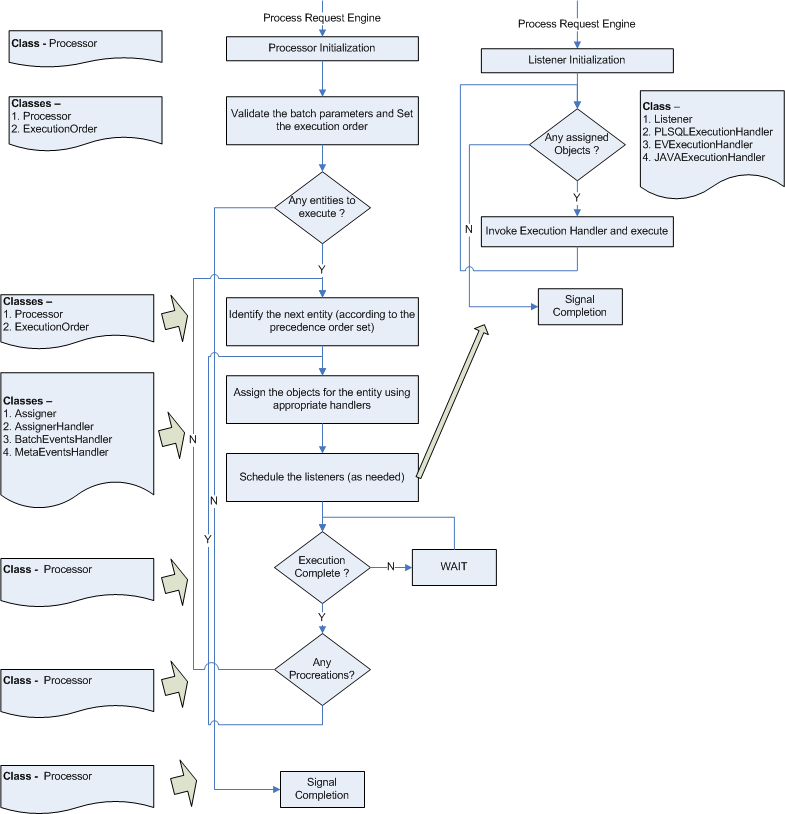


## Log Feed [CORE to MONITOR]

Log feed are the different types of information sent from the CORE to the MONITOR as a part of logging activity.



## Batch Execution Flow



# Working with JBEAM source code

Following are the important classes of JBEAM modules

## Core

|  |  |  |
| --- | --- | --- |
| **Class Name** | **Package** | **Description** |
| Processor.java | logic | The main processor class or the entry point into the CORE system.  The ‘heart and the soul’ of the CORE system.  This class is invoked and instantiated through Process Request Engine.  This class is responsible for using all other classes in the system (except Listeners) to achieve –   1. Procreation – procreating meta events (PRE / POST) as configured 2. Assignment – Assigning of the batch jobs with listeners 3. Scheduling – Scheduling of listeners in Process Request Engine 4. Execution – Execution, even though is done by Listeners, it still iterates in cycles and is termed as execution from the batch perspective |
| Listener.java | logic | The listeners are invoked through PRE and do the simplest of jobs that of executing a batch job or set of batch jobs assigned to it.  Listeners are uniquely identified by a listener identifier.  The processor assigns the listener identifier. |
| AssignerHandler.java | logic | Primarily responsible for assignment of listeners to a set of batch jobs.  Processor is responsible of using this assignment handler class to achieve assignment.  There are two types of batch objects and has a special handler class to cater to those.  Meta Events (PRE / POST) – logic/MetaEventsHandler.java  Batch Objects – logic/ BatchEventsHandler.java |
| ExecutionHandler.java | logic | Primarily responsible for execution of a batch object.  Invoked and used by the Listener.  There are three special execution handlers each catering to a job type.  JV – logic/ JAVAExecutionHandler.java  EV – logic/EventParserObjectExecutionHandler.java  PL – logic/PLSQLExecutionHandler.java  With advent of new job types or override the default implementation, one can create new or override the existing implementation and have it configured. *Please refer the section “How To” for more details.* |
| ExecutionOrder.java | logic | Responsible for setting the execution order for a batch.  Processor uses this class as a part of its initialization / validation phases. |
| InterruptBatch.java | messagehandler | Special Message handler class to get and process an instruction from the monitor.  The message that are processed by this class is ‘BSSTOBATCH’ i.e. STOP batch |
| IEmailContentGenerator.java | logic | Interface to override default implementation / create new email content for the emails at different stages during the proceedings of the batch. |
| MonitorInstructionPoller.java | util | As the name suggest polls the I\_QUEUE of the CORE database for any instruction from the MONITOR system. |
| CheckEndTime.aj | aspects | Special Aspect J class for determining whether the end of time is realized and the batch has to STOP.  All or most important methods annotated as ‘Marker’ would be picked up before the execution of the method to check the end of time functionality. |
| ProgressReport.aj | aspects | Special Aspect J class to identify different progress levels in the batch proceedings and recording those for logging purpose.  All methods in the system annotated with ‘LogTime’ would be picked up to track the progress level. |
| SendEmailAj.aj | aspects | Special Aspect J class that sends email after the execution of those methods marked or annotated with ‘Email’ |

## Core Communication

|  |  |  |
| --- | --- | --- |
| **Class Name** | **Package** | **Description** |
| OutBoundQueuePoller | util | As the name suggests, it polls for ant out bound messages to the MONITOR system. |
| TransmitBatchDetails.java | messagehandlers | Special message handler class to transmit the batch details or information.  The messages processed by this special class are –  BSADDBATCH – When a batch information is to be added  BSUPDBATCH – When the batch information is to be updated |
| TransmitBatchLog.java | messagehandlers | Special message handler class to transmit the batch log and related information.  The message processed by this class is –  BSADDBALOG – when a log entry in added into the LOG table. |
| TransmitProgressLevel.java | messagehandlers | Special message handler class to transmit the batch progress level information.  The messages processed by this class  SSADDBAPRG – when a progress level is to be added  SSUPDBAPRG – when a progress level is to be updated |
| TransmitSystemInformation.java | messagehandlers | Special message handler class to transmit the system information on which the batch is run.  The message processed by this class –  SSADDSYSIN – when system information is to be added. |

## Monitor Communication

|  |  |  |
| --- | --- | --- |
| **Class Name** | **Package** | **Description** |
| OutBoundQueuePoller | util | As the name suggests, it polls for ant out bound messages to the CORE system. |
| RunBatch | messagehandlers | Special class to transmit the information from the MONITOR to the CORE system to start an unscheduled batch.  The message processed by this class is –  BSRUNBATCH – Instruction to start the batch |
| StopBatch | messagehandlers | Special class to transmit the information from the MONITOR to the CORE system to stop a running batch.  BSSTOBATCH – Instruction to stop the batch  Note – This message also becomes the in-bound message for the CORE system |
| AddCalendar | messagehandlers | Special class to transmit the information from the MONITOR to the CORE system to add calendar data.  BSCALENDAR – Instruction to add calendar |

# Build instructions

## Building components

## JBEAM Core components

All components, except FLEX UI, are mavenized and are under the parent project jbeam

|  |  |
| --- | --- |
| Build script | Component |
| Pom.xml | The main pom.xml of jbeam |

Use following commands to build JBEAM core components

1. You can use these commands from the command prompt by being in the jbeam parent directory and using below mentioned commands

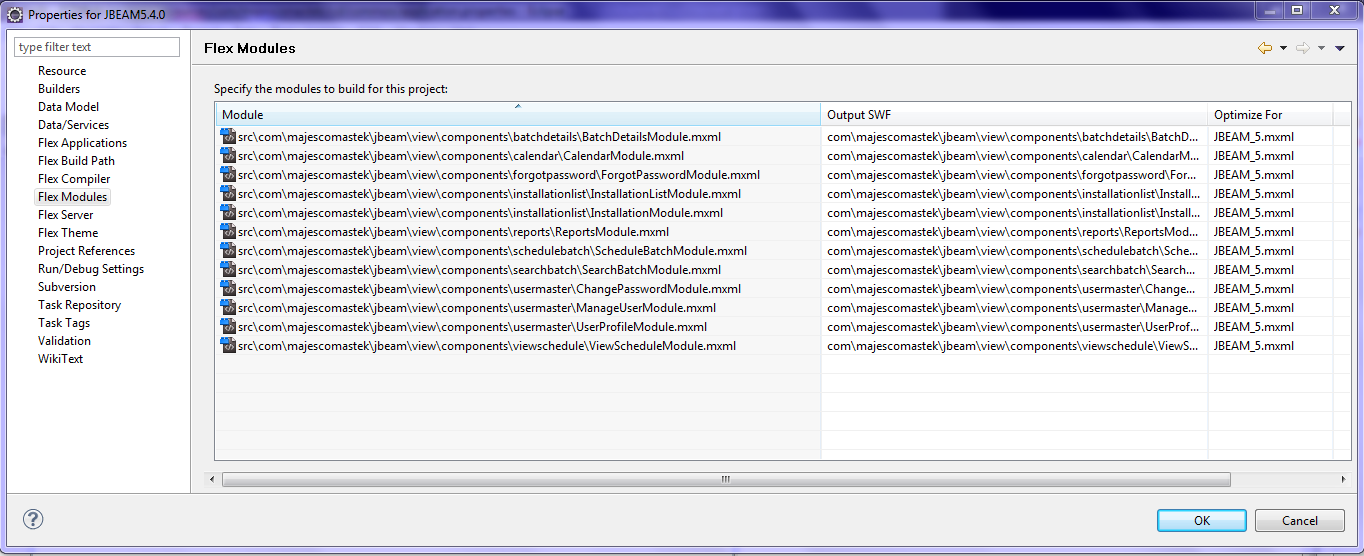
> mvn clean compile package

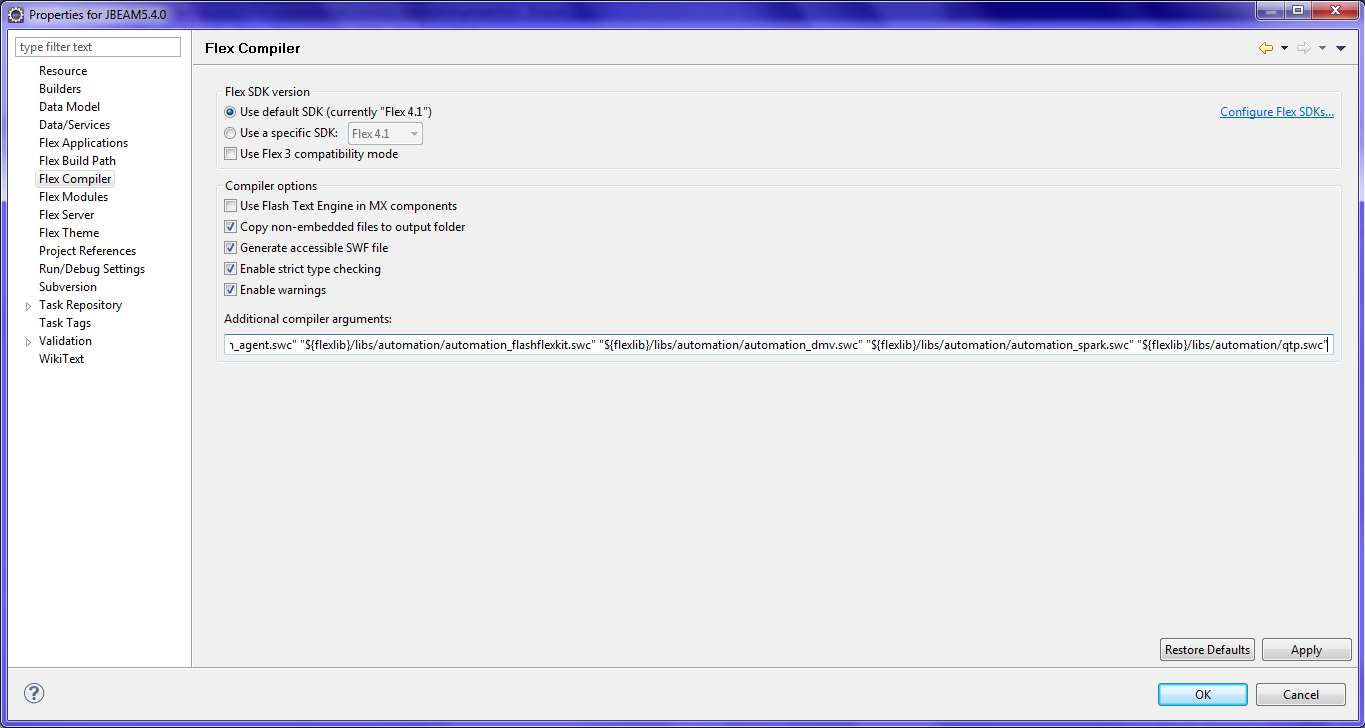
> mvn clean compile install

> mvn clean compile build

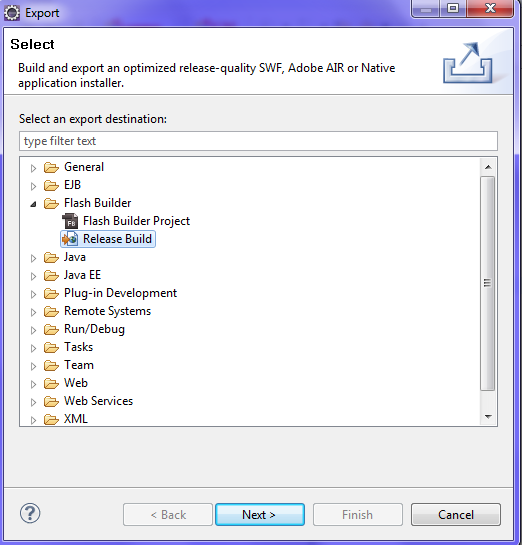
## Building JBEAM UI

1. Import JBEAM project in eclipse.
2. Right click on JBEAM project and select 'Properties'
3. Select 'Flex Modules' option and ensure that all required modules are added.

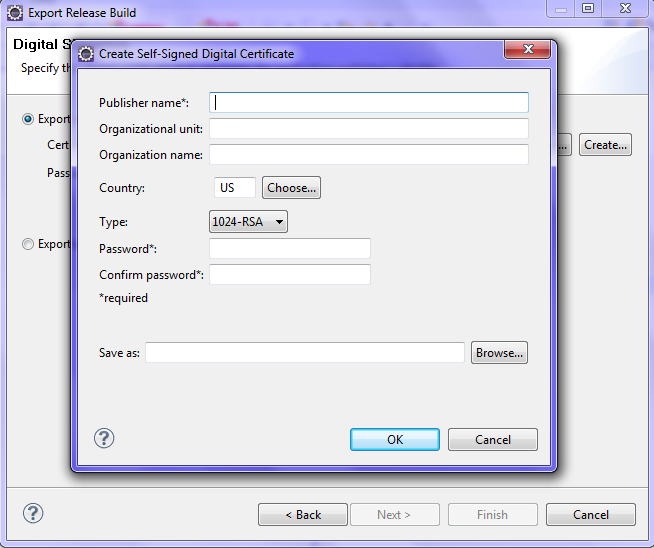




1. Clean and compile JBEAM project.
2. Right click on JBEAM project and select 'Export'
3. Select ‘Release Build’ option under Flash builder and click on Next



1. Select Export to AIR file in Export section. Provide path to export AIR file and click Next
2. Select ‘Export and sign AIR file with digital certificate’.
3. Use existing digital certificate or create new digital certificate by entering required information.



1. Click on Finish.

## Deploying components

Refer appropriate [JBEAM Installation guide](https://github.com/MastekLtd/JBEAM/tree/master/documentation/installation%20guide) for deployment instructions.

## Configuring the Components

The configuration of the components can be achieved by altering the appropriate CONFIGURATION table.

For CORE and CORE-COMM system, use the CONFIGURATION table of the CORE schema.

For MONITOR-COMM and MONITOR-SERVICES use the CONFIGURATION table of the MONITOR schema.

Below are the descriptions of each of the entries against each sub-system –

**CORE** [CODE1=’CORE’]

|  |  |  |  |
| --- | --- | --- | --- |
| **CODE2** | **CODE3** | **SAMPLE VALUE** | **DESCRIPTION** |
| BATCH\_LISTENER | MAX\_LISTENERS | 5 | The max number of listeners to be spawned for batch objects. Please ensure that the connections should also be increased with an increase in the number of listeners |
| DATE\_FORMAT | BATCH\_JOB\_DATE | **dd/MM/yyyy** | The format for the batch job date |
| DATE\_FORMAT | BATCH\_RUN\_DATE | dd-MMM-yyyy HH:mm:ss | The date format of the batch run date |
| EMAIL | CONTENT\_HANDLER | com.stgmastek.core.util.email.DefaultEmailContentGenerator | The default email content handler implementation. |
| EMAIL | NOTIFICATION | N | Indication whether to send email alerts or not |
| EMAIL | NOTIFICATION\_GROUP | **batch-operations@company.com** | The email group to which the email alerts to be sent |
| EXECUTION\_HANDLER | EV | com.stgmastek.core.logic.EventParserObjectExecutionHandler | Default execution handler implementation for EVent parser batch objects |
| EXECUTION\_HANDLER | JV | com.stgmastek.core.logic.JAVAExecutionHandler | Default execution handler implementation for JAVA batch objects |
| EXECUTION\_HANDLER | PL | com.stgmastek.core.logic.PLSQLExecutionHandler | Default execution handler implementation for PLSQL batch objects |
| EXECUTION\_HANDLER | FE | com.stgmastek.jbeam.billing.impl.FlowExecutionHandler | Default execution handler implementation for launch flow |
| FUTURE\_DATE\_RUN | MAX\_NO\_OF\_DAYS | 50000 | The max number of future days for which the batch could be run |
| GLOBAL\_PARAMETER | REQUIRED | N | Indication whether to set the global parameters. Usually would be 'Y' |
| INSTALLATION | CODE | **BILLING-DV** | The current or self-installation code. |
| INSTALLATION | NAME | **Client User Acceptance Testing** | The current or self-installation name |
| MODE | DEV\_OR\_PRE | PRE | Primarily used for development ease. Real-time would always be PRE. |
| POLLER | WAIT\_PERIOD | 5000 | The wait period for pollers in the batch core system |
| PRE | VERSION | preV1.0R28.00.D065 | The PRE version on which the batch is run. |
| PRE | WAIT\_PERIOD | 5000 | The wait period to check the status requested to the PRE engine to execute |
| PRE\_POST\_LISTENER | MAX\_LISTENERS | 5 | The max number of listeners to be spawned for PRE / POST objects. Please ensure that the connections should also be increased with an increase in the number of listeners |
| PRE\_REQUEST\_TYPE | VALUE | GENERAL | The max number of future days for which the batch could be run |
| PRE\_STUCK\_THREAD | LIMIT | 120 | The limit for stuck thread in min |
| PRE\_STUCK\_THREAD | MAX\_LIMIT | 180 | The max limit for stuck thread in min |
| REPORT\_RUNTIME | LOG\_DIR | **E:\gwcc\_qc2\jbeam\birt-runtime-2\_5\_2\logs** | Directory where the logs will reside. |
| REPORT\_RUNTIME | OUTPUT\_FOLDER | **E:\gwcc\_qc2\jbeam\PRE28\reports** | The Report Output directory. |
| REPORT\_RUNTIME | HOME\_DIR | **E:\gwcc\_qc2\jbeam\birt-runtime-2\_5\_2\ReportEngine** | The Runtime Report Engine Directory |
| SAVEPOINT | DIRECTORY | **E:\gwcc\_qc2\jbeam\savepoints\** | The savepoint file directory |

**CORE-COMM**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CODE1** | **CODE2** | **CODE3** | **VALUE** | **DESCRIPTION** |
| COMM | POLLER | WAIT\_PERIOD | 5000 | The wait period for pollers in the core communication system |
| INSTALLATION\_WS | **BILLING-DV** | SERVICES | **172.16.209.143:10001** | The <IP>:<PORT> to publish the services |
| MONITOR\_WS | MONITOR\_WS | SERVICES | **172.16.209.143:10011** | The <IP>:<PORT> of the monitor communication system. Needed for the core system to communicate |
| COMM | POLLER | WAIT\_PERIOD | 5000 | The wait period for pollers in the core communication system |

**MONITOR-COMM**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CODE1** | **CODE2** | **CODE3** | **VALUE** | **DESCRIPTION** |
| **BILLING-DV** | COLLATOR | WAIT\_PERIOD | 10000 | The collator wait period for installation BILLING-DV |
| MONITOR\_WS | OUTBOUND\_Q\_POLLER | WAIT\_PERIOD | 15000 | The monitor poller waiting period |
| INSTALLATION\_WS | **BILLING-DV** | SERVICES | **172.16.209.143:10001** | The <IP>:<PORT> of the published communication services for BILLING-DV installation |
| MONITOR\_WS | MONITOR\_WS | SERVICES | **172.16.209.143:10011** | The <IP>:<PORT> to publish the monitor communication services |

**MONITOR-SERVICES**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CODE1** | **CODE2** | **CODE3** | **VALUE** | **DESCRIPTION** |
| MONITOR\_WS | MONITOR\_UI\_WS | SERVICES | **172.16.209.143:15235** | The <IP>:<PORT> to publish the monitor services for the UI |

## Starting the Components

The below table describes the way individual components are run or started in development (DEV) or production (PRE) mode.

|  |  |  |
| --- | --- | --- |
| **Component** | **Development Mode (DEV)** | **Production / UNIX Mode (PRE)** |
| CORE | Through eclipse using the ‘com.stgmastek.core.main.StartCore’ class by passing appropriate parameters to start the batch | Process Request Engine (PRE) would be responsible for invoking and executing the batch. The PRE job, though would have to be scheduled / requested through the UI. |
| CORE-COMM | Once the configurations for the CORE communication system is set, ‘com.stgmastek.core.comm.main. StartCoreCommunication’ needs to be invoked either through eclipse or issuing a command ‘java -jar core-comm.jar’ | Shell script is bundled in the deployable namely ‘start-core-comm.sh’ for the system to be started. |
| MONITOR-COMM | Once the configurations for the MONITOR communication system is set, ‘com.stgmastek.monitor.comm.main. StartMonitorCommunication’ needs to be invoked either through eclipse or issuing a command ‘java -jar monitor-comm.jar’ | Shell script is bundled in the deployable namely ‘start-monitor-comm.sh’ for the system to be started. |
| MONITOR-SERVICES | Once the configurations for the MONITOR services system is set, ‘com.stgmastek.monitor.ws.main. StartMonitorWS’ needs to be invoked either through eclipse or issuing a command ‘java -jar monitor-services.jar’ | Shell script is bundled in the deployable namely ‘start-monitor-services.sh’ for the system to be started. |

# Automated Testing

All JBEAM modules have junit test cases defined for them. These test cases can be run as standard Junit test cases.

Also new junit test cases should be added for any new functionality or changes done.

Sample Test class which provides test cases for BatchDAO method addInstructionLog

|  |
| --- |
| /\*\*  \* JUnit class to test the BatchDAO method addInstructionLog  \*  \* **@author** grahesh.shanbhag  \*  \*/  **public** **class** TestBatchDAOAddInstructionLog **extends** TestCase{  CReqInstructionLog instructionLog = **null**;  InstructionParameters instructionParameters;      **protected** **void** setUp() **throws** Exception {  instructionLog = **new** CReqInstructionLog();    instructionLog.setSeqNo(12);  instructionLog.setMessage("BSRUNBATCH");  instructionLog.setInstructingUser("ADMIN");  instructionLog.setInstructionTime(**new** Date().getTime());      List<InstructionParameters> instructionParametersList = **new** ArrayList<InstructionParameters>();  **for**(**int** i = 1; i <= 6; i++){  instructionParameters = **new** InstructionParameters();  instructionParameters.setSlNo(i);  **switch**(i){  **case** 1:  //Insert Batch Name  instructionParameters.setName("BATCH\_NAME");  instructionParameters.setValue("NEW BATCH");  instructionParameters.setType("S");  **break**;  **case** 2:  //Insert Batch Date  instructionParameters.setName("BATCH\_DATE");  instructionParameters.setValue(String.*valueOf*(**new** Date().getTime()));  instructionParameters.setType("DT");  **break**;  **case** 3:  //Insert Batch Type  instructionParameters.setName("BATCH\_TYPE");  instructionParameters.setValue("SPECIAL");  instructionParameters.setType("S");  // System.out.println(i + "] Expected Values is Batch Type ");  **break**;  **case** 4:  //Insert Batch Type  instructionParameters.setName("POLICY\_1");  instructionParameters.setValue("POL-98789");  instructionParameters.setType("S");  // System.out.println(i + "] Expected Values is Batch Type ");  **break**;  **case** 5:  //Insert Batch Type  instructionParameters.setName("AGENCY\_2");  instructionParameters.setValue("AG-435");  instructionParameters.setType("S");  // System.out.println(i + "] Expected Values is Batch Type ");  **break**;  **case** 6:  //Insert Batch Type  instructionParameters.setName("BATCH\_END\_DATE");  instructionParameters.setValue(String.*valueOf*(**new** Date().getTime() + 60032334));  instructionParameters.setType("DT");  // System.out.println(i + "] Expected Values is Batch Type ");  **break**;  }    instructionParametersList.add(instructionParameters);    }  instructionLog.setInstructionParametersList(instructionParametersList);  // parameters.setId(2);  // parameters.setBatchName("NEW BATCH 2 IN CORE");  // parameters.setCreatedBy("ADMIN");  // parameters.setBatchStartTime(new Date().getTime());  // parameters.setCreatedOn(new Date().getTime());  // parameters.setInstructionLog("POLICY=POL011;POLICY=POL012;POLICY=POL021;ACCOUNT=AC-23213;ACCOUNT=AC-233;AGENCY=AG-23");  }        /\*\*  \* Test only with mandatory fields  \*/  **public** **void** testAddinstructionLogMandatoryFields() **throws** Exception {  IBatchDAO dao = DAOFactory.*getBatchDAO*();  Connection connection = **null**;  **try** {  connection = ConnectionManager.*getInstance*().getDefaultConnection();  Integer i = dao.addInstructionLog(instructionLog,connection);    *assertNotNull*(i);  }**finally** {  dao.releaseResources(**null**, **null**, connection);  }  }        } |

# Code snippets - Useful code samples

Some example code snippets -

## Create a Java Batch Job

A Java Batch job is created by implementing com.stgmastek.core.interfaces.IExecutableBatchJob interface. A sample code is as given below.

|  |
| --- |
| public class AgencyClearingProcess implements IExecutableBatchJob {  //Define program specific variables  private static final String AGENCY\_CLEARING\_PROCESS =  "{call AGENCY\_CLEARING.CLEARING\_PROCESS(?)}";  //Define logger  private static final transient Logger logger = Logger.getLogger(ProcessAutoWriteOffBatchJob.class); |

The method init is called by JBEAM by passing the batch context and batch object that must be acted upon.

|  |
| --- |
| public void init(BatchContext batchContext) {  //perform any activity that you may need it on a global level.  //example creating a ThreadPoolExecutor object, etc.  } |

Implement the method execute. There are few things that are mandatory and must be executed prior to executing the batch object. Namely marking the batch object as UC and then setting of global parameters only in case of Constants.SET\_GLOBAL\_PARAMETERS.

|  |
| --- |
| public void execute(BatchContext batchContext, BatchObject batchObject) throws BatchException {    CallableStatement cs = null;  String successFailure = "99";  String message = "";  CConnection connection = null;  if (logger.isInfoEnabled()) {  logger.info("Trying to execute AGENCY\_CLEARING.CLEARING\_PROCESS..");  }  IAppDao dao = DaoFactory.getAppDao();  try{  connection = batchContext.getApplicationConnection();  //First mark the status as 'UC'  dao.markUC(appConnection, batchObject);    if(Constants.SET\_GLOBAL\_PARAMETERS){  dao.setGlobalParameters(connection, batchContext, batchObject);  }    cs = connection.prepareCall(AGENCY\_CLEARING\_PROCESS);  cs.registerOutParameter(1,Types.VARCHAR);  cs.execute();  successFailure = cs.getString(1);  if("00".equals(successFailure)) {  batchObject.setStatus("CO");  }  if(!"00".equals(successFailure)) {  throw new BatchException(message);  }  } catch (SQLException e) {  logger.error("SQLException caught " + e.getMessage(), e);  throw new BatchException(e);  } finally {  dao.releaseResources(null, cs, connection);  }  } |

The java batch job is complete. Now you need to configure the same in the JBEAM. Either it can be in the form of PRE/POST job or it can be a *i*batch\_executor*/i* job.

## Creating a Batch Execution Handler

The execution handler responsibility is to execute or handle a particular type of objects. The object need not be just java or PL/SQL or event parser object but they can be of any nature. Example is say your project has got lot of shell scripts that must be executed as a part of Batch Execution. Then one can create a handler and implement the functionality. The handler gets to hold the ObjectMapDetails object that is associated with the BatchObject which otherwise is not available to the IExecutableBatchJob.

The example uses [Apache Commons Exec](http://commons.apache.org/exec/) framework. The following example shows how an Execution Handler can be written.

|  |
| --- |
| public class ShellScriptExecutionHandler extends BaseExecutionHandler {  private static final Logger logger = Logger.getLogger(ShellScriptExecutionHandler.class);  /\* (non-Javadoc)  \* @see com.stgmastek.core.logic.BaseExecutionHandler#execute(  com.stgmastek.core.util.BatchObject,  com.stgmastek.core.util.ObjectMapDetails,  com.stgmastek.core.util.BatchContext)  \*/  @Override  public BatchObject execute(BatchObject batchObject, ObjectMapDetails objectMapDetails,  BatchContext batchContext) throws BatchException {  Connection conn = null;  IAppDao aDao = DaoFactory.getAppDao();  try {  //Get the connection object  conn = batchContext.getApplicationConnection();  //Mark the object as UC  aDao.markUC(conn, batchObject);  //Create a common environment for executing the shell scripts  Map<String, String> environmentMap = new HashMap<String, String>();  environmentMap.put("SomeKey", "SomeValue");  //Get the name of the shell script from the ObjectMapDetails  String executable = objectMapDetails.getObjectName();  //Get the parameters from the batch object taskname  String[] parameters = batchObject.getTaskname().split(";");  CommandLine cmd = new CommandLine(executable);  for (String parameter : parameters) {  cmd.addArgument(parameter);  }  //Create a default executor  DefaultExecutor executor = new DefaultExecutor();  PumpStreamHandler handler = new PumpStreamHandler();  executor.setStreamHandler(handler);  //Execute the command  int returnValue = executor.execute(cmd, environmentMap);  //Identify if the return exit value equals 0  if (returnValue == 0) {  batchObject.setStatus("CO");  } else {  batchObject.setStatus("99");  }  } catch (Exception e) {  logger.fatal(e);  batchObject.setStatus("99");  } finally {  if (batchObject.getStatus() == null) {  batchObject.setStatus("99");  }  aDao.releaseResources(null, null, conn);  }  return batchObject;  }  } |

## A more complex batch job

The example given below is from coreprint batch job. The init methods initializes the Thread Pool executor service and destroy method shuts the service down.

|  |
| --- |
| public class PrintExecutorBatchJob {  /\*\*  \* The Thread Pool executor.  \*/  private ThreadPoolExecutor executor;  /\*\*  \*  \*/  private LinkedBlockingQueue<Runnable> printQueue;  /\* (non-Javadoc)  \* @see com.stgmastek.core.interfaces.IExecutableBatchJob#destroy(com.stgmastek.core.util.BatchContext)  \*/  public void destroy(BatchContext batchContext) {  executor.shutdown();  }  /\* (non-Javadoc)  \* @see com.stgmastek.core.interfaces.IExecutableBatchJob#execute(com.stgmastek.core.util.BatchContext,  \* com.stgmastek.core.util.BatchObject)  \*/  public void execute(BatchContext batchContext, BatchObject batchObject)  throws BatchException {  PreparedStatement psSelectPE = null;  ResultSet rsSelectPE = null;  CConnection con = null;  try {  con = batchContext.getApplicationConnection();  //First mark all print executor tasks as 99 as it gets updated to CO when the actual job gets executed successfully.  //Otherwise it remains in 99 status.  PrintExecutorDAO.updatePrintExecutorBatch(con, "99", new Date(), batchContext.getBatchInfo().getBatchRunDate().getTime());  psSelectPE = con  .prepareStatement(PrintExecutorDAO.SELECT\_PRINT\_EXECUTOR\_SQL);  psSelectPE.setTimestamp(1, new java.sql.Timestamp(batchContext.getBatchInfo()  .getBatchRunDate().getTime()));  rsSelectPE = psSelectPE.executeQuery();  rsSelectPE.setFetchSize(500);  while (rsSelectPE.next()) {  PrintExecutorVO vo = populate(rsSelectPE);  PrintExecutorTask task = null;  // currently this is hardcoded but can be modified later to read  // from either a database or xml.  if (!"ALW".equalsIgnoreCase(vo.getPrintType())) {  String programName = (vo.getProgramName() == null ? "REP"  : vo.getProgramName());  if (programName.toLowerCase().indexOf(".pdf") > -1) {  task = new CopyFilePETask("Copy", batchContext, vo);  } else {  task = new ExecuteReportPETask("ExecuteReport", batchContext, vo);  }  } else {  task = new AlwPETask("ALW", batchContext, vo);  }  System.out.println("Executing task " + task.getTaskInfo());  executor.execute(task);  }  batchObject.setStatus("CO");  } catch (Exception e) {  batchObject.setStatus("99");  throw new BatchException(e);  } finally {  PrintExecutorDAO.close(rsSelectPE, psSelectPE, con);  executor.shutdown();  while (executor.isTerminating()) {  System.out.println("Still not terminated.. Waiting... #" + executor.getActiveCount());  try {  TimeUnit.SECONDS.sleep(1);  } catch (InterruptedException e) {  }  }  }  }  /\* (non-Javadoc)  \* @see com.stgmastek.core.interfaces.IExecutableBatchJob#init(com.stgmastek.core.util.BatchContext)  \*/  public void init(BatchContext batchContext) {  printQueue = new LinkedBlockingQueue<Runnable>();  executor = new ThreadPoolExecutor(5, 5, 10, TimeUnit.MINUTES,  printQueue);  // new LinkedBlockingQueue());  executor  .setRejectedExecutionHandler(new ThreadPoolExecutor.CallerRunsPolicy());  executor.allowCoreThreadTimeOut(true);  }  } |

# Debugging - Helpful tips

Guide for debugging JBEAM code, this will help contributors during defect fixing.

## Refer following table for some common issues -

|  |  |  |
| --- | --- | --- |
| Complaint | Action item | Probable Reason |
| The service bus is down. | Check Monitor-services. It might be down. | 1) Database might be down.  2) Services stopped for some reason like server or database restart or somebody stopped it accidently.  3) Port might not be open |
| User Authentication failed | Check User details in USER\_MASTER of monitor schema | 1) User id is wrong  2) Password is wrong. In case user has forgotten password, ask to use Forgot Password link. If that also doesn't work, ask the ADMIN of the project to reset the password for that user.  3) The effective date is a future date (greater than database system date )  4) The expiry date is a back date (User validity in JBEAM expired). (less than database system date ) In both cases, ADMIN can reset the dates. |
|  | Check the ip and port in configuration table against MONITOR\_UI\_WS. |  |
| Batch is not running (Showing 'Loading new batch…) | Wait for some time. | This may happen if monitor-comm / core-comm/ PRE28 is down. |
|  | Check if core-comm is up. Else start it.  Alternative is to check this, one can go to INSTRUCTION\_PARAMETERS table of core schema. Find the requested Batch name / date / entities (Policy/ account etc.) in this table. If found the core-comm and monitor-comm are working fine. Else both or either of them are down. | If monitor-comm is up and running, but the core-comm is down for some reason, the sent request will throw an error and will be saved in DEAD\_MESSAGE\_QUEUE table. |
|  | Check if PRE is up. Else start it. Always check if the console shows it in RUNNING mode. | Sometimes the PRE might be started 2-3 times to avail clustering. In this case, one PRE28 will be always in RUNNING mode and others in PASSIVE mode. If the RUNNING PRE28 is killed somehow and PASSIVE is running, it will take care of the batch and starts itself in RUNNING mode. If somehow, this doesn't happen, then batch goes in unconditional loop. |
|  | On windows servers, check if any old process for the PRE is running (it might be terminated but not killed.) If found kill it and start the PRE afresh. | Same as above |
|  | Check if the indicator in BATCH\_LOCK table of core schema is 'L'. Make it 'O', if the PRE is restarted. | This happends when the running PRE instance is killed and batch is still alive. The BATCH\_LOCK table is updated with 'L' i.e. Locked.  Although in latest UI, user will come to know within few seconds. |
|  | If still the issue persists, terminate PRE and required channels and start afresh. | System might be hanged due to overload or space issue. |

## JBEAM TIPS

### *Tip 1*: Future Run Restriction

The configuration table has an entry for code1='CORE' code2='FUTURE\_DATE\_RUN' and code3='MAX\_NO\_OF\_DAYS'. The default value goes as '50000'. This means that JBEAM core can allow the batch to be executed even if the batch run date is greater than the 50000 days. Please make sure that the value is changed to 1 or maximum 2. So that the JBEAM core will throw an exception if the batch run date is greater than system date by 1 or 2 days respectively.

### *Tip 2*: Terminate the Batch in case a critical batch job fails

In the object\_map table there is a provision to set a flag indicating the JBEAM to exit in case of a batch job fails. The flag is on\_fail\_exit should be kept as Y.

Recently Rider NI team faced an issue where the outputs were generated in the output folder directly instead of generating within a date directory inside output folder. The reason was that the generate file name job was failing due to unique constraint violation. Due to this the print generation job was generating the outputs in the output folder directly. So in order to fail the subsequent jobs from executing, once the prior job fails, please make sure the object\_map is updated for field on\_fail\_exit = 'Y'.

### *Tip 3*: Create your project specific jars using ANT or Maven or any other build tool.

Automating the build process helps you to avoid human errors as well as it standardizes or ad-hers to a predefined release process. With the build automation it is possible to create a versioning of the JAR file by adding your version numbers to the MANIFEST along with creation time, etc. Sample entries from MANIFEST of JBEAM core are provided below

* Manifest-Version: 1.0
* Ant-Version: Apache Ant 1.7.0
* Created-By: 19.1-b02 (Sun Microsystems Inc.)
* Bundle-SymbolicName: core
* Bundle-Name: JBEAM Core
* Bundle-Activator: com.stgmastek.core.main.StartCore
* Bundle-Version: V3.3
* Major-Version: V3
* Minor-Version: 3
* Build-Number: 49
* Bundle-Vendor: STGMastek
* Bundled-By: kedar460043
* Bundled-On: 20110617 16:49:36
* Main-class: com.stgmastek.core.main.StartCore

Thus it is then possible to find when the jar was created, who created it, what were the Major-Minor version as well as the build number of the JAR. So please make sure that you use such ANT or MAVEN or any other build automation tool to package your classes (core print and jbeam-impl) into JAR files to ease the confusion around the creation timestamp of the jar file.

### *Tip 4*: Speed up the waiting time between the Processor and Listener

JBeam uses PRE (Process Request Engine) as an executor/scheduler. The value of the property waitinterval in file pr.properties should be lowered to decrease the wait period between two scans for queued requests.

# Appendix

## Database Design & Table Structure

**Core**

|  |
| --- |
| **Table Name: BATCH**  **Note: The batch table with all batch related attributes**  **A batch is unique combination of batch number and batch revision number and not just the batch number.** |
| |  |  | | --- | --- | | **Column** | **Description** | | BATCH\_NO | The batch number. *Use BATCH\_SEQ for fetching the next value for the batch number.* | | BATCH\_REV\_NO | The batch revision number. | | BATCH\_NAME | The batch name. Optional. If provided then would be used else a system default generated name would be used | | BATCH\_TYPE | The batch type.  SPECIAL – When the batch is run for a user chosen combination of entities  DATE – When the batch is either scheduled or the user chooses to run it for a date.  Note: All scheduled batches are DATE type batches. | | BATCH\_END\_REASON | The reason for which the batch ended. The reasons could be either one of –  BATCH\_COMPLETED – The batch has completed all its activities.  USER\_INTERRUPTED – When the user chooses to stop a running batch  END\_OF\_TIME – When the time allotted for the batch is exhausted.  BATCH\_FAILED – When an object marked as on-fail-exit = ‘Y’ fails there by halting the batch execution.  PRE\_ISSUED\_STOP – When PRE decides to bring down the all the jobs executing under PRE because of some fault or user requests. | | EXEC\_START\_TIME | The start time for the batch. Ideally this is the time when PRE picks up the job and the batch starts its proceedings. This will be System Time and not the database time if they differ (that is in case where database is set to Fixed Date). | | EXEC\_END\_TIME | The end time for the batch. This will be System Time and not the database time if they differ (that is in case where database is set to Fixed Date). | | BATCH\_START\_USER | The user that initiated the batch. | | BATCH\_END\_USER | The user that stopped the batch. | | PROCESS\_ID | The PRE process identifier that is associated with the current batch | | BATCH\_END\_REASON | The reason to end the batch. It can be BATCH\_COMPLETED, USER\_INTERRUPTED, | | FAILED\_OVER | The flag which will be updated if one PRE crashes and another PRE will continue processing of batch. | | INSTRUCTION\_SEQ\_NO | The unique sequence number from the instruction log. This instruction sequence number has to be the same as that when issued from the MONITOR system for conformance. No sequence is or would be needed for this sequence number. | |

|  |
| --- |
| **Table Name: BATCH\_LOCK**  **Note: The batch lock table. This table is to avoid two batches running on the same environment.** |
| |  |  | | --- | --- | | **Column** | **Description** | | REQ\_ID | The PRE request identifier that has started the batch. | | LOCK\_TIME | The time at which the batch has locked the | | INDICATOR | An indicator whether –  ‘L’ocked for execution  ‘O’pen for execution | |

|  |
| --- |
| **Table Name: COLUMN\_MAP**  **Note: Important table for the batch proceedings. This table is used for –**   1. **Setting the execution order for the batch** 2. **Deciding on the columns to look into while building the query for assignment and execution** 3. **Decides whether the batch should mark all object for the same entity-value if one of them fails.** |
| |  |  | | --- | --- | | **Column** | **Description** | | ENTITY | The entities in the batch.  EX: PRE, POLICY, POST etc.  Note: GENERAL is optional and is a provision for those objects that do not fall into other entities defined. So in the above case, if !PRE and !POLICY and !POST then others fall into the GENERAL. | | LOOKUP\_COLUMN | The primary lookup column | | LOOKUP\_VALUE | The primary lookup value column, could be null in which case the value column would be used | | VALUE\_COLUMN | The value column. This should contain the field name of the column from batch executor. The where clause will be calculated based on the value in this field. If you want to associate two or more fields then these must be separated by a # sign. Also, remember that a similar change is also required in the monitor schema for the same table. | | PRECEDENCE\_ORDER | The execution order. If POLICY has precedence 2 and ACCOUNT has 3, then POLICY as an entity would be executed before ACCOUNT. ACCOUNT would have to wait for the entire POLICY execution to be completed. | | ON\_ERROR\_FAIL\_ALL | ‘Y’ to mark all other ‘similar’ objects as suspended.  ‘N’ or null otherwise  EX. There are 10 records in for POLICY P1.  1 through to 3 have executed successfully. 4th has failed.  If ON\_ERROR\_FAIL\_ALL = ‘Y’, then the status would be –  1 to 3 = CO  4 = ‘99’  5 to 10 = ‘SP’  If ON\_ERROR\_FAIL\_ALL = ‘N’, then the status would be –  1 to 3 = CO  4 = ‘99’  5 to 10 = depends upon the execution status of individual objects | |

|  |
| --- |
| **Table Name: CONFIGURATION**  **Note: Table to set the configurations for the batch core system.** |
| |  |  | | --- | --- | | **Column** | **Description** | | CODE1 | Defines the first level configuration value | | CODE2 | Defines the second level configuration value | | CODE3 | Defines the third level configuration value | | VALUE | The configuration value | | VALUE\_TYPE | The configuration value type  S – String  I – Integer  D – Date | | DESCRIPTION | The description for each of the configuration item. | |

|  |
| --- |
| **Table Name: DEAD\_MESSAGE\_QUEUE**  **Note: Table where all message that could not be processed would fall** |
| |  |  | | --- | --- | | **Column** | **Description** | | ID | The identifier of the message | | I\_O\_MODE | Inbound or Outbound message | | MESSAGE | The message | | PARAM | The message parameters, if any. | | ERROR\_DESCRIPTION | The exception stack trace as to why the processing of the message failed. | |

|  |
| --- |
| **Table Name: I\_QUEUE**  **Note: Table where all IN bound messages should fall for processing** |
| |  |  | | --- | --- | | **Column** | **Description** | | ID | The unique identifier for the message. *Use I\_QUEUE\_SEQ for fetching the next value.* | | MESSAGE | The actual message | | PARAM | The parameters for the message, if any | |

|  |
| --- |
| **Table Name: INSTRUCTION\_LOG**  **Note: Table where all instructions for the batch core would be stored.** |
| |  |  | | --- | --- | | **Column** | **Description** | | SEQ\_NO | The unique sequence number for the instruction log. This instruction sequence number has to be the same as that when issued from the MONITOR system for conformance. No sequence is or would be needed for this sequence number. | | BATCH\_NO | The batch number. It is not mandatory that the batch number and revision number would always exist, as there could be an instruction to start a batch e.g. BSRUNBATCH and there would not be any batch number to assign. It would eventually be updated with the batch and revision number. | | BATCH\_REV\_NO | The batch revision number | | MESSAGE | The actual message | | MESSAGE\_PARAM | The message parameters | | INSTRUCTING\_USER | The instructing user | | INSTRUCTION\_TIME | The instruction time | | BATCH\_ACTION | The batch core action on the instruction. Updated once the batch core acts on the instruction. | | BATCH\_ACTION\_TIME | The batch core action time on the instruction | |

|  |
| --- |
| **Table Name: INSTRUCTION\_PARAMETERS**  **Note: Table where all the instruction parameters for an instruction for the batch core would be stored.** |
| |  |  | | --- | --- | | **Column** | **Description** | | INSTRUCTION\_LOG\_NO | The instruction log sequence number | | SL\_NO | The serial number of the parameter | | NAME | The name of the parameter | | VALUE | The value of the parameter | | TYPE | The type in which the parameter value should be treated | |

|  |
| --- |
| **Table Name: LOG**  **Note: Table where the object execution details would be stored as log files. Most fields are borrowed from JOB\_SCHEDULE table and would contain data as it is from the JOB\_SCHEDULE table. Therefore it is important that any change in column length or addition of column in JOB\_SCHEDULE make sure that the same modification is done in this table from both CORE as well as MONITOR schema.** |
| |  |  | | --- | --- | | **Column** | **Description** | | SEQ\_NO | The unique sequence number for the log entry. Uses LOG\_SEQ for fetching the sequence number. | | BATCH\_NO | The batch number that executed the object | | BATCH\_REV\_NO | The batch revision number | | BE\_SEQ\_NO | The JOB\_SCHEDULE sequence number | | TASK\_NAME | The JOB\_SCHEDULE task name | | OBJ\_EXEC\_START\_TIME | The execution started time for the object. This will be used as per the configuration done in PRE to either make use of Database date time or SERVER date time. | | OBJ\_EXEC\_END\_TIME | The execution ended time for the object. This will be used as per the configuration done in PRE to either make use of Database date time or SERVER date time. | | STATUS | The JOB\_SCHEDULE status | | SYS\_ACT\_NO | The JOB\_SCHEDULE system activity number | | USER\_PRIORITY | The JOB\_SCHEDULE user priority | | PRIORITY\_CODE1 | The JOB\_SCHEDULE priority code 1 | | PRIORITY\_CODE2 | The JOB\_SCHEDULE priority code 2 | | PRE\_POST | Identifies whether the object is of type PRE (to be executed prior to the actual batch objects) or POST (after the execution of the batch objects have been executed) | | JOB\_TYPE | The JOB\_SCHEDULE job type | | LINE | The JOB\_SCHEDULE line | | SUBLINE | The JOB\_SCHEDULE sub line | | BROKER | The JOB\_SCHEDULE broker | | POLICY\_NO | The JOB\_SCHEDULE policy number | | POLICY\_RENEW\_NO | The JOB\_SCHEDULE policy renew number | | VEH\_REF\_NO | The JOB\_SCHEDULE vehicle reference number | | CASH\_BATCH\_NO | The JOB\_SCHEDULE cash batch number | | CASH\_BATCH\_REV\_NO | The JOB\_SCHEDULE cash batch revision number | | GBI\_BILL\_NO | The JOB\_SCHEDULE GBI bill number | | PRINT\_FORM\_NO | The JOB\_SCHEDULE print form number | | NOTIFY\_ERROR\_TO | The JOB\_SCHEDULE notify error to | | DATE\_GENERATE | The JOB\_SCHEDULE date generated | | GENERATE\_BY | The JOB\_SCHEDULE generated by | | REC\_MESSAGE | The JOB\_SCHEDULE recorded message | | JOB\_DESC | The JOB\_SCHEDULE job description | | OBJECT\_NAME | The JOB\_SCHEDULE object name | | DATE\_EXECUTED | The JOB\_SCHEDULE date executed | | LIST\_IND | The listener identifier that executed the batch job | | ENTITY\_TYPE | The JOB\_SCHEDULE entity type | | ENTITY\_CODE | The JOB\_SCHEDULE entity code | | REF\_SYSTEM\_ACTIVITY\_NO | The JOB\_SCHEDULE reference system activity number | | ERROR\_TYPE | The error type, if any, if status = ‘99’ | | ERROR\_DESCRIPTION | The error description, stack trace, if any, if status = ‘99’ | | CYCLE\_NO | The cycle number as per the batch progress | | USED\_MEMORY\_BEFORE | The memory available before starting the batch | | USED\_MEMORY\_AFTER | The memory available after completing the batch | |  |  | |

|  |
| --- |
| **Table Name: META\_DATA**  **Note: The table holds the configurations or the set up information for the PRE / POST events or jobs. These jobs would be procreated as needed into the JOB\_SCHEDULE table and then would be executed as normal batch objects, though would yet be identified as PRE / POST jobs.** |
| |  |  | | --- | --- | | **Column** | **Description** | | SEQ\_NO | The unique sequence number. As it is a setup table, it does not use any sequence. A new job would have MAX SEQ\_NO + 1 as its sequence number. | | TASK\_NAME | The task name for the PRE / POST object | | EFF\_DATE | The effective date for the PRE / POST object | | EXP\_DATE | The expiry date for the PRE / POST object | | ON\_FAIL\_EXIT | This functionality has been moved to Object Map. As any job registered in the Meta Data table must have an entry in object map therefore the on fail exit from Object Map is taken into consideration and this field here is ignored. | | PRIORITY\_CODE1 | The priority code 1.  The batch can run any PRE / POST event in parallel or as dependent objects. The priority code 1 plays an important here.  EX: There are in all 10 PRE objects. There are three marked as PRIORITY\_CODE1 = 1. Five other objects are marked as 2 and the remaining two are marked as 3. Then those with PRIORITY\_CODE1 = 1 would be picked up first and executed in parallel. ONLY ONCE the execution of these three objects are complete, would those with PRIORITY\_CODE1 = 2 picked up for execution. The cycle continues till there are no more objects to be picked up. | | PRIORITY\_CODE2 | Inspiration from JOB\_SCHEDULE table and retained for future probable needs. | | PRE\_POST | Identification whether the configured object is a PRE or a POST event. | | JOB\_TYPE | The job type for the object.  EV – Event Parser  JV – Java  PL – PLSQL  FE - Flow Execution | | LINE | The line | | SUBLINE | The sub line | | DATE\_GENERATE | The date generated | | GENERATE\_BY | The generated by | | JOB\_DESC | The job description, if any | | OBJECT\_NAME | The object name. | |

|  |
| --- |
| **Table Name: O\_QUEUE**  **Note: Table where all OUT bound messages should fall for processing** |
| |  |  | | --- | --- | | **Column** | **Description** | | ID | The unique identifier for the message. *Use I\_QUEUE\_SEQ for fetching the next value.* | | MESSAGE | The actual message | | PARAM | The parameters for the message, if any | |

|  |
| --- |
| **Table Name: OBJECT\_MAP**  **Note: Mapping table that maps the object name with the actual object to be executed** |
| |  |  | | --- | --- | | **Column** | **Description** | | ID | The JOB\_SCHEDULE object name (in upper case) | | OBJECT\_NAME | The actual object to be invoked or executed with the super set of the parameter list (including those having default values)  EX: There exist a stored procedure with signature  SOME\_PKG.SOME\_SP(  Id IN number,  Name IN varchar2,  DOB IN date,  Department IN varchar2 default ‘ADMIN’  )  then this field would have  SOME\_PKG.SOME\_SP(:N,:VC,:DT, :VC) | | OBJECT\_TYPE | The type for the object.  EV – Event Parser  JV – Java  PL – PLSQL  FE – Flow Execution | | EFF\_DATE | The effective date | | EXP\_DATE | The expiry date | | DEFAULT\_VALUES | Optional. Provision to provide default value in case the parameter values are not supplied at run time from JOB\_SCHEDULE.BE\_TASK\_NAME. | | ON\_FAIL\_EXIT | Indication whether to halt / stop the batch if the execution fails.  ‘Y’ – stop the batch  ‘N’ or null – continue with batch proceedings | | ON\_FAIL\_EMAIL | Indication whether to send email if the batch execution fails.  ‘Y’ – send email  ‘N’ or null – do not send email | | MIN\_TIME | The minimum time an object should take to execute | | AVG\_TIME | The average time an object should take to execute | | MAX\_TIME | The maximum time an object should take to execute | | MIN\_TIME\_ESCL | Indication whether to send email if the object takes less than minimum time to execute  ‘Y’ – send email  ‘N’ or null – do not send email | | ESCALATION\_LEVEL | The level of escalation (HIGH / MEDIUM / LOW) | | CASE\_DATA | The YAWL request (For OBJECT\_TYPE = ‘FE’) | |

|  |
| --- |
| **Table Name: PROGRESS\_LEVEL**  **Note: Table that records the progress level for a batch** |
| |  |  | | --- | --- | | **Column** | **Description** | | BATCH\_NO | The batch number | | BATCH\_REV\_NO | The batch revision number | | INDICATOR\_NO | The progress level indicator number | | PRG\_LEVEL\_TYPE | The progress level type or the type of entity being worked upon.  EX: PRE, POLICY, POST etc. | | PRG\_ACTIVITY\_TYPE | The progress activity type. It could be one of  EX:  INITIALIZATON – Initialization of the batch  EXECUTION ORDER – Setting up of the execution order for the batch  PROCREATION – Procreating PRE / POST events  ASSIGNMENT – Assigning of batch objects for an entity [Iterates in cycles]  SCHEDULING – Scheduling of the assigned batch objects for an entity [Iterates in cycles]  EXECUTION – Execution of the scheduled batch objects for an entity [Iterates in cycles]  CLOSURE – Closing of the batch | | CYCLE\_NO | The cycle number for the current iteration | | STATUS | The status | | START\_DATETIME | The start time for the progress level activity | | END\_DATETIME | The end time for the progress level activity | | ERROR\_DESC | Error stack trace, if any, if status = ‘99’ | | FAILED\_OVER | The flag which will be updated if one PRE crashes and another PRE will continue processing of batch. This is possible with Terracota installation and configuration. | |

|  |
| --- |
| **Table Name: SYSTEM\_INFO**  **Note: Table that records the system / environment information on which the batch is run** |
| |  |  | | --- | --- | | **Column** | **Description** | | BATCH\_NO | The batch number | | BATCH\_REV\_NO | The batch revision number | | JAVA\_VERSION | The JAVA version on which the batch is run | | PRE\_VERSION | The PRE version on which the batch is run | | OS\_CONFIG | The Operating system information | | OUTPUT\_DIR\_PATH | The output directory used during the batch proceedings | | OUTPUT\_DIR\_FREE\_MEM | The free secondary storage capacity output directory had before the execution | | MAX\_MEMORY | The maximum memory available on the system | | USED\_MEMORY | The used memory for executing batch | |

**Monitor**

Most tables are essentially replica of their counter part from the core system to retain the data transmitted. Additionally each table has the installation code to identify the transmitting installation. Only those new ones pertaining only to Monitor database are mentioned below

|  |
| --- |
| **Table Name: GRAPH\_DATA\_LOG**  **Note: Table that collates the data for the graph shown in the UI** |
| |  |  | | --- | --- | | **Column** | **Description** | | INSTALLATION\_CODE | The installation code | | GRAPH\_ID | The graph id (GraphPlotter / FailedObjectsPieChartCollator) | | BATCH\_NO | The batch number | | BATCH\_REV\_NO | The batch revision number | | COLLECT\_TIME | The time taken for the objects to execute | | GRAPH\_X\_AXIS | The object name | | GRAPH\_Y\_AXIS | Another item if required to display in graph | | GRAPH\_VALUE |  | |

|  |
| --- |
| **Table Name: INSTALLATION**  **Note: Installation master table** |
| |  |  | | --- | --- | | **Column** | **Description** | | INSTALLATION\_CODE | The installation code | | INSTALLATION\_DESC | The installation description | | EFF\_DATE | The effective date for the installation | | EXP\_DATE | The expiry date for the installation | | CREATED\_ON | Created on | | CREATED\_BY | Created by | | MODIFIED\_ON | Modified on | | MODIFIED\_BY | Modified by | | BATCH\_NO | The current batch number | | BATCH\_REV\_NO | The current batch revision number | | TIMEZONE\_ID | The timezone id as per the geographical position | |

|  |
| --- |
| **Table Name: USER\_INSTALLATION\_ROLE**  **Note: The master table for user installation and role mapping.** |
| |  |  | | --- | --- | | **Column** | **Description** | | USER\_ID | The user id | | INSTALLATION\_CODE | The installation code which is assigned to an user. If the role is ADMIN or CONNECT, the installation code will be ‘null’ | | ROLE\_ID | The role which is assigned to an user for an installation. | |

|  |
| --- |
| **Table Name: USER\_MASTER**  **Note: User master table** |
| |  |  | | --- | --- | | **Column** | **Description** | | USER\_ID | The user id | | USER\_NAME | The user name | | TELEPHONE\_NO | The contact number | | FAX\_NO | The fax number | | EMAIL\_ID | The email id. Emails will be sent to this email id in following cases:   1. A new user created (email with user name and password 2. Administrator resets the password. 3. Password retrieved using ‘Forgot Password’ facility | | EFF\_DATE | The effective date for the user. | | EXP\_DATE | The expiry date for the user. | | CREATED\_ON | The created on date for the user. | | CREATED\_BY | The user\_id of the user who creates the particular user | | PASSWORD | The password of the user. It will be always in encrypted format. | | FORCE\_PASSWORD\_FLAG | The flag to indicate if the user needs to change the password forcefully.  Indication whether to change the password on screen  ‘Y’ – User needs to change password  ‘N’ - User does not need to change password | | MODIFIED\_BY | The user\_id of the user who changes the user data on screen | | MODIFIED\_ON | The modified on date for the user. | | HINT\_QUESTION | The user has to set the hint question in User Profile screen. If this question is set, it will help in retrieving the password, in case user forgets it. If it is not set and user wants to retrieve password, then it won’t be possible for the user. In this case, only ADMIN can reset the password. The user can change the hint question any time from Edit Profile screen. | | HINT\_ANSWER | The user has to set the hint answer in User Profile screen. If this answer is set, it will help in retrieving the password, in case user forgets it. If it is not set and user wants to retrieve password, then it won’t be possible for the user. In this case, only ADMIN can reset the password. The user can change the hint answer any time from Edit Profile screen. | | ADMIN\_ROLE | Indication whether the user is ADMINSTRATOR  ‘Y’ – User with administrative rights  ‘N’ - User without administrative rights  The user with administrative rights  An user with ADMIN role has following capabilities:   1. Create new user with roles and installations 2. Edit user with roles and installations 3. Reset password 4. To give or remove access to JBEAM UI | | CONNECT\_ROLE | Indication whether the user can connect to JBEAM UI  ‘Y’ – User can access JBEAM application on particular server  ‘N’ - User cannot access JBEAM application on particular server  The user with ADMIN role can decide whether to give the CONNECT role to a particular user or not. | | DEFAULT\_VIEW | Default view that the user selects upon initial login. Valid values are POD\_VIEW and LIST\_VIEW. | |

## Create a new communication message

Steps to introduce a new message or pass information through the communication channel

EX: New batch related information has to be transmitted from CORE to MONITOR

1. Do the normal insert into the CORE database as one would normally do through the CORE system.
2. Introduce a trigger upon insert / update to insert into O-QUEUE of the CORE database passing parameters to uniquely identify the newly inserted record in step 1.
3. Create a new service in the MONITOR-COMM to receive the transmitted information.
4. Create the stubs of the services published in MONITOR-COMM and store it in CORE-COMM project under ‘com.stgmastek.monitor.comm.client’
5. Create a new message handler class in the CORE-COMM system implementing ‘IOutboundMessageProcessor’ that fetches the client stubs and calls the service published in MONITOR-COMM system.
6. Register the handler in MessageConstants or the CORE-COMM system.
7. At MONITOR-COMM end, once the transmitted data is received use as needed.

Existing example - messagehandlers/TransmitBatchDetails.

## Create new / Override email content

The steps to create new email content / override the default email content

1. Create a class implementing ‘com.stgmastek.core.util.email.IEmailContentGenerator’
2. Configure it in the CONFIGURATION table
3. Compile and set the classpath accordingly for the system to pick it up during runtime
4. Execute the batch as one would normally do

## Create / Override batch job type / execution handler

There are three identified batch job types.

* PL – PLSQL batch jobs
* EV – Event Parser batch jobs
* JV – Java batch jobs
* FE – YAWL Flow batch jobs

The default implementations for these job types are already included in the bundle. The following are the hints one can use to create a new job type.

1. Create a class extending ‘com.stgmastek.core.logic.BaseExecutionHandler’
2. Configure it in CONFIGURATION table of the CORE database [One might want to create a new job type / override the default implementation. The configurations would have to be altered appropriately. One can add a new record for new job type on similar lines as done for JV/EV/PL/FE]
3. Run the batch as one would normally do

## Enhance text logging

The text logging can be enhanced through the Aspect J class ‘com.stgmastek.core.aspects.Logging’.

All methods that are annotated as ‘@Log’ would be picked up for logging. The logging Aspect J class is naïve and needs revamp as per the wants of the implementation. If new methods needs logging, they have to be marked as ‘@Log’ as others.

## Create new PRE / POST events

New PRE/POST jobs can be created as per the implementation requirements. These jobs could be either of types PL, JV or EV. [If a new job type is introduced, a new execution handler would have to be associated as mentioned in the section ‘Create /Override batch job type / execution handler’. Once the decision is made about the job type, insert a new record into the META\_DATA table of the CORE database with appropriate parameters and details as others. It would be batch processors responsibility of procreating these meta events into the BATCH\_EXECUTOR table as needed. The job type if ‘JV’ would have to be set in the class path during runtime. The following are the step by step guide in implementing the same where we have identified that the job is of JV type.

1. Create the wrapper java class that implements IExecutableBatchJob.
2. Package it in either jbeam-impl.jar or any other jar that suites the purpose and deploy the jar in PRE28/lib.
3. Make the entry of this object in META-DATA table as below.

(In META-DATA you will need to check for the max priority code1 and provide the next priority code1 to the new object in case it is to be processed SEQUENTIALLY otherwise to execute them in parallel then you may assign the same priority code1 from the existing entry where you need them to be executed in parallel.)

Insert into META\_DATA (SEQ\_NO,TASK\_NAME,EFF\_DATE,EXP\_DATE,ON\_FAIL\_EXIT,PRIORITY\_CODE1,PRIORITY\_CODE2,PRE\_POST,JOB\_TYPE,LINE,SUBLINE,DATE\_GENERATE,GENERATE\_BY,JOB\_DESC,OBJECT\_NAME) values ('4','com.stgmastek.jbeam.billing.impl.ProcessOutboundInterfaces',to\_date('01-JAN-2000 00:00:00','DD-MON-YYYY HH24:MI:SS'),null,'N',2,0,'POST','JV','AA','AA',to\_date('01-JAN-2000 00:00:00','DD-MON-YYYY HH24:MI:SS'),'ADMIN','PROCESS OUTBOUND INTERFACES','PROCESSOUTBOUNDINTERFACES');

1. Make the entry of this object in OBJECT\_MAP table as below.

Insert into OBJECT\_MAP (ID,OBJECT\_NAME,OBJECT\_TYPE,EFF\_DATE,EXP\_DATE,DEFAULT\_VALUES,ON\_FAIL\_EXIT,ON\_FAIL\_EMAIL,MIN\_TIME,AVG\_TIME,MAX\_TIME,MIN\_TIME\_ESCL,ESCALATION\_LEVEL) values ('PROCESSOUTBOUNDINTERFACES','com.stgmastek.jbeam.billing.impl.ProcessOutboundInterfaces','JV',to\_date('01-JAN-2000 00:00:00','DD-MON-YYYY HH24:MI:SS'),null,null,null,'Y',1000,null,120,null,null);

NOTE: The above inserts are provided as a sample basis and the values are all example values. Please correct them as per your needs.