# *Java - Batch Execution & Monitoring*

## JBEAM-Manual

Document Version [5.6.0] and Date [11/13/2014 3:25 PM]

By Shantanu Charpe

Shantanu.Charpe@mastek.com

## Contents

## Document Information

| Document Name: | JBEAM-5.6.0 Manual |
|---|---|

### Document Revision History:

| Sr. No. | Version | Release Date | Author | Reviewer | Reason for change |
|---|---|---|---|---|---|
| 1 | 0.1 | 12/01/2009 | Grahesh | | Initial Version |
| 2 | 1.0 | 08/09/2010 | Mandar Vaidya | Kedar Raybagkar | Updated Version |
| 3 | 2.0 | 07/12/2010 | Mandar Vaidya | Kedar Raybagkar | Updated Version |
| 4 | 2.0.1 | 11/01/2011 | Kedar Raybagkar | | Added Pro*C and JBatch conversion section. |
| 5 | 3.0 | 05/04/2011 | Kedar Raybagkar | | |
| 6 | 4.0.5 | 16/01/2012 | Kedar Raybagkar | | |
| 7 | 5.0.7 | 10/05/2013 | Mandar Vaidya | Kedar Raybagkar | For ICD changes |
| 8 | 5.6.0 | 13/11/2014 | Shantanu Charpe | Kedar Raybagkar | |

### Document Revision Compatibility History:

| Sr. No. | Version | Release Date | Core Version | Core-Comm Version | Monitor-Comm Version | Monitor-Services Version |
|---|---|---|---|---|---|---|
| 1 | 0.1 | 12/01/2009 | 1.0 | --- | --- | --- |
| 2 | 1.0 | 08/09/2010 | 3.0 | 1.0 | 1.0 | 1.0 |
| 3 | 2.0 | 07/12/2010 | 3.2 | 1.2 | 1.2 | 2.0 |
| 4 | 2.0.1 | 11/01/2011 | 3.2 | 1.2 | 1.2 | 2.0 |
| 5 | 3.0 | 05/04/2011 | 3.3 | 3.3 | 3.3 | 3.3 |
| 6 | 4.0.5 | 16/01/2012 | 4.0.5 | 4.0.5 | 4.0.5 | 4.0.5 |
| 7 | 5.0.7 | 10/05/2013 | 5.0.7 | 5.0.7 | 5.0.7 | 5.0.7 |
| 8 | 5.6.0 | 13/11/2014 | 5.6.0 | 5.6.0 | 5.6.0 | 5.6.0 |

## Purpose

The purpose of the document is to outline –

1. Pre-requisites
2. Requirements
3. System Design & Architecture
4. Software Context & Deliverables
5. Communication Flow
6. Important Classes and its significance
7. Future probable requirements and approach
8. Database Design

## Pre-Requisites

### Resources / Developer

The resources or the developer working for the maintenance and enhancement to have the following skill set –

| **Technology / Skills** | **Level**<br>Junior<br>Mid-Level<br>Senior |
|---|---|
| JAVA 1.6 | Mid-Level |
| FLEX (AIR) | Senior |
| AOP (Aspect J) | Mid-Level |
| Web Services (CXF) | Mid-Level |
| ANT | Junior |
| Database ORACLE | Mid-Level |
| Process Request Engine (PRE) | Mid-Level |
| Eclipse (Galileo) | Mid-Level |
| FLEX Developer | Mid-Level |
| SVN | Mid-Level |

### Deployment

#### Core

| **Environment** | **Developed** | **Minimum** |
|---|---|---|
| Sun JAVA | 1.6.0.14 | 1.5+ |
| Apache CXF (included in the bundle) | 2.1.3 | 2.1.3 |

| | | |
|---|---|---|
| Aspect J (included in the bundle) | 1.6.4 | 1.6.4 |
| Process Request Engine | 27.01 | 27.01 |

**Core Communication / Monitor Communication / Monitor Services**

| Environment | Developed | Minimum |
|---|---|---|
| Sun JAVA | 1.6.0.14 | 1.5+ |
| Apache CXF (included in the bundle) | 2.1.3 | 2.1.3 |
| Aspect J (included in the bundle) | 1.6.4 | 1.6.4 |

**User Interface**

| Environment | Developed | Minimum |
|---|---|---|
| Adobe AIR Installer | 1.5.2 | 1.5.2 |
| Adobe Flash Player | 10 | 10 |

**Database**

| Environment | Developed | Minimum |
|---|---|---|
| ORACLE | 10.2.0.2 | 10.2.0.2 |

**Configuring Eclipse**

Below are the steps one has to follow for configuring any of the sub-systems. The below example is for CORE. The same can be repeated for other sub-systems.

1. Use Eclipse 3.x, ideally 3.5 (Galileo).
2. Install Aspect J plug-in for 3.5
3. Install SVN plug-in
4. Create new Aspect J project with JRE set as 1.6
5. Right Click on the project → Team → Configure SVN
6. Configure the SVN properties. Locate 'Core' project. [*http://172.16.209.156:8080/svn/jbeam/Product_Base trunk/Projects/Batch/Code/Java/Core]*
7. Choose desired project from branches / trunk.
8. Set the build path and include all the JAR files in the build path.
9. Build the project

## Requirements

The requirements framed for the new system is, in reality, the limitations of the prevailing system. The requirement also includes enhancement and have been designed with an eye on having provisions for new ‘unknown’ requirements in the future.

### Prevailing system and drawbacks

In reality, there is no one existing system that can be considered as the base system. The legacy system was developed in PRO*C and later it was rewritten using the JAVA technology, which in itself has become a legacy code. As and when the need arise for the system (either PRO*C or JAVA) in new implementations, it is then modified directly in the source code as used. This has resulted in a system that can not be used ‘as it is’ and no separation between core processing and implementation specific logics or requirements. The drawbacks of following such a system are manifold and can be summarized as below.

1. The CORE system cannot be used ‘as it is’. The implementation specific logics and needs are so hard wired into the system and that one cannot separate those and use just the processing system. The code has hence become uncategorized and extremely difficult to understand and maintain. The execution logic has (over the period of time) becomes so complex that routines have thousands of line of code.
2. OOP is lost. The components are not structured and hence are less flexible with new requirements or changes.
3. Logging is done in data files that are difficult to understand and hence it is less useful to build historical statistics that could be used to better the system and overall batch processing efficiency. It becomes difficult to build a ‘Monitor User Interface (UI)’ over such data files to give the system ‘online’ kind of behavior.
4. Documenting such a system is extremely difficult resulting in lack of knowledge transfer to new project implementation.
5. The pre-processing logic and post-processing logic is a part of the main execution unit there by cannot be configured and changed at runtime.

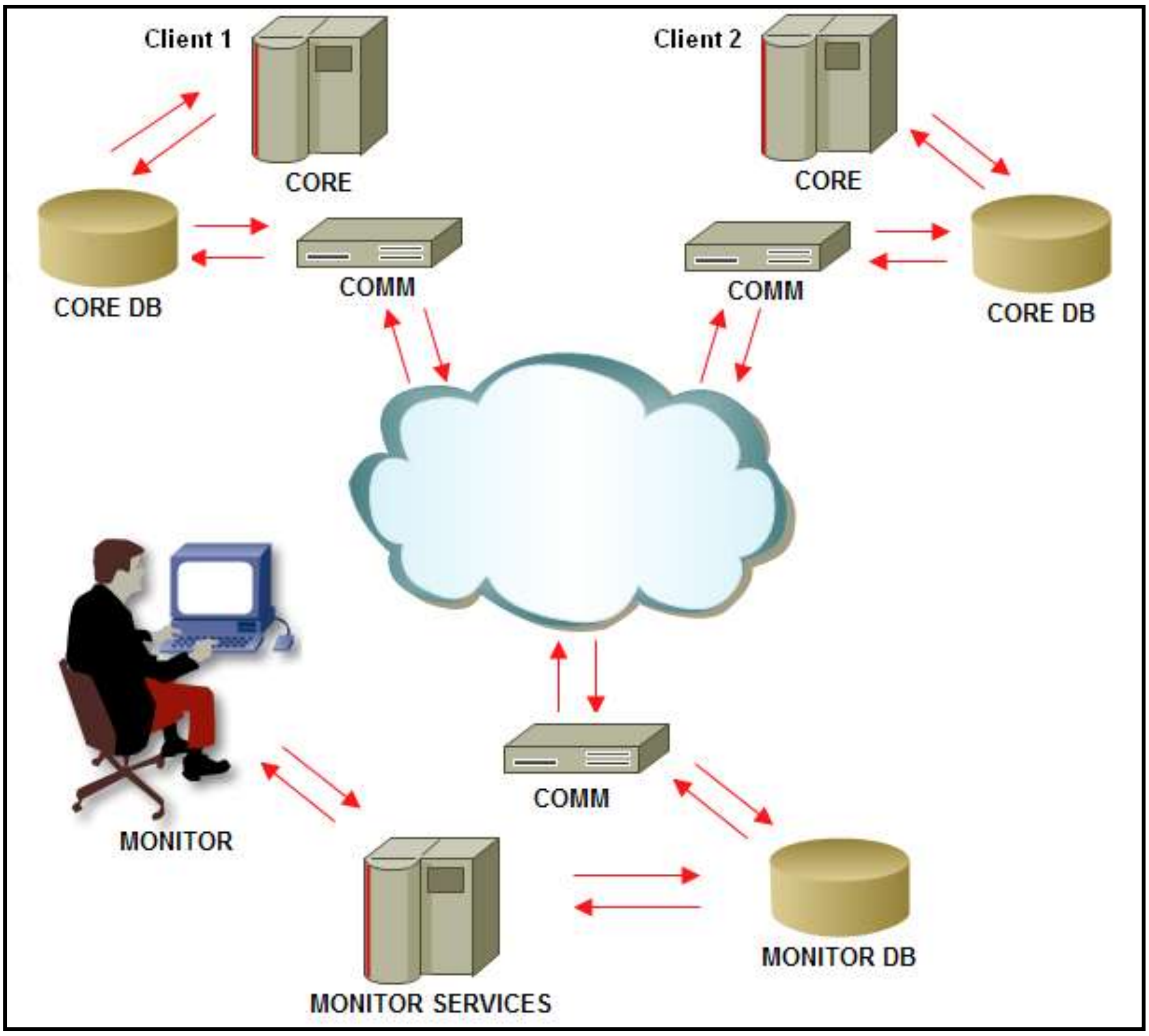
### New requirements

Other than the limitations mentioned in the prevailing system, new enhancements and or requirements are added into the new system to make it more adaptable to current market needs and ease with which the system can be used or leveraged. Some of the requirements are met in the prevailing system, though an elaboration and also enhancement to those are achieved.

1. The CORE processing and the monitoring system should never be altered or modified for any new project implementation. Project specific needs to be add-ons in the new system as either pre-processing event or post-processing event.
2. The system should be compliant and leveraging existing functional systems like the PRE (Process Request Engine) and the EP (Event Parser)
3. Project specific (both pre-processing and post-processing) needs should be configurable. These objects could either be database objects, event parser objects or Java classes. There should be provisions for execution control as –
    a. Grouped – for parallel processing and hence utilizing the CPU to the fullest
    b. Dependent – to set a dependency tree either among groups or for individual objects or both
    c. Parallel – to utilize the CPU to the fullest, whereby, independent objects or groups can be executed simultaneously.

4. The system should provide a user interface for monitoring and controlling the batch proceedings. There are two UI modules to be provided as per client requirements.
    a. One UI for the batch administrator or for client personnel to view and control the batch proceedings. This UI should be clubbed in with the rest of the application screens with appropriate security (conforming to the application security, in general). The layout and overall architecture of the UI module should conform to the rest of the applications architecture.
    b. One UI to view and control various batch proceedings across implementations for the STG Suite administration or the operations team, whereby, the administrator would be able to view and control the batch proceedings of several batch implementations across clients and across geographies.
5. The monitoring UI (both individual as well as across implementation) should provide the following details and functionalities.
    a. VIEW
        i. The UI should allow the user to view the current proceedings of a batch in an ‘online’ mode, where the execution status of each logical unit should be displayed on the screen as it happens or completed.
        ii. Failures of execution of individual objects and reasons, if any, from previous and current cycle of batches.
        iii. Historical data to be collated and presented
        iv. Overall ‘health’ of the current batch cycle
    b. FUNCTIONAL
        i. The UI should allow the user to START the batch
        ii. The UI should allow the user to STOP and RESUME (RESTART) the batch
        iii. The batch should be able to be configured for running in ‘time slots’. This configuration setting should be allowed through the UI. Ex: The user can use the UI to set the time slots as 22:00 (or 10:00 PM) to 00:00 (or 12:00 AM) and 2:00 (2:00 AM) till it completes for the current batch cycle. This would direct the batch to run from 22:00 to 00:00 to a logical save point (that could be a little over 00:00) and then restart itself at 2:00. Threshold for the time should also be configurable.
6. The system should be able to do a system compliance check prior to starting the cycle for memory, dependent engine (like the PRE, report engine, web service engine etc) statuses, secondary storage capabilities. It should be able to report any deviations and should exit if any critical situation is encountered.
7. The system should be able to a statistical check with historical data and inform the administrator about the ‘estimated time to complete’ and use this data during the START / PAUSE / RESUME functionalities internally as well.
8. The system should be sensitive to ‘on-failure-do-what’ mechanism, whereby pre-processing or post-processing object could declare the on failure behavior and these should be configurable. Ex: A pre-processing object could declare that the batch should run even if execution errors are encountered during its execution. At the same time, it could also signify to ABORT the current batch cycle.
9. Using the same CORE installation, one should be able to run various un-scheduled batches at the same time. Ex: The user should be able to un-scheduled execute policy batches for different policies at the same time.
10. The MONITOR should, before the start of the actual batch execution provide the execution order of its objects. So any final corrections for the batch needed could be made. This would avoid any runtime dependency issues. This is optional and the system need not always require user’s confirmation.

## 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.

2. 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.

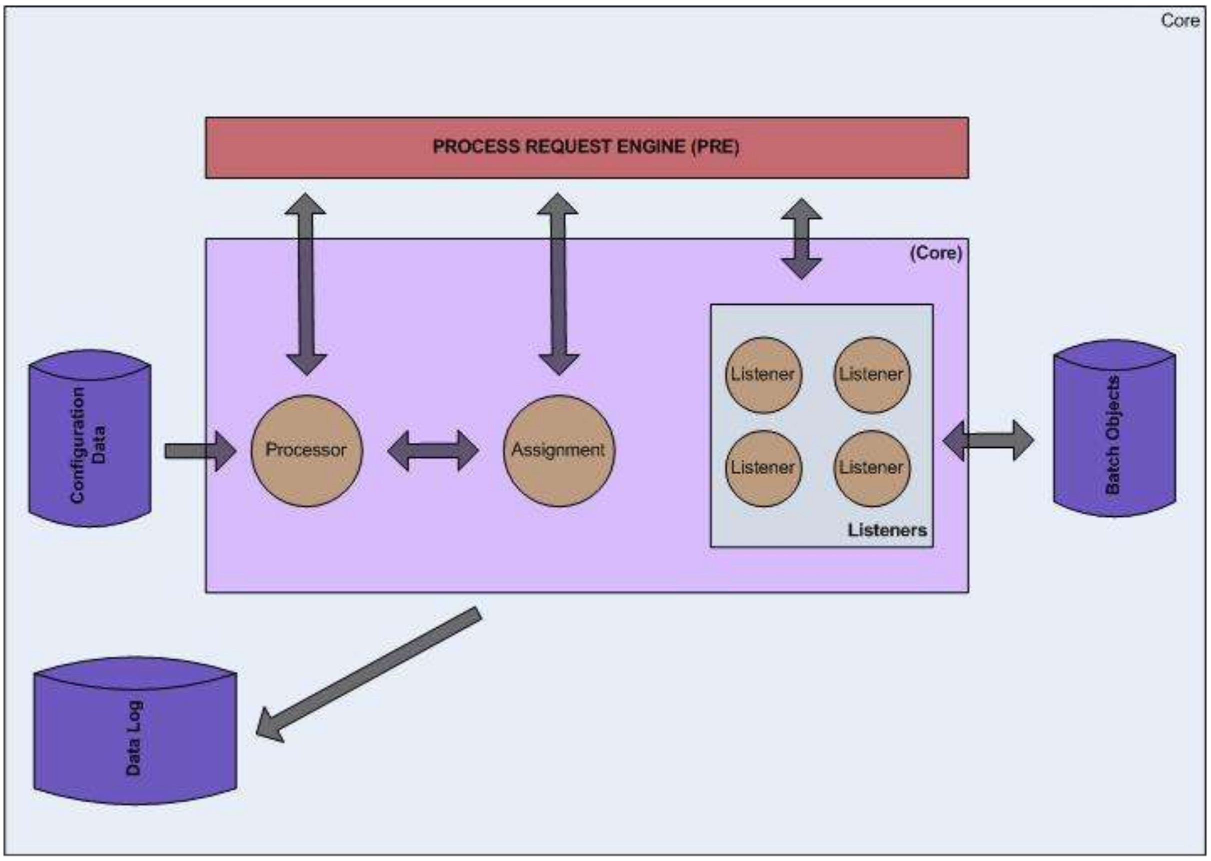
3. Listener Component

   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.

4. 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**

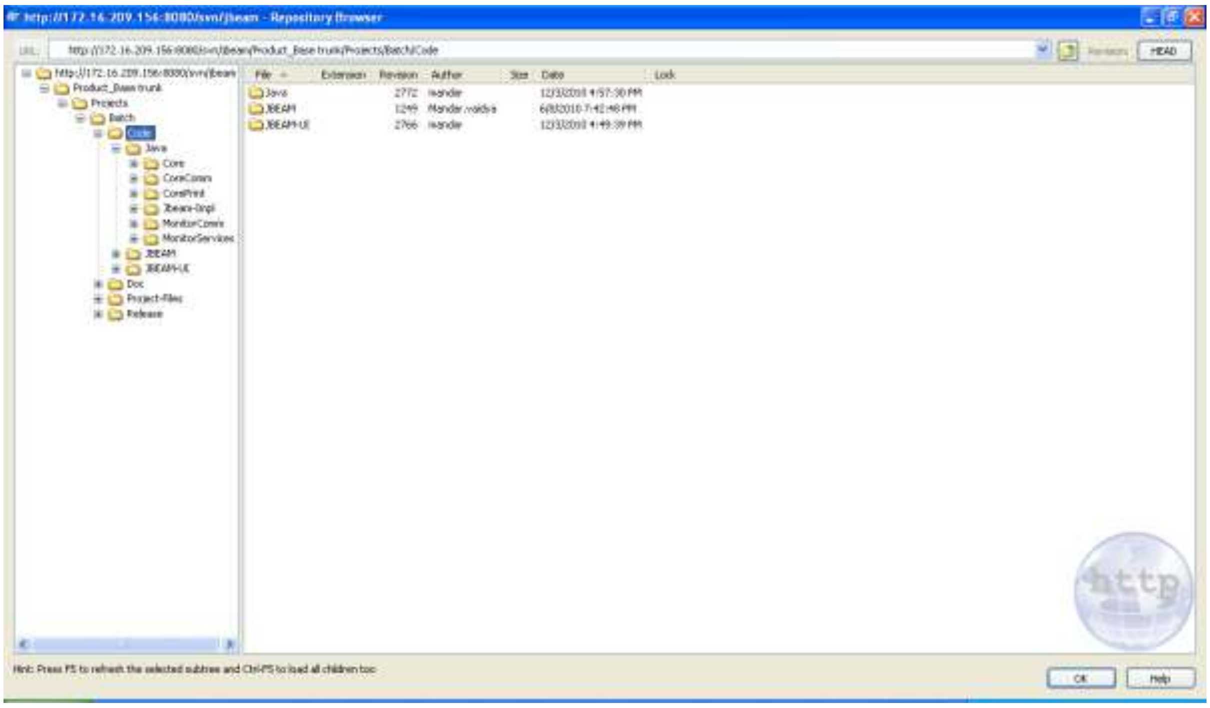
The COMMUNICATION system, as the name suggests, is used to deliver and receive request messages between the CORE and the MONITOR. The system consists of a server and client components at each end. The server components contains of all the web services. The client component would have the client call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

## Source Repository

SVN location: *http://172.16.209.156:8080/svn/jbeam/Product_Base trunk/Projects/Batch/Code/Java*


http://172.16.209.156:8080/svn/jbeam - Repository Browser
URL: http://172.16.209.156:8080/svn/jbeam/Product_Base trunk/Projects/Batch/Code
HEAD
http://172.16.209.156:8080/svn/jbeam
Product_Base trunk
Projects
Batch
Code
Java
Core
CoreComm
CorePrint
Jbeam-Impl
MonitorComm
MonitorServices
JBEAM
JBEAM-UI
Doc
Project-Files
Release
OK
Help

| File | Extension | Revision | Author | Size | Date | Lock |
|---|---|---|---|---|---|---|
| Java | | 2772 | mandar | | 12/13/2010 4:57:30 PM | |
| JBEAM | | 1249 | Mandar.vaidya | | 6/8/2010 7:42:48 PM | |
| JBEAM-UI | | 2766 | mandar | | 12/13/2010 4:49:39 PM | |

Hint: Press F5 to refresh the selected subtree and Ctrl-F5 to load all children too

## Software Context & Deliverables

The system would be used for batch processing needs for both new and existing implementations. The system is delivered as a zipped file that contains –

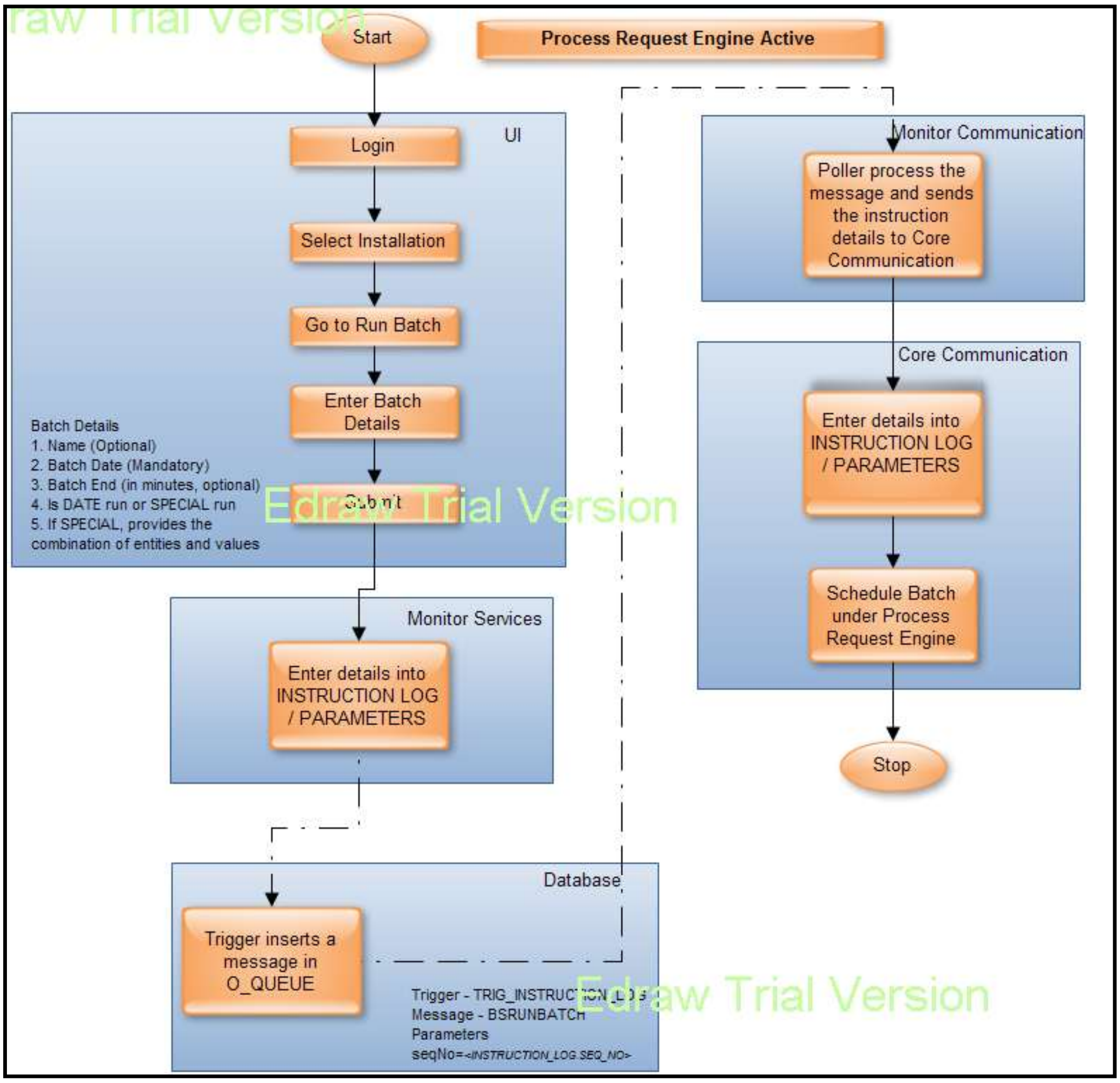
1. The *core.jar* that contains the needed classes for batch functioning
2. The Process Request Engine deployable Each installation or client would have a PRE instance of its own to avoid cross cutting with other project scheduling needs and also to enhance performance.
3. The database object creation / alteration script used by the system
4. The COMMUNICATION system is used to communicate messages across network through various components or sub systems. This sub system comprises of two components, one at the CORE end *core-comm.jar* and the other at the MONITOR *monitor-comm.jar*. Each of these components is further broker down into a server component to receive the requests and a client component to send the request message.
5. The monitoring UI *JBEAM.air* that contains the User Interface
6. The monitoring services monitor-services.jar that communicates with the UI
7. The database script of the monitoring system for database objects creations.
8. An installation guide to assist the setting up of the BPMS system.

The impact to move from the legacy batch system to the new ones for existing implementations or projects is as detailed below –

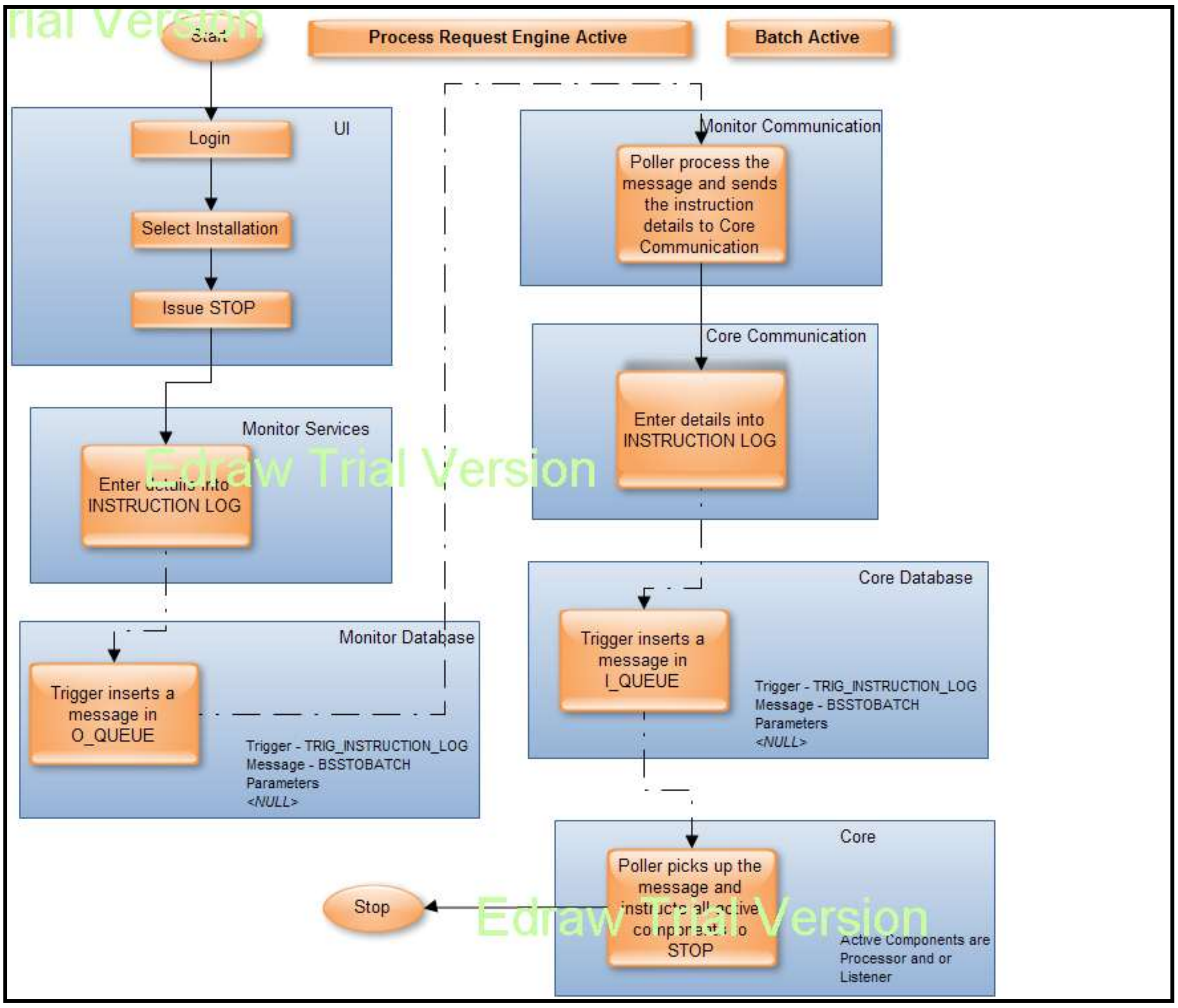
| # | Change Description | Severity |
|---|---|---|
| 2 | New schema would be introduced for –<br>• CORE<br>• MONITOR | LOW |
| 3 | Java 6 would have to be installed. It is not mandatory for the application or project implementation to be moved to JAVA 6. JAVA 6 would be needed for the system only. | MEDIUM |
| 6 | One time setup for the setup table would be needed. Usually it is an one time activity –<br>1. Setting up of pre-processing and post-processing activities or events in the new columns<br>2. Setting up of the Object map table that maps the object name with the actual executing object | MEDIUM |
| 7 | Regression and or integration test. | MEDIUM |

## Communication Flow

### Start / Restart Batch *[MONITOR to CORE]*

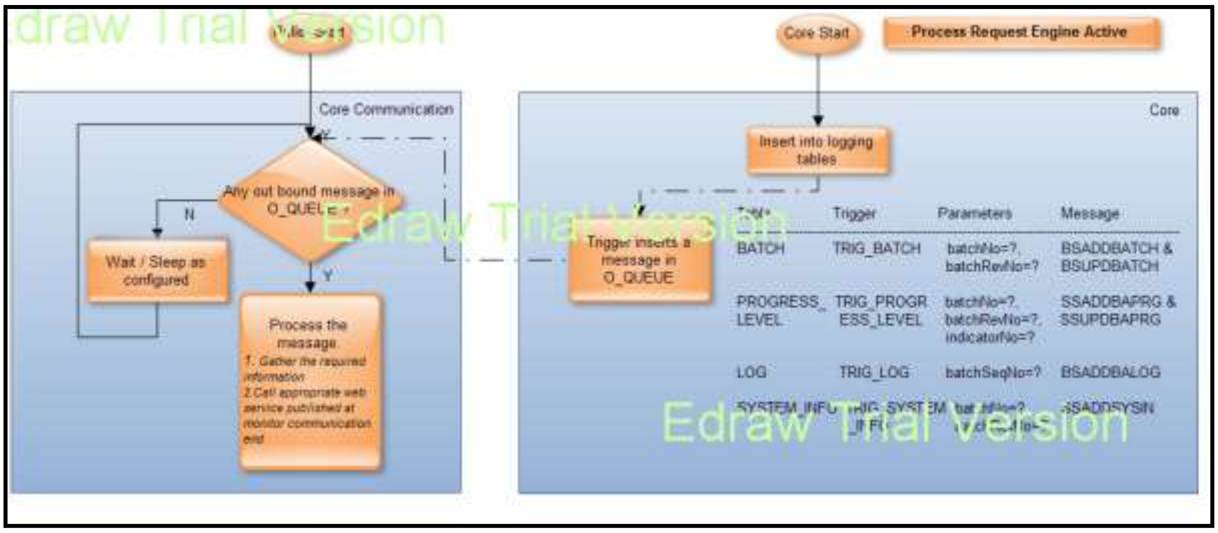

Start
Process Request Engine Active
UI
Login
Select Installation
Go to Run Batch
Enter Batch Details
Batch Details
1. Name (Optional)
2. Batch Date (Mandatory)
3. Batch End (in minutes, optional)
4. Is DATE run or SPECIAL run
5. If SPECIAL, provides the combination of entities and values
Submit
Monitor Services
Enter details into INSTRUCTION LOG / PARAMETERS
Database
Trigger inserts a message in O_QUEUE
Trigger - TRIG_INSTRUCTION_LOG
Message - BSRUNBATCH
Parameters
seqNo=<INSTRUCTION_LOG.SEQ_NO>
Monitor Communication
Poller process the message and sends the instruction details to Core Communication
Core Communication
Enter details into INSTRUCTION LOG / PARAMETERS
Schedule Batch under Process Request Engine
Stop

## Stop Batch *[MONITOR to CORE]*


Start
Process Request Engine Active
Batch Active
UI
Login
Select Installation
Issue STOP
Monitor Services
Enter details into INSTRUCTION LOG
Monitor Database
Trigger inserts a message in O_QUEUE
Trigger - TRIG_INSTRUCTION_LOG
Message - BSSTOBATCH
Parameters
<NULL>
Monitor Communication
Poller process the message and sends the instruction details to Core Communication
Core Communication
Enter details into INSTRUCTION LOG
Core Database
Trigger inserts a message in I_QUEUE
Trigger - TRIG_INSTRUCTION_LOG
Message - BSSTOBATCH
Parameters
<NULL>
Core
Poller picks up the message and instructs all active components to STOP
Active Components are Processor and or Listener
Stop

**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.



Core Start
Process Request Engine Active
Core Communication
Core
Insert into logging tables
Trigger inserts a message in O_QUEUE
Any out bound message in O_QUEUE ?
N
Y
Wait / Sleep as configured
Process the message
1. Gather the required information
2. Call appropriate web service published at monitor communication end
Edraw Trial Version

| Table | Trigger | Parameters | Message |
|---|---|---|---|
| BATCH | TRIG_BATCH | batchNo=?, batchRevNo=? | BSADDBATCH & BSUPDBATCH |
| PROGRESS_LEVEL | TRIG_PROGRESS_LEVEL | batchNo=?, batchRevNo=?, indicatorNo=? | SSADDBAPRG & SSUPDBAPRG |
| LOG | TRIG_LOG | batchSeqNo=? | BSADDBALOG |
| SYSTEM_INFO | TRIG_SYSTEM_INFO | [illegible] | [illegible] |

## Important Classes & Significance

### Core

| Class Name | Package | Description |
|---|---|---|
| Processor.java | logic | The main processor class or the entry point into the CORE system.<br>The 'heart and the soul' of the CORE system.<br>This class is invoked and instantiated through Process Request Engine.<br>This class is responsible for using all other classes in the system (except Listeners) to achieve –<br>1. Procreation – procreating meta events (PRE / POST) as configured<br>2. Assignment – Assigning of the batch jobs with listeners<br>3. Scheduling – Scheduling of listeners in Process Request Engine<br>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.<br>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.<br><br>Processor is responsible of using this assignment handler class to achieve assignment.<br><br>There are two types of batch objects and has a special handler class to cater to those.<br><br>Meta Events (PRE / POST) - logic/MetaEventsHandler.java<br><br>Batch Objects - logic/ BatchEventsHandler.java |
| ExecutionHandler.java | logic | Primarily responsible for execution of a batch object.<br><br>Invoked and used by the Listener.<br><br>There are three special execution handlers each catering to a job type.<br><br>JV - logic/ JAVAExecutionHandler.java<br><br>EV - logic/EventParserObjectExecutionHandler.java<br><br>PL - logic/PLSQLExecutionHandler.java<br><br>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.<br><br>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.<br><br>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.<br><br>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.<br><br>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**

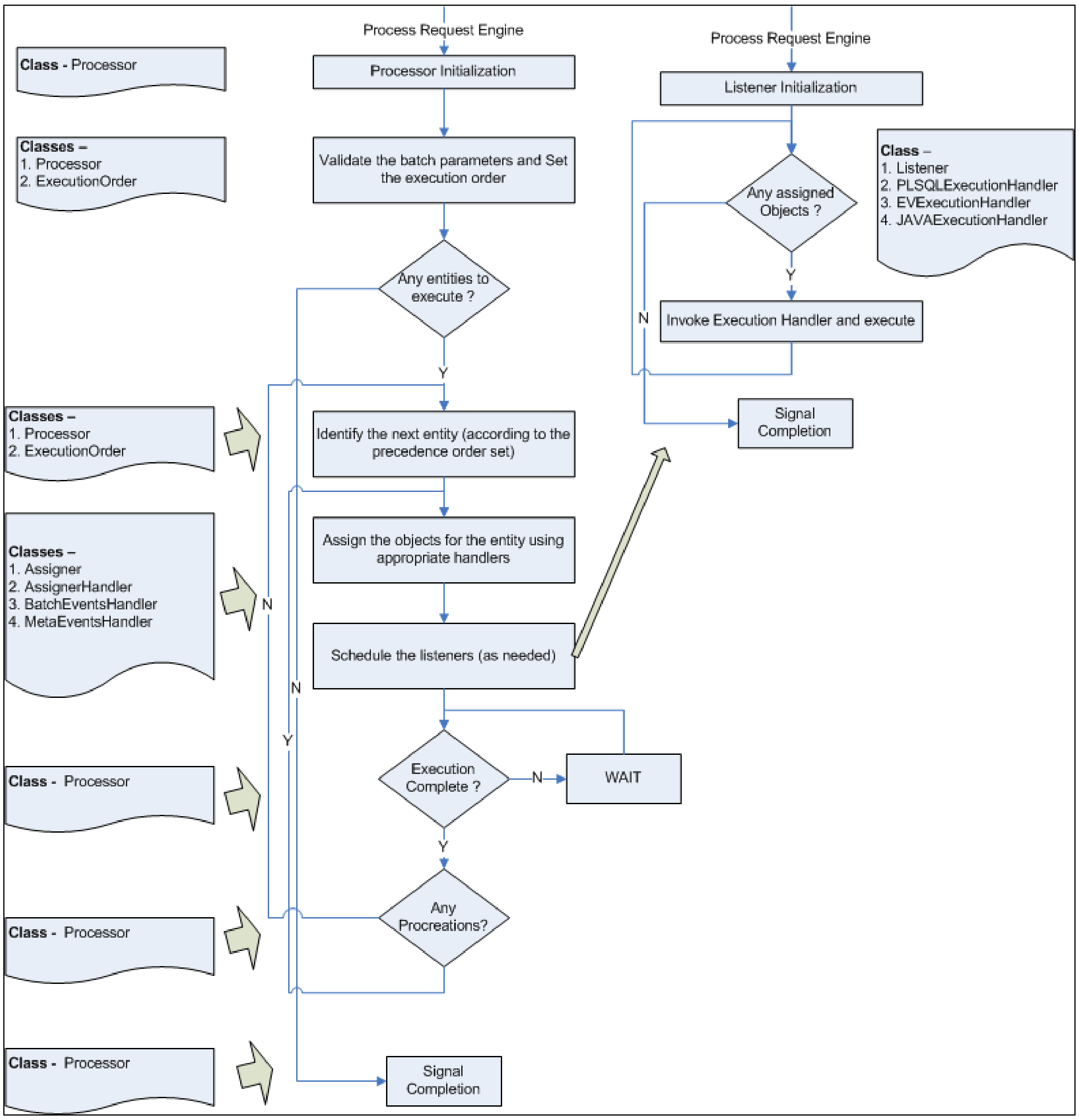
| Class Name | Package | Description |
|---|---|---|
| OutBoundQueuePoller | util | As the name suggests, it polls for ant out bound messages to the MONITOR system. |
| TransmitBatchDetails.java | messagehandlers | Special message handler class to transmit the batch details or information.<br><br>The messages processed by this special class are –<br><br>BSADDBATCH – When a batch information is to be added<br><br>BSUPDBATCH – When the batch information is to be updated |
| TransmitBatchLog.java | messagehandlers | Special message handler class to transmit the batch log and related information.<br><br>The message processed by this class is –<br><br>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.<br><br>The messages processed by this class<br><br>SSADDBAPRG – when a progress level is to be added<br><br>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.<br><br>The message processed by this class –<br><br>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.<br><br>The message processed by this class is –<br><br>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.<br><br>BSSTOBATCH – Instruction to stop the batch<br><br>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.<br><br>BSCALENDAR – Instruction to add calendar |

## Appendix

### Batch Execution Flow


Process Request Engine
Processor Initialization
Validate the batch parameters and Set the execution order
Any entities to execute ?
Y
N
Identify the next entity (according to the precedence order set)
Assign the objects for the entity using appropriate handlers
Schedule the listeners (as needed)
Execution Complete ?
N
WAIT
Y
Any Procreations?
Signal Completion
Class - Processor
Classes – 1. Processor 2. ExecutionOrder
Classes – 1. Processor 2. ExecutionOrder
Classes – 1. Assigner 2. AssignerHandler 3. BatchEventsHandler 4. MetaEventsHandler
Class - Processor
Class - Processor
Class - Processor
Process Request Engine
Listener Initialization
Any assigned Objects ?
Y
N
Invoke Execution Handler and execute
Signal Completion
Class – 1. Listener 2. PLSQLExecutionHandler 3. EVExecutionHandler 4. JAVAExecutionHandler

## 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.<br>SPECIAL – When the batch is run for a user chosen combination of entities<br>DATE – When the batch is either scheduled or the user chooses to run it for a date.<br>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 –<br>BATCH_COMPLETED – The batch has completed all its activities.<br>USER_INTERRUPTED – When the user chooses to stop a running batch<br>END_OF_TIME – When the time allotted for the batch is exhausted.<br>BATCH_FAILED – When an object marked as on-fail-exit = 'Y' fails there by halting the batch execution.<br>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 –<br>'L'ocked for execution<br>'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.<br>EX: PRE, POLICY, POST etc.<br>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.<br>‘N’ or null otherwise<br>EX. There are 10 records in for POLICY P1.<br>1 through to 3 have executed successfully. 4th has failed.<br>If ON_ERROR_FAIL_ALL = ‘Y’, then the status would be –<br>1 to 3 = CO<br>4 = ‘99’<br>5 to 10 = ‘SP’<br>If ON_ERROR_FAIL_ALL = ‘N’, then the status would be –<br>1 to 3 = CO<br>4 = ‘99’<br>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<br>S – String<br>I - Integer<br>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 | ~~Indication when this object fails, exit the batch or continue with the batch.~~<br>~~EX: If an object marked as ‘Y’ fails execution, then the batch would be marked as a FAILED_BATCH and would be stopped with immediate effect. If an object marked as ‘N’ fails, then the batch would still continue the proceedings marking it as ‘99’~~<br>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.<br>The batch can run any PRE / POST event in parallel or as dependent objects. The priority code 1 plays an important here.<br>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.<br>EV – Event Parser<br>JV – Java<br>PL – PLSQL<br>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)<br>EX: There exist a stored procedure with signature<br>SOME_PKG.SOME_SP(<br>Id IN number,<br>Name IN varchar2,<br>DOB IN date,<br>Deparment IN varchar2 default 'ADMIN'<br>)<br>then this field would have<br>SOME_PKG.SOME_SP(:N,:VC,:DT, :VC) |
| OBJECT_TYPE | The type for the object.<br>EV – Event Parser<br>JV – Java<br>PL – PLSQL<br>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.<br>'Y' – stop the batch<br>'N' or null – continue with batch proceedings |
| ON_FAIL_EMAIL | Indication whether to send email if the batch execution fails.<br>'Y' – send email<br>'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<br>'Y' – send email<br>'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.<br>EX: PRE, POLICY, POST etc. |
| PRG_ACTIVITY_TYPE | The progress activity type. It could be one of<br>EX:<br>INITIALIZATON – Initialization of the batch<br>EXECUTION ORDER – Setting up of the execution order for the batch<br>PROCREATION – Procreating PRE / POST events<br><br>ASSIGNMENT – Assigning of batch objects for an entity [Iterates in cycles]<br>SCHEDULING – Scheduling of the assigned batch objects for an entity [Iterates in cycles]<br>EXECUTION – Execution of the scheduled batch objects for an entity [Iterates in cycles]<br>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:<br>i) A new user created (email with user name and password<br>ii) Administrator resets the password.<br>iii) 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.<br>Indication whether to change the password on screen<br>'Y' – User needs to change password<br>'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<br>'Y' – User with administrative rights<br>'N' - User without administrative rights<br>The user with administrative rights<br>An user with ADMIN role has following capabilities:<br>i) Create new user with roles and installations<br>ii) Edit user with roles and installations<br>iii) Reset password<br>iv) To give or remove access to JBEAM UI |
| CONNECT_ROLE | Indication whether the user can connect to JBEAM UI<br>'Y' – User can access JBEAM application on particular server<br>'N' - User can not access JBEAM application on particular server<br>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. |

## How To

### Build / Deploy / Configure / Start components

#### Building 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 |
| | |

## 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-systems –

CORE [CODE1='CORE']

| CODE2 | CODE3 | 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@mastek.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 | **ERIE 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 |
| ICD_SERVICE | USER_ID | csr1 | User id for ICDService |
| ICD_SERVICE | PASSWORD | Icd123 | Password for ICDService |
| ICD_END_POINT_URL | URL | http://172.16.93.17:8789/ICDService/services/ProcessManager/handleRequest | Default execution handler implementation for launch flow |

**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 |

## Converting existing Pro*C

### Step 1: Identify Package Variables being set before calling Batch Object

The pro*C as was provided by Oracle was able to access package variables directly from within and therefore there was no need to have procedures to set these before calling the actual batch objects. But in case of JBEAM as it uses the JDBC API there is no direct access to the package variables and therefore it is important to first identify the package variables being updated from pro*C. The monitor pro*C updates certain package variables and the listener pro*C updates another set of variables. Therefore there are two procedures already created namely `set_core_val_app` and `set_core_val_app_lis` (refer to the Billing team). If the functionality of these two procedures is not adequate then you may have to modify the packages to suite your needs.

### Step 2: Identify the listener If conditions and populate Object Map

The pro*C code for listener has IF <condition> THEN <call procedure/function> hard coded within it. It is therefore necessary to go through each IF condition and then identify what procedure is being called. This procedure name goes into `Object_Map` table. The default handlers provided with the JBEAM executes the procedure as per the `be_taskname` in the `batch_executor` table. Say `be_taskname` has `abc('114','999')` and say we have mapped the task to `abc` procedure then we need the handler is going to execute the procedure with just 2 parameters. Now important point is that if the `abc` procedure is expecting 3 parameters and none have default value (in other words, all parameters are mandatory) then the default handler will fail to execute the `abc`. Therefore a wrapper Java class must be written to handle such situations.

### Step3: Identify Pre and Post events

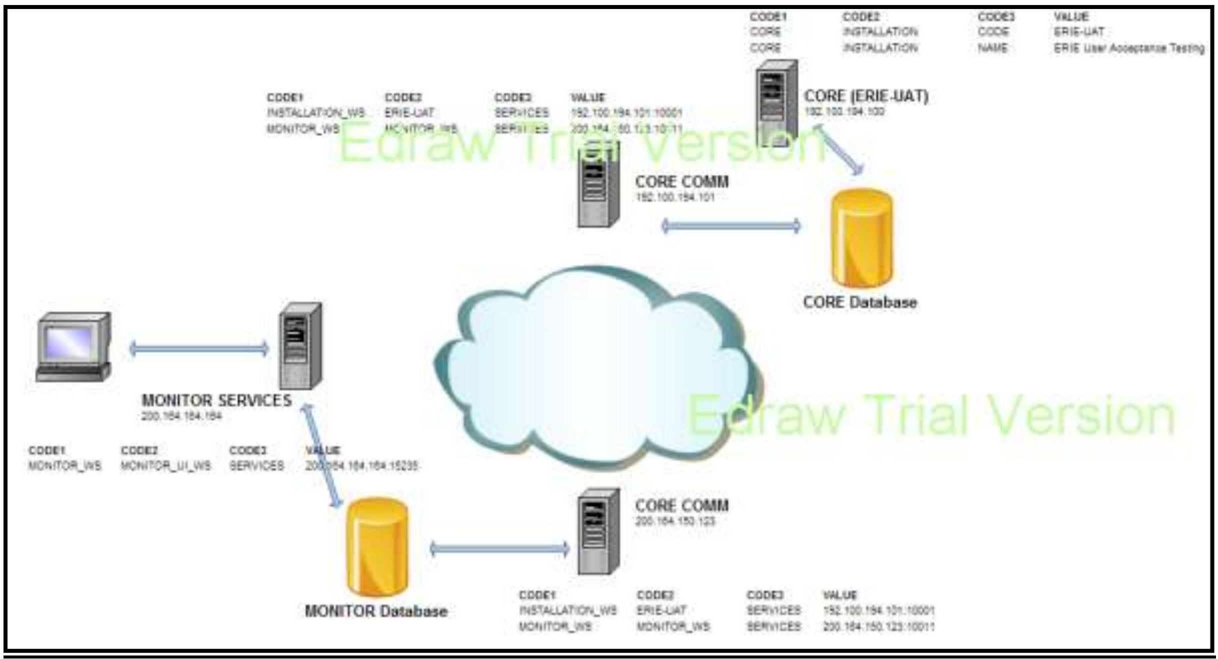
Identify pre and post events that are hardwired in the monitor pro*C file and correspondingly create its own java wrappers. As explained in the earlier sections of the document these pre/post events must be configured in the `Object_Map` table.

## Converting existing JBatch

Same steps as above though the step1 probably may not be required as the procedures should already exist in this case.

## Deploying the Components

Deploying the components may involve multiple servers. The configurations would have to change according to the servers and the components deployed on the server. Following is a snap shot of an example. The example takes a worst case scenario where all components are deployed on different machines or servers and the way configurations should be done.



| CODE1 | CODE2 | CODE3 | VALUE |
|---|---|---|---|
| CORE | INSTALLATION | CODE | ERIE-UAT |
| CORE | INSTALLATION | NAME | ERIE User Acceptance Testing |

| CODE1 | CODE2 | CODE3 | VALUE |
|---|---|---|---|
| INSTALLATION_WS | ERIE-UAT | SERVICES | 192.100.194.101:10001 |
| MONITOR_WS | MONITOR_WS | SERVICES | 200.164.150.123:10011 |

| CODE1 | CODE2 | CODE3 | VALUE |
|---|---|---|---|
| MONITOR_WS | MONITOR_UI_WS | SERVICES | 200.164.164.164:15235 |

| CODE1 | CODE2 | CODE3 | VALUE |
|---|---|---|---|
| INSTALLATION_WS | ERIE-UAT | SERVICES | 192.100.194.101:10001 |
| MONITOR_WS | MONITOR_WS | SERVICES | 200.164.150.123:10011 |

**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. |

**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');
```

4 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.ProcessOutboundInte
rfaces','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.

**Schedule batch**

The batch can be scheduled now. The details are as below

1. The UI module provides screens for marking holidays (both business and weekends).
2. The UI captures the required schedule for a batch associating a calendar
3. With each run of the batch a schedule will be associated.

This information i.e. the Holidays + Schedule + Batch Run details (with parameters) is sent over to the CORE system where it resides in the Process Request Engine related tables.

1. End of time – When a scheduled batch has an end-of-time attribute, the batch during closing of the batch revision (when end-of-time is realized) procreates new Process Request Engine jobs appropriately. This is done recursively till the batch is completed.
2. Batch Parameters – The scheduled batch can be DATE run or SPECIAL run and will have more flexibility.

**Fail-Over / Clustered batch**

Even though the fail-over and cluster is managed by Process Request Engine, important points met by the batch are–

1. Whenever an attribute fetched from the Process Request Engine context is altered, it is imperative that the altered value is set back into the context with the same attribute name. The Process Request Engine, while setting an attribute serializes it for the fail-over or for running in cluster mode.
2. The tables used for logging in the CORE / MONITOR databases would have to be updated with a new field / column to hold the information of the Process Request Engine's server identifier or information. [In cluster mode, there would multiple PR engines running, either as active-active or active-passive for fail-over]

**Object Execution**

The identification of an object (in the LOG table) executed in which cycle of the batch is achieved by the following steps –

1. Introduced the column CYCLE_NO in the LOG table.
2. Populated the column CYCLE_NO as like the other columns and transmit the same back to the MONITOR.

**Expiry time for batch objects**

Each object should have an expiry time, after which should be termed as '99' or some other new status. The steps to implement this requirement –

1. An object is defined in the OBJECT_MAP table and hence a new column is to be introduced in this table which defines the expiry time of the object (i.e. type)
2. In the listener code, a new thread would have to introduced that would check for the start time (before the start of the execution) and the end time. The expiry time is to be made available to the listener before hand.
3. If the object under consideration over shoots the time allotted for it, then, would indicate to the listener about marking it as '99' or any appropriate status.

### Web Service Security

The latest version of CORE-COMM and MONITOR-COMM has SSL implementation.

## Requirements & Hints

### Total # of Objects for the batch

The requirement of identifying the total number of objects for an assignment is to be implemented.

The following steps can be followed to achieve –

1. A new column has to be added at a progress level (PROGRESS_LEVEL).
2. Whenever the assignment is done, the DAO method to return the number of updated rows. The summation of these update values should be update against an assignment cycle.
3. The total of the number of objects against each assignment cycle (could happen in procreating objects) could be summarized as the total objects for the batch.

### List of Installation

The UI is coded to display pods instead of a list. The UI should be able to display the list as well, upon request from the user. The user should be able to toggle between the two modes.

Hints –

1. The UI should have a check box that would be used to toggle between the two display modes.

### Blocking Users for Monitor access

The system should be able to lock a user from accessing the monitor from viewing the batch results, especially when the batch has failed or due to maintenance activities

### Re-processing failed message

The UI should be able to re-process failed message. The message if not processed falls into the DEAD_MESSAGE_QUEUE. The UI should pull up this information and allow the user to re-process these messages.

### Business Exception Handling

An object can be marked as failed for either the execution has failed (i.e. SQLException) or for business reasons. If the object fails for business reason, the reason should be derived or fetched from the ERROR_LOG table of the APPLICATION schema and stored into the logging tables.

## Future probable requirements & Hints

### New Collators

Collators are essentially those component that does additional operation on the same data, usually in interval or periodically.

EX: The graph screen on the UI would show the batch dynamics at run time, polling over the logging table. But once the batch is completed and user comes back to see the graph after closing the screen, the graph would be plotted in straight not capturing the exact dynamics. Hence a collator fits in here, where it is the collator's responsibility to poll the logging tables and dump the collated information into a special table that the UI pings. There by when the user comes back to observe the statistics for a completed batch, the graph would be displayed as it was during the actual execution of the batch. An existing implementation of the batch is that of the graph data collator that dumps the results into a special table GRAPH_DATA_LOG. Usually the collators reside in the MONITOR-COMM that collates and sets the results into the MONITOR database to which FLEX UI module pings. Also, collators cannot be configured from the outside as it becomes a part of the internals working.

The following are the steps to follow for creating a new collator –

1. Create class that extends 'com.stgmastek.monitor.comm.util.Collator'.
2. Implemented the needed method using the dedicated connection supplied by the framework
3. The wait period for the collators is supplied through the CONFIGURATION table and can be changed as per the needs.
4. Register / Make an entry of the collator in the singleton class 'com.stgmastek.monitor.comm.util.Collators' as done for others.

**Data Consumer**

Data consumers are essentially different approaches of handling the incoming data. The system now heavily relies on the database for its persistence needs. This, going ahead might be changed such as the data could be written to a file system etc. So essentially with each incoming web service a data consumer would have to be associated as to what needs to be done with the data.

The hints can be used –

1. The message in the communication system would have to include a handler or type of handlers to be used with the data that is transmitted. There by within the web service call, the handler specification could be included
2. The server (or the receiver) would in turn should have the handlers registered in their respective sub-systems, invoke the appropriate handler and leave it to the implementation of the handler for the consumption of data. The data could be persisted in to the database / file system / channelled to the UI directly. The handlers for a type / group of message / individual message can be configured from the outside and not to be hard wired in the system

**Statistics upon request**

Statistics upon request are those requests to batch information that were never transmitted during the execution of the batch.

EX:

1. Suppose a new batch has started and the communication channel is down. In that case the batch statistics and log data would not reach the MONITOR system.
2. The MONITOR database logs may be purged over a period of time and information of 'old' batch is requested.

The following steps would have to be followed and instead of transmission or push approach and rather simpler request or pull approach would have to be adopted.

1. The UI would have to provide a mechanism to identify when to request the information (as it is not available in the MONITOR database).
2. Required services (and or service) would have to be created at the CORE-COMM system that takes the basic batch information, like the batch # and the batch revision #, and in turn supplies the information to the MONITOR system for it to display.

Important Note

1. The collators residing in the MONITOR-COMM would also have to be implemented in the CORE, else information would be lost. The special collator tables reside only in the MONITOR database.
2. Few columns that are additional added into the MONITOR log tables would also have to be updated in the CORE log tables.

## Template Amendment History

| Version | Date (dd/mm/yyyy) | Amendment History | Remarks |
|---|---|---|---|
| 0.10 | 08/06/2009 | Original Version | |

QMS Document Control Information

| | | |
|---|---|---|
| Division : Software Development | Doc No: | Version: |