**For Incoming request:**

AsyncOperationsBsBean class has the method **scheduleAsyncOperation** which is being called through batch, we have done changes under this method for retrieving tracking id from work area (which was already existing in the code ) and to set it in the **AsyncOperationDTO** inorder persist tracking Id value to the table **VMGRASOP\_ASYNC\_OPERATION\_V66**.

@Stateless

**public** **class** AsyncOperationsBsBean **implements** AsyncOperationsBs {

…….

…….

……

/\*\* LoggingManager \*/

**private** LoggingManager loggingManager;

**private** **void** initLoggingManager() {

**if** (loggingManager == **null**) {

loggingManager = **new** LoggingManager();

}

loggingManager.initWorkArea();

}

@Override

**public** **void** scheduleAsyncOperation(AsyncOperationDTO payload, ScheduleAsyncOperation scheduleAsyncOperation) {

initLoggingManager();

String trackingId = loggingManager.retrieveFromWorkArea(LoggingContextEnum.***CONTEXT\_KEY\_TRACKING\_ID***);

***LOGGER***.fine(

"Entering scheduleAsyncOperation with parameter async operation={2}, trackingId={1} and payload={0}",

scheduleAsyncOperation, trackingId, payload);

ArgumentChecker.*checkNotNull*(payload, "payload");

payload.setTrackingId(trackingId);

// If async operation is batch or the queue mechanism is deactivated

**if** (ScheduleAsyncOperation.***BATCH***.equals(scheduleAsyncOperation) || !configurationReaderBs

.getConfigurationValue(Boolean.**class**, FoundationSrvConfigConstants.***ASYNC\_QUEUE\_ACTIVATED***)) {

AsyncOperationBE asyncOperationBe = AsyncOperationDTO2EntityConverter

.*convertAsyncOperationDTO2Entity*(payload);

asyncOperationsBf.scheduleAsyncOperation(asyncOperationBe);

} **else** {

asyncMessageSenderBf.sendMessageAsync(payload);

}

***LOGGER***.fine("Exiting scheduleAsyncOperation");

}

}

1. **After Executing the batch:**

**AsyncOperationExecutorBsBean** class has the method **execute(AsyncOperationDTO operation)** which is being called through batch, we have done changes under this method for retrieving tracking id from **AsyncOperationDTO** and set it to work area.

**public** **class** AsyncOperationExecutorBsBean **implements** AsyncOperationExecutorBs {

…….

…..

…..

@Override

**public** AsyncOperationResultDTO execute(AsyncOperationDTO operation) {

initLoggingManager();

String trackingId = operation.getTrackingId();

***LOGGER***.fine("In AsyncOperationExecutorBsBean the TrackingId trackingId={0}, The complete AsyncOperationDTO: {1}", trackingId, operation.getPayload());

loggingManager.storeToWorkAreaAndThreadLocal(LoggingContextEnum.***CONTEXT\_KEY\_TRACKING\_ID***, trackingId);

…..

….

}

…

}

And retrieving the **trackingId** and logging it in the below methods which are under the **OperationsExecutorTaskBsBean** class.

public class OperationsExecutorTaskBsBean implements OperationsExecutorTaskBs {

……

…..

…

@Override

**public** **void** finalizeTask(SoeTaskProgressDTO taskProgressDTO) {

String trackingId = loggingManager.retrieveFromWorkArea(LoggingContextEnum.***CONTEXT\_KEY\_TRACKING\_ID***);

***LOGGER***.info(SoeSrvAsyncOperationsExecutionPaiMessageIds.***PAI\_MSG\_03***, "The batch task {0}, trackingId {1} has finished sucessfully.",

taskProgressDTO.getTaskId(), trackingId);

}

@Override

**public** **void** taskFailed(SoeTaskProgressDTO taskProgressDTO) {

String trackingId = loggingManager.retrieveFromWorkArea(LoggingContextEnum.***CONTEXT\_KEY\_TRACKING\_ID***);

***LOGGER***.severe(SoeSrvAsyncOperationsExecutionPaiMessageIds.***PAI\_MSG\_04***, "The batch task {0} has failed with trackingId {1} .",

taskProgressDTO.getTaskId(), trackingId);

}

…..

…..

….

}