**BNPP Simulator**

Simulator acts as a subsystem for handling Earmarking, Charges, SSW and Forex request which receives from BPH, process it and responds back to BPH.

Following are the subsystems:

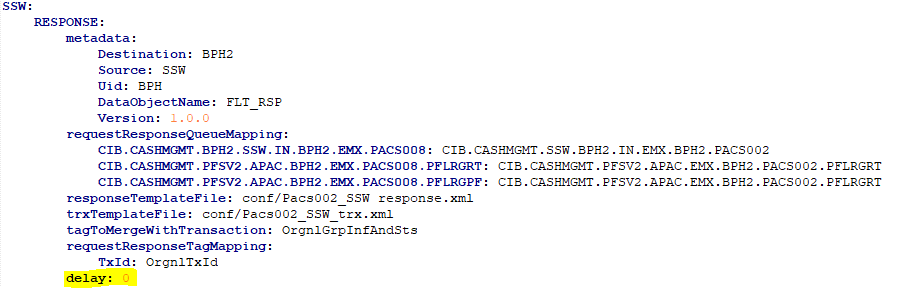
* **Earmarking – ATLAS2 –** supports CFT and JMS (different request and response queue for every country).
* **SSW – SSW Filtering -** supports JMS (Request queue and response queue are based on the modes which supports for APAC region and for India there is different request and response queue.)
* **Charges – TBS2 -** supports JMS (only one request queue and one response queue which supports for all country).
* **Forex –** supports JMS ((only one request queue and one response queue which supports for all country).
* **ICheque-** supports cheque request handling.
* **Jompay-** for biller validation.
* **Account Verification Request - CAS/PCS –** Account Verification Request for TH PP and **S**G PayNow.

**Benefits:**

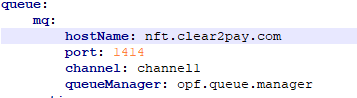
* Simulator can be used in NFT where bulk transactions get processed for different product and different country.
* No need to upload the subsystem response manually.
* Simulator help processing Payment file with bulk transactions in one go in QA environment.
* No effort in creating response files for all subsystems.
* It can work with different automation tools such as JETQA, TFK etc.
* This simulator can easily communicate with different type of queue managers like ActiveMQ, WebSphere MQ manager.
* This simulator can process the request whether is coming through file system or queue.
* Can put the delay in the response as well if required.

Here are the steps require for setting up Simulator:

1. Java 8 must be installed in the node where simulator is being setup.
2. Extract the simulator-mq.zip
3. Make sure that EMXQCF is pointing to MQ JMS provider and not Tibco EMS
4. Make sure that following CFT paths are setup and mount across the nodes and should be accessible by Simulators and can be modified the same in the simulator-mq/conf**/application.yml** if required
   * Earmarking
     + Request Path:
       - /apps/bph2/msg/CAMP/IN/tmp
       - /apps/bph2/msg/CAMP/SEA/tmp
       - /apps/bph2/msg/CAMP/SG/tmp
     + Response path:
       - /apps/bph2/msg/CAMP/SEA/recv
       - /apps/bph2/msg/CAMP/IN/recv
       - /apps/bph2/msg/CAMP/SG/recv
5. Make sure that following queues are created on MQ:
   1. Earmarking
      1. REQUEST QUEUE:
         * CIB.CASHMGMT.BPH2.CAMP.IN.BPH2.EMX.EMKREQ
         * CIB.CASHMGMT.CAMPS.SG.ALL.EMX.PXML.EMKREQ
         * CIB.CASHMGMT.CAMPS.MY.ALL.EMX.PXML.EMKREQ
         * CIB.CASHMGMT.CAMPS.TH.ALL.EMX.PXML.EMKREQ
         * CIB.CASHMGMT.CAMPS.VN.ALL.EMX.PXML.EMKREQ
      2. RESPONSE QUEUE:
         * CIB.CASHMGMT.CAMP.BPH2.IN.EMX.BPH2.EMKRSP
         * CIB.CASHMGMT.CAMPS.SG.EMX.ALL.PXML.EMKRSP
         * CIB.CASHMGMT.CAMPS.MY.EMX.ALL.PXML.EMKRSP
         * CIB.CASHMGMT.CAMPS.TH.EMX.ALL.PXML.EMKRSP
         * CIB.CASHMGMT.CAMPS.VN.EMX.ALL.PXML.EMKRSP
   2. SSW
      1. REQUEST QUEUE:
         * CIB.CASHMGMT.PFSV2.APAC.BPH2.EMX.PACS008.PFLRGRT
         * CIB.CASHMGMT.BPH2.SSW.IN.BPH2.EMX.PACS008
         * CIB.CASHMGMT.PFSV2.APAC.BPH2.EMX.PACS008.PFLRGPF
      2. RESPONSE QUEUE:
         * CIB.CASHMGMT.SSW.BPH2.IN.EMX.BPH2.PACS002
         * CIB.CASHMGMT.PFSV2.APAC.EMX.BPH2.PACS002.PFLRGRT
         * CIB.CASHMGMT.PFSV2.APAC.EMX.BPH2.PACS002.PFLRGRT
   3. Charge
      1. REQUEST QUEUE  :
         * CIB.CASHMGMT.BPHV2.APAC.BPH2.EMX.PACS008.CHARGE
      2. RESPONSE QUEUE:
         * CIB.CASHMGMT.BPHV2.APAC.EMX.BPH2.PACS002.CHARGE
   4. Forex
      1. REQUEST QUEUE  :
         * CIB.CASHMGMT.FXBUNDLE.APAC.ALL.EMX.PACS008.FX
      2. RESPONSE QUEUE:
         * CIB.CASHMGMT.FXBUNDLE.APAC.EMX.BPH2.PACS002.FX
   5. BVM
      1. REQUEST QUEUE  :
         * CIB.CASHMGMT.JOMP.MY.BPH2.EMX.PXML.BVMREQ
      2. RESPONSE QUEUE:
         * CIB.CASHMGMT.JOMP.MY.EMX.BPH2.PXML.BVMRES
6. Delay can be introduced in the response of the interfaces(SSW, Charge, Forex and Earmarking) using the delay attribute which can be seen for SSW. The value of delay is in milliseconds.



1. Update the file application.yml located at: simulator-mq/conf**/**
   1. simulator-mq/conf**/application.yml** : Update the queue manager details.



1. Update the emulator.sh located at simulator-mq/:

* Update the Java bin path at Line No 16 and required memory for simulator is mentioned as well



1. How to run the simulator:

* Go to the location: simulator-mq
* ./emulator.sh start

1. Stop the simulator
   1. Go to the location: simulator-mq
   2. ./emulator.sh stop
2. Simulator logs will be generated at “simulator-mq/logs/emulator.log”
3. There are two modes through which simulator can connect to different queue manager.
   1. If mode = nft then it will use the Websphere queue manager
   2. If mode = dev then it will use the Active MQ.

Need to make this change in the emulator.sh file



1. If need to be profiling the simulator then need to make the highlighted change in the emulator.sh file.



1. To increase the memory of simulator then need to make the highlighted change in the emulator.sh file.



1. To change the location of log file then need to make the highlighted change in the emulator.sh file.



**Implementations:**

This is Spring Boot based application using the following feature:

* Spring In

The Spring Integration Java configuration and DSL provides a set of convenient builders and a fluent API that lets you configure Spring Integration message flows from Spring @Configuration classes.

The Java DSL for Spring Integration is essentially a facade for Spring Integration. The DSL provides a simple way to embed Spring Integration Message Flows into your application by using the fluent Builder pattern together with existing Java configuration from Spring Framework and Spring Integration. We also use and support lambdas (available with Java 8) to further simplify Java configuration.

The DSL is presented by the IntegrationFlows factory for the IntegrationFlowBuilder.

This produces the IntegrationFlow component, which should be registered as a Spring bean (by using the @Bean annotation). The builder pattern is used to express arbitrarily complex structures as a hierarchy of methods that can accept lambdas as arguments.

The IntegrationFlowBuilder only collects integration components (MessageChannel instances, AbstractEndpoint instances, and so on) in the IntegrationFlow bean for further parsing and registration of concrete beans in the application context by the IntegrationFlowBeanPostProcessor.

*@Bean*

**public** IntegrationFlow myFlow() {

**return** IntegrationFlows.from(integerSource::getAndIncrement,

c -> c.poller(Pollers.fixedRate(100)))

.channel("inputChannel")

.filter((Integer p) -> p > 0)

.transform(Object::toString)

.channel(MessageChannels.queue())

.get();

}

Endpoints are expressed as verbs in the DSL to improve readability. The following list includes the common DSL method names and the associated EIP endpoint:

* transform → Transformer
* filter → Filter
* handle → ServiceActivator
* split → Splitter
* aggregate → Aggregator
* route → Router
* bridge → Bridge

*@Bean*

**public** IntegrationFlow integerFlow() {

**return** IntegrationFlows.from("input")

.<String, Integer>transform(Integer::parseInt)

.get();

}

*@Bean*

**public** MessageChannel queueChannel() {

**return** MessageChannels.queue().get();

}

*@Bean*

**public** MessageChannel publishSubscribe() {

**return** MessageChannels.publishSubscribe().get();

}

*@Bean*

**public** IntegrationFlow channelFlow() {

**return** IntegrationFlows.from("input")

.fixedSubscriberChannel()

.channel("queueChannel")

.channel(publishSubscribe())

.channel(MessageChannels.executor("executorChannel", **this**.taskExecutor))

.channel("output")

.get();

}

* from("input") means "find and use the *MessageChannel* with the "input" id, or create one".
* fixedSubscriberChannel() produces an instance of FixedSubscriberChannel and registers it with a name of channelFlow.channel#0.
* channel("queueChannel") works the same way but uses an existing queueChannel bean.
* channel(publishSubscribe()) is the bean-method reference.
* channel(MessageChannels.executor("executorChannel", this.taskExecutor)) is the IntegrationFlowBuilder that exposes IntegrationComponentSpec to the ExecutorChannel and registers it as executorChannel.
* channel("output") registers the DirectChannel bean with output as its name, as long as no beans with this name already exist.

Application.properties

jmslistener.concurrency=7-15

ssw.jmslistener.concurrency=7-15

application.yaml

spring:

profiles:

active: "dev"

activemq:

in-memory: false

pool:

enabled: false

queue:

mq:

hostName: be-nft5-bnppi-db-01.clear2pay.com

port: 1414

channel: channel1

queueManager: opf.queue.manager

activemq:

brokerURL: tcp://localhost:61616

queuePrefetch: 10

FOREX:

REQUEST:

QUEUE: CIB.CASHMGMT.FXBUNDLE.APAC.ALL.EMX.PACS008.FX

RESPONSE:

metadata:

Destination: BPH2

Source: FX\_Bundle\_Incoming

Uid: BPH

DataObjectName: CCT\_PMT

SrcCtry: SG

DestCtry: SG

Version: 1.0.0

QUEUE: CIB.CASHMGMT.FXBUNDLE.APAC.EMX.BPH2.PACS002.FX

delay: 0

responseTemplateFile: conf/FxResponse.xml

SSW:

RESPONSE:

metadata:

Destination: BPH2

Source: SSW

Uid: BPH

DataObjectName: FLT\_RSP

Version: 1.0.0

requestResponseQueueMapping:

CIB.CASHMGMT.BPH2.SSW.IN.BPH2.EMX.PACS008: CIB.CASHMGMT.SSW.BPH2.IN.EMX.BPH2.PACS002

CIB.CASHMGMT.PFSV2.APAC.BPH2.EMX.PACS008.PFLRGRT: CIB.CASHMGMT.PFSV2.APAC.EMX.BPH2.PACS002.PFLRGRT

CIB.CASHMGMT.PFSV2.APAC.BPH2.EMX.PACS008.PFLRGPF: CIB.CASHMGMT.PFSV2.APAC.EMX.BPH2.PACS002.PFLRGRT

responseTemplateFile: conf/Pacs002\_SSW response.xml

trxTemplateFile: conf/Pacs002\_SSW\_trx.xml

tagToMergeWithTransaction: OrgnlGrpInfAndSts

requestResponseTagMapping:

TxId: OrgnlTxId

delay: 0

SSW\_IN:

REQUEST:

QUEUE: CIB.CASHMGMT.BPH2.SSW.IN.BPH2.EMX.PACS008

SSW\_APAC\_RGRT:

REQUEST:

QUEUE: CIB.CASHMGMT.PFSV2.APAC.BPH2.EMX.PACS008.PFLRGRT

SSW\_APAC\_RGPF:

REQUEST:

QUEUE: CIB.CASHMGMT.PFSV2.APAC.BPH2.EMX.PACS008.PFLRGPF

CHARGE:

REQUEST:

QUEUE: CIB.CASHMGMT.BPHV2.APAC.BPH2.EMX.PACS008.CHARGE

RESPONSE:

metadata:

Destination: BPH2

Source: TBS2

Uid: BPH

DataObjectName: CHG\_RSP

Version: 1.0.0

QUEUE: CIB.CASHMGMT.BPHV2.APAC.EMX.BPH2.PACS002.CHARGE

delay: 0

responseTemplateFile: conf/Pacs002\_Charge response.xml

trxTemplateFile: conf/Pacs002\_Charge\_trx.xml

tagToMergeWithTransaction: OrgnlGrpInfAndSts

requestResponseTagMapping:

InstrId: OrgnlInstrId

EndToEndId: OrgnlEndToEndId

TxId: OrgnlTxId

BVM:

REQUEST:

QUEUE: CIB.CASHMGMT.JOMP.MY.BPH2.EMX.PXML.BVMREQ

RESPONSE:

metadata:

Destination: BPH2

Source: BVM

Uid: BPH

DataObjectName: FLT\_RES

SrcCtry: MY

DestCtry: MY

Version: 1.0.0

QUEUE: CIB.CASHMGMT.JOMP.MY.EMX.BPH2.PXML.BVMRES

delay: 0

responseTemplateFile: conf/BvmResponse.xml

EARMRK:

RESPONSE:

metadata:

Destination: BPH2

Source: CAMP

DataObjectName: EMK\_RSP

requestResponseCFTPathMapping:

/tmp/bnppindia/apps/bph2/msg/CAMP/SEA/tmp: /tmp/bnppindia/apps/bph2/msg/CAMP/SEA/recv

/tmp/bnppindia/apps/bph2/msg/CAMP/IN/tmp: /tmp/bnppindia/apps/bph2/msg/CAMP/IN/recv

/tmp/bnppindia/apps/bph2/msg/CAMP/SG/tmp: /tmp/bnppindia/apps/bph2/msg/CAMP/SG/recv

requestResponseQueueMapping:

CIB.CASHMGMT.BPH2.CAMP.IN.BPH2.EMX.EMKREQ: CIB.CASHMGMT.CAMP.BPH2.IN.EMX.BPH2.EMKRSP

CIB.CASHMGMT.CAMPS.SG.ALL.EMX.PXML.EMKREQ: CIB.CASHMGMT.CAMPS.SG.EMX.ALL.PXML.EMKRSP

CIB.CASHMGMT.CAMPS.MY.ALL.EMX.PXML.EMKREQ: CIB.CASHMGMT.CAMPS.MY.EMX.ALL.PXML.EMKRSP

CIB.CASHMGMT.CAMPS.TH.ALL.EMX.PXML.EMKREQ: CIB.CASHMGMT.CAMPS.TH.EMX.ALL.PXML.EMKRSP

CIB.CASHMGMT.CAMPS.VN.ALL.EMX.PXML.EMKREQ: CIB.CASHMGMT.CAMPS.VN.EMX.ALL.PXML.EMKRSP

CIB.CASHMGMT.EMXMEDIATION.ALL.ALL.APX.ALL.ALLREQ: CIB.CASHMGMT.EMXAT2.MY.EMX.BPH2.PXML.EMKRSP

METADATAHANLDER: EamkMetadataHandler

TRXTAG: CIB.CASHMGMT.CINT.IN.EMX.BPH.EMKRSP

RULES: delay

responseTemplateFile: conf/EarmrkReq2Template.xml

trxTemplateFile: conf/EarmrkTxRqstInf.xml

tagToMergeWithTransaction: RqstPvtInf

requestResponseTagMapping:

TxRef: TxRef

RqstRef: RqstRef

delay: 0

EARMRK\_IN:

REQUEST:

FOLDER: /tmp/bnppindia/apps/bph2/msg/CAMP/IN/tmp

QUEUE: CIB.CASHMGMT.BPH2.CAMP.IN.BPH2.EMX.EMKREQ

EARMRK\_SG:

REQUEST:

FOLDER: /tmp/bnppindia/apps/bph2/msg/CAMP/SG/tmp

QUEUE: CIB.CASHMGMT.CAMPS.SG.ALL.EMX.PXML.EMKREQ

EARMRK\_MY:

REQUEST:

FOLDER: /tmp/bnppindia/apps/bph2/msg/CAMP/SEA/tmp

QUEUE: CIB.CASHMGMT.CAMPS.MY.ALL.EMX.PXML.EMKREQ

EARMRK\_RTPM\_MY:

REQUEST:

FOLDER: /tmp/bnppindia/apps/bph2/msg/CAMP/SEA/tmp

QUEUE: CIB.CASHMGMT.EMXMEDIATION.ALL.ALL.APX.ALL.ALLREQ

EARMRK\_TH:

REQUEST:

FOLDER: /tmp/bnppindia/apps/bph2/msg/CAMP/SEA/tmp

QUEUE: CIB.CASHMGMT.CAMPS.TH.ALL.EMX.PXML.EMKREQ

EARMRK\_VN:

REQUEST:

FOLDER: /tmp/bnppindia/apps/bph2/msg/CAMP/SEA/tmp

QUEUE: CIB.CASHMGMT.CAMPS.VN.ALL.EMX.PXML.EMKREQ

EMULATOR:

RULE:

- name: delay

property:

duration: 5000.0

- name: multiresponse

property:

file1: PAIN.001.001.02\_Response1.xml

file2: PAIN.001.001.02\_Response2.xml

filecount: 2

- name: 'noresponse '

- name: headerreconfail

- name:

payloadreconfail: req:/Document/pain.001.001.02/GrpHdr/InitgPty/Nm

property:

filename: PAIN.001.001.02\_Response\_fail.xml

JMSHEADER:

metadata:

AppInstanceID: EMX

DestCountry: IN

Destination: BPH2

GUID: 00

Source: CAMP

SrcCountry: IN

Timestamp: 00

UID: 00

VersionNo: 1.0.0

Target: AT2

inbound:

failed:

path: inbound/failed

file:

poller:

fixed:

delay: 3

max:

messages:

per:

poll: 100

thread:

pool:

size: 10

filename:

regex: ([^\s]+(\.(?i)(tkn))$)

out:

path: inbound/out

processed:

path: inbound/processed

read:

path: inbound/read

write:

path: inbound/write

install:

path: /mnt/sdb1/dev/bnppindia-0.0.1/setup/Emulator

out:

filename:

suffix: .txt

---

spring:

profiles: dev

EARMRK:

RESPONSE:

metadata:

Destination: BPH2

Source: CAMP

DataObjectName: EMK\_RSP

requestResponseCFTPathMapping:

/tmp/bnppindia/apps/bph2/msg/CAMP/SEA/tmp: /tmp/bnppindia/apps/bph2/msg/CAMP/SEA/recv

/tmp/bnppindia/apps/bph2/msg/CAMP/IN/tmp: /tmp/bnppindia/apps/bph2/msg/CAMP/IN/recv

/tmp/bnppindia/apps/bph2/msg/CAMP/SG/tmp: /tmp/bnppindia/apps/bph2/msg/CAMP/SG/recv

requestResponseQueueMapping:

CIB.CASHMGMT.BPH2.CAMP.IN.BPH2.EMX.EMKREQ: CIB.CASHMGMT.CAMP.BPH2.IN.EMX.BPH2.EMKRSP

CIB.CASHMGMT.CAMPS.SG.ALL.EMX.PXML.EMKREQ: CIB.CASHMGMT.CAMPS.SG.EMX.ALL.PXML.EMKRSP

CIB.CASHMGMT.CAMPS.MY.ALL.EMX.PXML.EMKREQ: CIB.CASHMGMT.CAMPS.MY.EMX.ALL.PXML.EMKRSP

CIB.CASHMGMT.CAMPS.TH.ALL.EMX.PXML.EMKREQ: CIB.CASHMGMT.CAMPS.TH.EMX.ALL.PXML.EMKRSP

CIB.CASHMGMT.CAMPS.VN.ALL.EMX.PXML.EMKREQ: CIB.CASHMGMT.CAMPS.VN.EMX.ALL.PXML.EMKRSP

CIB.CASHMGMT.EMXMEDIATION.ALL.ALL.APX.ALL.ALLREQ: CIB.CASHMGMT.EMXAT2.MY.EMX.BPH2.PXML.EMKRSP

METADATAHANLDER: EamkMetadataHandler

TRXTAG: CIB.CASHMGMT.CINT.IN.EMX.BPH.EMKRSP

RULES: delay

responseTemplateFile: conf/EarmrkReq2Template.xml

trxTemplateFile: conf/EarmrkTxRqstInf.xml

tagToMergeWithTransaction: RqstPvtInf

requestResponseTagMapping:

TxRef: TxRef

RqstRef: RqstRef

delay: 0

EARMRK\_IN:

REQUEST:

FOLDER: /tmp/bnppindia/apps/bph2/msg/CAMP/IN/tmp

QUEUE: CIB.CASHMGMT.BPH2.CAMP.IN.BPH2.EMX.EMKREQ

EARMRK\_SG:

REQUEST:

FOLDER: /tmp/bnppindia/apps/bph2/msg/CAMP/SG/tmp

QUEUE: CIB.CASHMGMT.CAMPS.SG.ALL.EMX.PXML.EMKREQ

EARMRK\_MY:

REQUEST:

FOLDER: /tmp/bnppindia/apps/bph2/msg/CAMP/SEA/tmp

QUEUE: CIB.CASHMGMT.CAMPS.MY.ALL.EMX.PXML.EMKREQ

EARMRK\_RTPM\_MY:

REQUEST:

FOLDER: /tmp/bnppindia/apps/bph2/msg/CAMP/SEA/tmp

QUEUE: CIB.CASHMGMT.EMXMEDIATION.ALL.ALL.APX.ALL.ALLREQ

EARMRK\_TH:

REQUEST:

FOLDER: /tmp/bnppindia/apps/bph2/msg/CAMP/SEA/tmp

QUEUE: CIB.CASHMGMT.CAMPS.TH.ALL.EMX.PXML.EMKREQ

EARMRK\_VN:

REQUEST:

FOLDER: /tmp/bnppindia/apps/bph2/msg/CAMP/SEA/tmp

QUEUE: CIB.CASHMGMT.CAMPS.VN.ALL.EMX.PXML.EMKREQ

---

spring:

profiles: nft

EARMRK:

RESPONSE:

metadata:

Destination: BPH2

Source: CAMP

DataObjectName: EMK\_RSP

requestResponseCFTPathMapping:

/apps/bph2/msg/CAMP/SEA/tmp: /apps/bph2/msg/CAMP/SEA/recv

/apps/bph2/msg/CAMP/IN/tmp: /apps/bph2/msg/CAMP/IN/recv

/apps/bph2/msg/CAMP/SG/tmp: /apps/bph2/msg/CAMP/SG/recv

requestResponseQueueMapping:

CIB.CASHMGMT.BPH2.CAMP.IN.BPH2.EMX.EMKREQ: CIB.CASHMGMT.CAMP.BPH2.IN.EMX.BPH2.EMKRSP

CIB.CASHMGMT.CAMPS.SG.ALL.EMX.PXML.EMKREQ: CIB.CASHMGMT.CAMPS.SG.EMX.ALL.PXML.EMKRSP

CIB.CASHMGMT.CAMPS.MY.ALL.EMX.PXML.EMKREQ: CIB.CASHMGMT.CAMPS.MY.EMX.ALL.PXML.EMKRSP

CIB.CASHMGMT.CAMPS.TH.ALL.EMX.PXML.EMKREQ: CIB.CASHMGMT.CAMPS.TH.EMX.ALL.PXML.EMKRSP

CIB.CASHMGMT.CAMPS.VN.ALL.EMX.PXML.EMKREQ: CIB.CASHMGMT.CAMPS.VN.EMX.ALL.PXML.EMKRSP

CIB.CASHMGMT.EMXMEDIATION.ALL.ALL.APX.ALL.ALLREQ: CIB.CASHMGMT.EMXAT2.MY.EMX.BPH2.PXML.EMKRSP

METADATAHANLDER: EamkMetadataHandler

TRXTAG: CIB.CASHMGMT.CINT.IN.EMX.BPH.EMKRSP

RULES: delay

responseTemplateFile: conf/EarmrkReq2Template.xml

trxTemplateFile: conf/EarmrkTxRqstInf.xml

tagToMergeWithTransaction: RqstPvtInf

requestResponseTagMapping:

TxRef: TxRef

RqstRef: RqstRef

delay: 0

EARMRK\_IN:

REQUEST:

FOLDER: /apps/bph2/msg/CAMP/IN/tmp

QUEUE: CIB.CASHMGMT.BPH2.CAMP.IN.BPH2.EMX.EMKREQ

EARMRK\_SG:

REQUEST:

FOLDER: /apps/bph2/msg/CAMP/SG/tmp

QUEUE: CIB.CASHMGMT.CAMPS.SG.ALL.EMX.PXML.EMKREQ

EARMRK\_MY:

REQUEST:

FOLDER: /apps/bph2/msg/CAMP/SEA/tmp

QUEUE: CIB.CASHMGMT.CAMPS.MY.ALL.EMX.PXML.EMKREQ

EARMRK\_RTPM\_MY:

REQUEST:

FOLDER: /apps/bph2/msg/CAMP/SEA/tmp

QUEUE: CIB.CASHMGMT.EMXMEDIATION.ALL.ALL.APX.ALL.ALLREQ

EARMRK\_TH:

REQUEST:

FOLDER: /apps/bph2/msg/CAMP/SEA/tmp

QUEUE: CIB.CASHMGMT.CAMPS.TH.ALL.EMX.PXML.EMKREQ

EARMRK\_VN:

REQUEST:

FOLDER: /apps/bph2/msg/CAMP/SEA/tmp

QUEUE: CIB.CASHMGMT.CAMPS.VN.ALL.EMX.PXML.EMKREQ

MessageConfiguration Class

@Configuration

**public** **class** MessagingConfiguration {

/\*\*

\* Create MQ Connection Factory

\* **@param** hostName

\* **@param** port

\* **@param** channel

\* **@param** queueManager

\* **@return** connectionFactory the connectionFactory

\*/

@Bean(name="mqConnectionFactory")

@ConditionalOnExpression("#{environment.getProperty('mode').contains('nft')}")

**public** ConnectionFactory mqConnectionFactory(@Value("${queue.mq.hostName}") String hostName,

@Value("${queue.mq.port}") **int** port, @Value("${queue.mq.channel}") String channel,

@Value("${queue.mq.queueManager}") String queueManager) {

MQQueueConnectionFactory mqQueueConnectionFactory = **new** MQQueueConnectionFactory();

mqQueueConnectionFactory.setHostName(hostName);

**try** {

mqQueueConnectionFactory.setTransportType(WMQConstants.***ADMIN\_QUEUE\_DOMAIN***);

mqQueueConnectionFactory.setCCSID(1208);

mqQueueConnectionFactory.setChannel(channel);

mqQueueConnectionFactory.setPort(port);

mqQueueConnectionFactory.setQueueManager(queueManager);

} **catch** (Exception e) {

e.printStackTrace();

}

**return** mqQueueConnectionFactory;

}

@Bean(name="activeMQConnectionFactory")

@ConditionalOnExpression("#{environment.getProperty('mode').contains('dev')}")

**public** ConnectionFactory activeMQConnectionFactory(@Value("${queue.activemq.brokerURL}") String brokerURL,ActiveMQPrefetchPolicy activeMQPrefetchPolicy){

ActiveMQConnectionFactory connectionFactory = **new** ActiveMQConnectionFactory();

connectionFactory.setBrokerURL(brokerURL);

connectionFactory.setPrefetchPolicy(activeMQPrefetchPolicy);

connectionFactory.setTrustedPackages(Arrays.*asList*("com.clear2pay","java.util","com.clear2pay.indhub.emulator.earmarking.jaxb.Document","org.w3c.dom.Document"));

**return** connectionFactory;

}

@Bean

@ConditionalOnExpression("#{environment.getProperty('mode').contains('dev')}")

**public** ActiveMQPrefetchPolicy activeMQPrefetchPolicy(@Value("${queue.activemq.queuePrefetch}") String queuePrefetch){

ActiveMQPrefetchPolicy activeMQPrefetchPolicy=**new** ActiveMQPrefetchPolicy();

activeMQPrefetchPolicy.setQueuePrefetch(10);

**return** activeMQPrefetchPolicy;

}

@Bean

**public** MessageConverter oxmJmsMessageConverter() {

MarshallingMessageConverter oxmJmsMessageConverter

= **new** MarshallingMessageConverter();

Jaxb2Marshaller jaxbMarshaller = **new** Jaxb2Marshaller ();

jaxbMarshaller.setPackagesToScan("com.clear2pay.indhub.emulator.earmarking.jaxb.Document");

jaxbMarshaller.setClassesToBeBound(com.clear2pay.indhub.emulator.earmarking.jaxb.Document.**class**);

oxmJmsMessageConverter.setUnmarshaller(jaxbMarshaller);

oxmJmsMessageConverter.setMarshaller(jaxbMarshaller);

**return** oxmJmsMessageConverter;

}

@Bean

**public** MessageConverter simpleMessageConverter() {

SimpleMessageConverter converter = **new** SimpleMessageConverter();

**return** converter;

}

@Bean

**public** DefaultJmsListenerContainerFactory jmsListenerContainerFactory(ConnectionFactory connectionFactory) {

DefaultJmsListenerContainerFactory factory = **new** DefaultJmsListenerContainerFactory();

factory.setConnectionFactory(connectionFactory);

factory.setMessageConverter(simpleMessageConverter());

// factory.setTaskExecutor(taskJMSExecutor());

// factory.setConcurrency("1-1");

**return** factory;

}

@Bean

TaskExecutor taskJMSExecutor() {

ThreadPoolTaskExecutor taskExecutor = **new** ThreadPoolTaskExecutor();

taskExecutor.setCorePoolSize(10);

**return** taskExecutor;

}

@Bean

**public** JmsTemplate jmsTemplate(ConnectionFactory connectionFactory) {

JmsTemplate template = **new** JmsTemplate();

template.setConnectionFactory(connectionFactory);

template.setMessageConverter(simpleMessageConverter());

**return** template;

}

}

FilePollingConfiguration

@Configuration

**public** **class** FilePollingConfiguration {

@Bean(name="inboundReadDirectory")

**public** File inboundReadDirectory(@Value("${inbound.read.path}") String path) {

**return** makeDirectory(path);

}

@Bean(name="inboundFailedDirectory")

**public** File inboundFailedDirectory(@Value("${inbound.failed.path}") String path) {

**return** makeDirectory(path);

}

@Bean(name="inboundOutDirectory")

**public** File inboundOutDirectory(@Value("${inbound.out.path}") String path) {

**return** makeDirectory(path);

}

@Bean(name="earmrkReadDirectory")

**public** File earmrkReadDirectory(@Value("${EARMRK\_IN.REQUEST.FOLDER}") String path) {

**return** makeDirectory(path);

}

@Bean(name="earmrkReadDirectorySG")

**public** File earmrkReadDirectorySG(@Value("${EARMRK\_SG.REQUEST.FOLDER}") String path) {

**return** makeDirectory(path);

}

@Bean(name="earmrkReadDirectoryApac")

**public** File earmrkReadDirectoryApac(@Value("${EARMRK\_MY.REQUEST.FOLDER}") String path) {

**return** makeDirectory(path);

}

**private** File makeDirectory(String path) {

File file = **new** File(path);

file.mkdirs();

**return** file;

}

}

@Configuration

**class** FilePollingIntegrationFlow {

@Autowired

**public** File earmrkReadDirectory;

@Autowired

**public** File earmrkReadDirectoryApac;

@Autowired

**public** File earmrkReadDirectorySG;

@Autowired

**private** ApplicationContext applicationContext;

@Bean

**public** IntegrationFlow earmkInboundFileIntegration(@Value("${inbound.file.poller.fixed.delay}") **long** period,

@Value("${inbound.file.poller.max.messages.per.poll}") **int** maxMessagesPerPoll,

TaskExecutor taskExecutor,

MessageSource<File> earmrkFileReadingMessageSourceIN) {

**return** IntegrationFlows.*from*(earmrkFileReadingMessageSourceIN,

c -> c.poller(Pollers.*fixedDelay*(period)

.taskExecutor(taskExecutor)

.maxMessagesPerPoll(maxMessagesPerPoll)

.transactionSynchronizationFactory(earmkTxSynchronizationFactory())

.transactional(transactionManager())))

.transform(Transformers.*fileToString*())

.handle("ermkFileProcessor", "process")

.channel(AppConfig.***INBOUND\_CHANNEL***)

.get();

}

@Bean

**public** IntegrationFlow earmkInboundFileIntegrationApac(@Value("${inbound.file.poller.fixed.delay}") **long** period,

@Value("${inbound.file.poller.max.messages.per.poll}") **int** maxMessagesPerPoll,

TaskExecutor taskExecutor,

MessageSource<File> earmrkFileReadingMessageSourceAPAC) {

**return** IntegrationFlows.*from*(earmrkFileReadingMessageSourceAPAC,

c -> c.poller(Pollers.*fixedDelay*(period)

.taskExecutor(taskExecutor)

.maxMessagesPerPoll(maxMessagesPerPoll)

.transactionSynchronizationFactory(earmkTxSynchronizationFactory())

.transactional(transactionManager())))

.transform(Transformers.*fileToString*())

.handle("ermkFileProcessor", "process")

.channel(AppConfig.***INBOUND\_CHANNEL***)

.get();

}

@Bean

**public** IntegrationFlow earmkInboundFileIntegrationSG(@Value("${inbound.file.poller.fixed.delay}") **long** period,

@Value("${inbound.file.poller.max.messages.per.poll}") **int** maxMessagesPerPoll,

TaskExecutor taskExecutor,

MessageSource<File> earmrkFileReadingMessageSourceSG) {

**return** IntegrationFlows.*from*(earmrkFileReadingMessageSourceSG,

c -> c.poller(Pollers.*fixedDelay*(period)

.taskExecutor(taskExecutor)

.maxMessagesPerPoll(maxMessagesPerPoll)

.transactionSynchronizationFactory(earmkTxSynchronizationFactory())

.transactional(transactionManager())))

.transform(Transformers.*fileToString*())

.handle("ermkFileProcessor", "process")

.channel(AppConfig.***INBOUND\_CHANNEL***)

.get();

}

@Bean

TaskExecutor taskExecutor(@Value("${inbound.file.poller.thread.pool.size}") **int** poolSize) {

ThreadPoolTaskExecutor taskExecutor = **new** ThreadPoolTaskExecutor();

taskExecutor.setCorePoolSize(poolSize);

**return** taskExecutor;

}

@Bean

PseudoTransactionManager transactionManager() {

**return** **new** PseudoTransactionManager();

}

@Bean

**public** FileProcessor fileProcessor() {

**return** **new** FileProcessor();

}

@Bean

**public** EarmarkingFileProcessor ermkFileProcessor() {

**return** **new** EarmarkingFileProcessor();

}

@Bean

TransactionSynchronizationFactory earmkTxSynchronizationFactory() {

ExpressionParser parser = **new** SpelExpressionParser();

ExpressionEvaluatingTransactionSynchronizationProcessor syncProcessor =

**new** ExpressionEvaluatingTransactionSynchronizationProcessor();

syncProcessor.setBeanFactory(applicationContext.getAutowireCapableBeanFactory());

syncProcessor.setAfterCommitExpression(parser.parseExpression("payload.renameTo(new java.io.File(@earmrkReadDirectory.path " +

" + T(java.io.File).separator + payload.name))"));

syncProcessor.setAfterRollbackExpression(parser.parseExpression("payload.renameTo(new java.io.File(@earmrkReadDirectory.path " +

" + T(java.io.File).separator + payload.name))"));

**return** **new** DefaultTransactionSynchronizationFactory(syncProcessor);

}

@Bean

**public** FileReadingMessageSource earmrkFileReadingMessageSourceIN(@Value("${inbound.filename.regex}") String regex) {

FileReadingMessageSource source = **new** FileReadingMessageSource();

source.setDirectory(**this**.earmrkReadDirectory);

source.setAutoCreateDirectory(**true**);

CompositeFileListFilter<File> filter = **new** CompositeFileListFilter<>(

Arrays.*asList*(**new** AcceptOnceFileListFilter<File>(),

**new** RegexPatternFileListFilter(regex))

);

source.setFilter(filter);

**return** source;

}

@Bean

**public** FileReadingMessageSource earmrkFileReadingMessageSourceSG(@Value("${inbound.filename.regex}") String regex) {

FileReadingMessageSource source = **new** FileReadingMessageSource();

source.setDirectory(**this**.earmrkReadDirectorySG);

source.setAutoCreateDirectory(**true**);

CompositeFileListFilter<File> filter = **new** CompositeFileListFilter<>(

Arrays.*asList*(**new** AcceptOnceFileListFilter<File>(),

**new** RegexPatternFileListFilter(regex))

);

source.setFilter(filter);

**return** source;

}

@Bean

**public** FileReadingMessageSource earmrkFileReadingMessageSourceAPAC(@Value("${inbound.filename.regex}") String regex) {

FileReadingMessageSource source = **new** FileReadingMessageSource();

source.setDirectory(**this**.earmrkReadDirectoryApac);

source.setAutoCreateDirectory(**true**);

CompositeFileListFilter<File> filter = **new** CompositeFileListFilter<>(

Arrays.*asList*(**new** AcceptOnceFileListFilter<File>(),

**new** RegexPatternFileListFilter(regex))

);

source.setFilter(filter);

**return** source;

}

}

@Component

**public** **class** MessageProcessingIntegrationFlow {

@Autowired

**public** File inboundOutDirectory;

@Bean

**public** IntegrationFlow writeToFile(@Qualifier("fileWritingMessageHandler") MessageHandler fileWritingMessageHandler) {

**return** IntegrationFlows.*from*(AppConfig.***INBOUND\_CHANNEL***)

.transform(m -> **new** StringBuilder((String)m).reverse().toString())

.handle(fileWritingMessageHandler)

.handle(loggingHandler())

.get();

}

@Bean (name = "fileWritingMessageHandler")

**public** MessageHandler fileWritingMessageHandler() {

FileWritingMessageHandler handler = **new** FileWritingMessageHandler(inboundOutDirectory);

handler.setAutoCreateDirectory(**true**);

**return** handler;

}

@Bean

**public** MessageHandler loggingHandler() {

LoggingHandler logger = **new** LoggingHandler("INFO");

logger.setShouldLogFullMessage(**true**);

**return** logger;

}

}

@ConfigurationProperties(prefix = "EARMRK.RESPONSE")

@Component("earmarkingProcessor")

**public** **class** EarmarkingProcessor **implements** InterfaceProcessor{

**private** Map<String,String> requestResponseTagMapping;

**private** String tagToMergeWithTransaction;

@Autowired

**private** String earMarkingContainerResponseTemplate;

@Autowired

**private** String earMarkingTransactionResponseTemplate;

@Override

**public** String process(**byte**[] requestPayload) {

Map<String, List<String>> requestedTagValuesMapping=extractRequestInfo(requestPayload);

StringBuilder transactionResponses=**new** StringBuilder();

**int** size=((Collection<List<String>>)requestedTagValuesMapping.values()).iterator().next().size();

**for** (**int** i = 0; i < size; i++) {

StringBuilder transactionResponse = **new** StringBuilder(earMarkingTransactionResponseTemplate);

**final** **int** index=i;

requestedTagValuesMapping.entrySet().forEach(entry->{

String responseTag = requestResponseTagMapping.get(entry.getKey());

String resposneValueToBeRepalce = "%" + responseTag + "%";

transactionResponse.replace(transactionResponse.indexOf(resposneValueToBeRepalce),

transactionResponse.indexOf(resposneValueToBeRepalce) + resposneValueToBeRepalce.length(),

entry.getValue().get(index));

});

transactionResponses.append(transactionResponse);

}

StringBuilder mainResponseBuilder = **new** StringBuilder(earMarkingContainerResponseTemplate);

String endTagToBeMergeWithTransaction=String.*format*("</"+tagToMergeWithTransaction+">");

mainResponseBuilder.insert(mainResponseBuilder.indexOf(endTagToBeMergeWithTransaction)+endTagToBeMergeWithTransaction.length()+1, transactionResponses);

**return** mainResponseBuilder.toString();

}

**private** Map<String, List<String>> extractRequestInfo(**byte**[] bytes) {

Map<String, List<String>> tagValuesMapping = **new** HashMap(requestResponseTagMapping.keySet().size());

**try** {

XMLInputFactory inputFactory = XMLInputFactory.*newInstance*();

XMLEventReader eventReader = inputFactory.createXMLEventReader(**new** ByteArrayInputStream(bytes));

**while** (eventReader.hasNext()) {

XMLEvent event = eventReader.nextEvent();

// reach the start of an item

**if** (event.isStartElement()) {

// data

**for** (String tagTobeExtract : requestResponseTagMapping.keySet())

**if** (event.asStartElement().getName().getLocalPart().equals(tagTobeExtract)) {

event = eventReader.nextEvent();

String value = event.asCharacters().getData();

**if** (tagValuesMapping.containsKey(tagTobeExtract)) {

tagValuesMapping.get(tagTobeExtract).add(value);

} **else** {

List<String> values = **new** ArrayList<>();

values.add(value);

tagValuesMapping.put(tagTobeExtract, values);

}

**break**;

}

}

}

} **catch** (FactoryConfigurationError e) {

e.printStackTrace();

} **catch** (XMLStreamException e) {

e.printStackTrace();

}

**return** tagValuesMapping;

}

**public** Map<String, String> getRequestResponseTagMapping() {

**return** requestResponseTagMapping;

}

**public** **void** setRequestResponseTagMapping(Map<String, String> requestResponseTagMapping) {

**this**.requestResponseTagMapping = requestResponseTagMapping;

}

**public** String getTagToMergeWithTransaction() {

**return** tagToMergeWithTransaction;

}

**public** **void** setTagToMergeWithTransaction(String tagToMergeWithTransaction) {

**this**.tagToMergeWithTransaction = tagToMergeWithTransaction;

}

}

@Component

@ConfigurationProperties(prefix = "SSW.RESPONSE")

**public** **class** SSWListener {

**private** **static** **final** Logger ***LOGGER*** = LoggerFactory.*getLogger*(SSWListener.**class**);

@Resource(name="sswProcessor")

**private** InterfaceProcessor processor;

**private** **final** ResponseService responseService;

**private** **int** delay;

/\*\*

\*

\* **@param** responseService

\*/

@Autowired

**public** SSWListener(ResponseService responseService) {

**this**.responseService = responseService;

}

**private** Map<String,String> requestResponseQueueMapping;

**private** Map<String, String> metadata;

@Autowired

**private** JmsTemplate jmsTemplate;

@Autowired

**private** JMSHeaderProcessor jmsHeaderProcessor;

**private** **volatile** String charset = "UTF-8";

@JmsListeners(value={@JmsListener(containerFactory = "jmsListenerContainerFactory", destination = "${SSW\_IN.REQUEST.QUEUE}",concurrency="${ssw.jmslistener.concurrency}"),

@JmsListener(containerFactory = "jmsListenerContainerFactory", destination = "${SSW\_APAC\_RGRT.REQUEST.QUEUE}",concurrency="${ssw.jmslistener.concurrency}"),

@JmsListener(containerFactory = "jmsListenerContainerFactory", destination = "${SSW\_APAC\_RGPF.REQUEST.QUEUE}",concurrency="${ssw.jmslistener.concurrency}")})

**public** **void** receiveMessage(Message<?> message) {

**try** {

**byte**[] requestPayload = **null**;

MessageHeaders messageheaders=message.getHeaders();

Queue requestQueue=((Queue)messageheaders.get("jms\_destination"));

metadata.put(BNPPJMSHeaderProperties.EmxHeaderProp.***SOURCECOUNTRY***.getHeaderName(),(String)messageheaders.get(BNPPJMSHeaderProperties.EmxHeaderProp.***SOURCECOUNTRY***.getHeaderName()));

metadata.put(BNPPJMSHeaderProperties.EmxHeaderProp.***DESTINATIONCOUNTRY***.getHeaderName(),(String)messageheaders.get(BNPPJMSHeaderProperties.EmxHeaderProp.***DESTINATIONCOUNTRY***.getHeaderName()));

Object payload = message.getPayload();

CommonUtil.*generateLogs*("SSW Request received",messageheaders);

**if** (payload **instanceof** **byte**[]) {

requestPayload = (**byte**[]) payload;

}

**else** **if** (payload **instanceof** String) {

**try** {

requestPayload = ((String) payload).getBytes(**this**.charset);

}

**catch** (UnsupportedEncodingException e) {

**throw** **new** MessageHandlingException(message, e);

}

}

**else** {

**throw** **new** MessageHandlingException(message,

"HdfsTextFileWriter expects " +

"either a byte array or String payload, but received: " + payload.getClass());

}

String sswResponse=processor.process(requestPayload);

MessageHeaders headers = message.getHeaders();

**if** (StringUtils.*isNotBlank*(sswResponse)) {

**if**(delay>0){

Thread.*sleep*(delay);

}

**this**.jmsTemplate.convertAndSend(CommonUtil.*getResponseQueue*(requestResponseQueueMapping, requestQueue.getQueueName()), sswResponse,

**new** MessagePostProcessor() {

**public** javax.jms.Message postProcessMessage(javax.jms.Message message)

**throws** JMSException {

CommonUtil.*generateLogs*("SSW Response sent",headers);

jmsHeaderProcessor.prepareHeader(message,

jmsHeaderProcessor.getMetadata(getMetadata()));

**if** (responseService.isHeaderReconFail()) {

message.setStringProperty("Guid",

**new** StringBuffer(headers.get("Uid").toString()).reverse().toString());

} **else** {

message.setStringProperty("Guid",

headers.get("Uid").toString());

}

**return** message;

}

});

} **else**

System.***out***.println("Error : Response message is null");

} **catch** (Exception e) {

e.printStackTrace();

}

}

**public** **int** getDelay() {

**return** delay;

}

**public** **void** setDelay(**int** delay) {

**this**.delay = delay;

}

**public** Map<String, String> getMetadata() {

**return** metadata;

}

**public** **void** setMetadata(Map<String, String> metadata) {

**this**.metadata = metadata;

}

**public** Map<String, String> getRequestResponseQueueMapping() {

**return** requestResponseQueueMapping;

}

**public** **void** setRequestResponseQueueMapping(Map<String, String> requestResponseQueueMapping) {

**this**.requestResponseQueueMapping = requestResponseQueueMapping;

}

}