**WFG\_EquipSales**

**1) WFL\_EquipSales\_Collect\_S3**

**Steps:**

1) Collect Files from JDE from location ***/JDE/BRIM/ES*** and collect ***CSV files that start with ES***

Eg. ES\_data.csv

2) Route(Analysis):

a) Before we do the processing we first print data regarding Port details, Thirsty enablitity , and so on.

b) We first send the file to location:

***/usr/sap/interfaces/MZP/EquipmentSales/collection/input***

c) We also the archive the file based on a flag(T- SEND & F - DONT)

***/usr/sap/interfaces/MZP/EquipmentSales/collection/archive***

d) We also then log the success or error based on outcome of processing.

**2) WFL\_EquipSales\_Sort\_OrderNo**

**Steps:**

1) Collect files from **WFL\_EquipSales\_Collect\_S3** output location :

***/usr/sap/interfaces/MZP/EquipmentSales/collection/input***

2) ***Sort\_OrderNo(Analysis)*** :

a) Sames as in Collect\_S3 we print the constants.

b) In case of the header we skip , for trailer we do a check for whether the Record count in file matches that of the trailer content , if not we log an error.

c) For the records in the csv, they are converted into xml format and then stored into a list, and also added into a map , (this map is present because different XML's are generated for different OrderNo's)

d) Next if the records OrderType = US or CF , AND LineType = EJ

It is an Onlocation,

***BussinessScenrio :* (OnLocationSales, we send a removal request for Equipment Lease process so we can terminate the lease on the same day the equipment has been sold. For Freestyle equipment we also create an installation request for EL)**

**Note: For Freestyle and On Location a csv is generated and sent back to JDE.**

**JDE** *from d.1 to f will be a shortform for path* ***/JDE/BRIM/EL***

d.1 Header is created and routed to JDE if hasn’t been created already.

d.2 We make a new Record for the csv for removal scenario( docType = IE and lineType = RS) and Routed to JDE.

e) If the EquipmentType is JD, JE, JF, JJ, JL, JM, JP, JT, MT and MV

Its is FreeStyle,

e.1 Header is created and routed to JDE if hasnt been created already.

e.2 We make a new Record for the csv for installment scenario( docType = UL and lineType = ES), then routed to JDE.

***drain :***

f) Create Trailer record for the csv and route to JDE

g) For different OrderNo we will have different list of records , 1 XML per workorder with all the EN/ES/IT lines present.

which are stored in :

***/usr/sap/interfaces/MZP/EquipmentSales/processing/input***

**3) WFL\_EquipSales\_Lookup\_WO**

1)Collect files from **WFL\_EquipSales\_Sort\_OrderNo** output location:

***/usr/sap/interfaces/MZP/EquipmentSales/processing/input***

2)***Filter\_LineType(Analysis)*** :

a) We display the constants that were present previously

b) Display the contents of the XML

c) Traverse the list of records we then check if the linetype is EN , ES , EJ , EK and IT , if so we then update the ProcessedIndicator =Y , else its N , in that case its discarded

3) ***Validate\_Mandatory\_Fields(Analysis):***

a) We traverse the records in the XML

if the values in the records such as OrderNo , OrderType, SoldTo , SalesOrderDate, InstallDate, LineType, ICCode, ShippedQuantity are not null or Empty.

and if LineType = IT, if the records ICCode is A- , B- , C- or 2- , then its a valid record

b) if the fields are valid , we then update the comments as "All Mandatory Fields present.", ProcessedIndictor = Y , and route to next Agent(***Route\_to\_Thirsty*)** .

c) if not valid , then we first check if the ECS Collection Agent (ECS\_Invalid\_XML) is valid , if yes then we route the XML record or other words the list to it.

c.b) else we route to

***/usr/sap/interfaces/MZP/EquipmentSales/processing/validation/extended\_cost***

***4) Route\_To\_Thirsty(Analysis) :***

a)We validate if the ICCode of all the values in the list is A-, B- , C- or 2-( these ICCode are meant for thirsty) , this check meant to check xml does consist of an IT line (same check is already done for

b) if so we create a ConsumeCycleUDR(this means we are going to create a request to Thirsty)

in consumecycleUdr.data we send a wrapper.

wrapper contains details such as *filename, aggregation\_key ( this is generated using the OrderNo , OrderType, SoldTo, ShipTo, InstallDate and SalesOrderDate of the first record that doesnt have LineType!=IT), AND also ListOf all the records for that OrderNo.*

And route to ***WFL\_EquipSales\_Lookup\_WO\_RT.***

c) if we cant any records with A-, B-, C- or 2-, ICCode then store in location:

***/usr/sap/interfaces/MZP/EquipmentSales/processing/output***

**4)WFL\_EquipSales\_Lookup\_WO\_RT:**

**Request Flow:**

**1.WFB\_Handler(Analysis):**

a) Since we created a ConsumeCycleUDR in *Route\_To\_Thirsty,* we check if input is ConsumeCycleUDR, if it is , then we print out the contents of wrapper and then route the wrapper to *Query\_WO\_State.*

**2. Query\_WO\_State(Aggregation):**

a) *In sessionInit*: the session has variable list called ***toSend\_ListEquipSalesRecords,*** in this vairable we store the list of all the records that have LineType = IT and ICCode is starting with ["A-","B-","C-","2-"].

b) If this variable list has content ,then the first record is picked from the list and we update the session.record field for it and also remove it from the list and route to *Prepare\_Query.*

c) else we send it back to *WFB\_Handler.*

**3. Prepare\_Query(Analysis):**

a) We first check if the record from *Query\_WO\_State* is having WorkOrderNumber is null or empty.

b) if so , then a workorderdetail udr is created the querystatus fields in the workorderdetails udr , will be populated with error messages and codes and routed back to *Query\_WO\_State.*

***msg= No Work Order Number present in the Equipment Sales Record.***

c) else , we create a query for Thirsty , and also create a RestCycleUDR with a GET call and the query encoded in the url eg: "/services/data/v43.0/query/?q=(query)"

query :

select SP\_ACN\_\_c, ProjectNumber\_\_c, Service\_Provider\_\_c, CustomerACN\_\_c, ChainACN\_\_c, HQACN\_\_c, JDEDocumentNumber\_\_c, Work\_Order\_Number\_\_r.TECH\_Ext\_workOrder\_\_c, Work\_Order\_Number\_\_r.WorkOrderNumber, ServiceType\_\_c, PartsAmount\_\_c, LaborAmount\_\_c, TravelAmount\_\_c, Miscellaneous\_Amount\_\_c, TotalLaborHours\_\_c, TotalLaborMinutes\_\_c, TotalTravelHours\_\_c, TotalTravelMinutes\_\_c, HoursWorked\_\_c, MinutesWorked\_\_c, TravelHours\_\_c, TravelMinutes\_\_c, Invoice\_Total\_\_c "

FROM Invoice\_\_c WHERE Work\_Order\_Number\_\_r.WorkOrderNumber = (here we be adding the WorkOrderNumber from the xml) limit 20 ;

**Response Flow:**

**1. Prepare\_Query(Analysis):**

1) Once we recieve a RESTCycleUDR ,

2) We first check if the responsecode from thirsty is 200,

If so we then create WorkOrderDetails UDR and the QueryStatus fields will be populated with RESTCycleUDR.Response Fields. and route the UDR Back to *Query\_WO\_State.*

3) We then use a try/catch block for decoding the Response Body , ( why try/catch? -> because if the jsonDecodeUdr fails decoding the ReponseBody , then the work flow will abort instead of that we have catch block ,

*catch block:*

*a. we like usual, create WorkOrderDetails UDR and populate its QueryStatus , with*

***msg = based on the exception that occured, errorcode and reponsecode as -2.*** etc

*b. Route the UDR back to Query\_WO\_State but as a ResponseESR.*

*try block:*

4) if the jsonDecode is successful then we first print the content's of the decode

5) the response should have 1 record only as we are only sending details of only 1 record.

if so , we then populate the contents into a new udr which will be used for populating values in the WorkOrderDetails UDR.

6) else it would mean nothing has been populated, *create WorkOrderDetails UDR and populate its QueryStatus , with* ***msg = No JSON in the Thirsty Response.*** *and route it back to Query\_WO\_State.*

*<----------------------------end of try block--------------------------------------------->*

7) We create a WorkOrderDetails UDR and populate its fields with that of the ResponseBody such as *SP\_ACN, Service\_Provider, TotalTravelHours, etc and its QueryStatus will have* ***msg = Thirsty WO Query Success.*** *and have success type reponse , and then route it back to Query\_WO\_State.*

**2. Query\_WO\_State(Aggregation):**

1) If we recieve the a equimsales Record , then that means we got a response from Thirsty

2) Now we compare the record present in the session and that of the response , we basically compare the important fields such as *OrderNo, OrderType, SoldTo, ShipTo, ICCode, LineType, InstallDate, SalesOrderDate, ExtendedCost and ExtendedPrice.*

3) If the record is matching it is then added to a response list present in the session.

4) If the ***toSend\_ListEquipSalesRecords*** *list is not empty then we update the session.record to the next record( basically the first record in the list , as the one which we got from thirsty has been removed from the list earlier). and then route it back to Prepare\_Query and like before we remove the record from the list.*

5) else , it would mean there are no more records present in ***toSend\_ListEquipSalesRecords ,*** we then update the records(that have been sent over thirsty) from the list that we recived from *WFB\_Handler, with that of the response list(in above step 3) , if the records details (mentioned in step 2) match , else we dont add to record.*

6) Next we route the updated list back to *WFB\_Handler and also close the session.*

**3. WFB\_Handler:**

a) Once we recieve the list from *Query\_WO\_State.* we update the wrapper with the new List which has now been populated with data from Thirsty to the ccu.data and route the ccu back to batch, and *cost\_compare.*

**4. cost\_compare:**

a) We now check if the records (that have LineType=IT & ICCode starts with A-,B-,C-,2-) ExtendedPrice and total work\_order cost(TotalParts+TotalLabor+TotalTravel+TotalMisc).

b) If the dont match then we create a new UDR called failedRecord, this udr will store the msg and certain details of record such as workorderNumber. and then route the udr to *Notifications.*

**5. Notifications:**

a) We send a mail per file including all the workorders that have failed on its cost comparison, we include record and query details.

<---------------------------------------------------------->

***Back to WFL\_EquipSales\_LookUp\_WO***

**5)Validate\_WO\_Details(Analysis):**

a)Now similar to what was done in *cost\_compare* we check whether the ExtenedPrice and total\_word\_order\_detail is equal or not.

b)if they are valid , then

b.1) we updated the Processed Indicator to Y and with msg=*Extended Cost Totals Matched to JDE Record.*

*b*.2)We updated the workorderdetail fields such as TotalLaborHours, etc for the records tht were sent over to Thirsty(IT and A-,B-,C-&2-) and all the records costs(aka TotalLabour,TotalMisc) are added to ONE IT record that have ICCode - 500/505/506/509/510 or thirsty WO present in the EN/ES equipments of the order , sent over to location:

***/usr/sap/interfaces/MZP/EquipmentSales/processing/output***

c) if they are not ,

c.1) We updated the processedIndicator to N and with msg=*Thirsty Extended Cost Totals NOT Matched to JDE Record.*

*c*.2) if the ECS is enabled then we send the list of records to ECS, and if not we send the records individually to location: ***/usr/sap/interfaces/MZP/EquipmentSales/processing/merge\_jde.***

***(the error's are also logged into txn\_detail\_log with category=ES)***

c.3) regardless if process followed the ECS or no, we also send the list OfRecords to reprocessing location: ***/usr/sap/interfaces/MZP/EquipmentSales/reprocessing/input.***

**5)WFL\_EquipSales\_Dispatch\_SD:**

1)Collect the files(a.k.a listOfRecords) from ***/usr/sap/interfaces/MZP/EquipmentSales/processing/output***

**a)Route\_to\_SD(Analysis):**

1) We create a wrapper and wrap the list of records to it and also create an aggregation key for the first record that is IT. and assign the wrapper to ConsumeCycleUDR data and then route the ConsumeCycleUdr to ***WFL\_EquipSales\_Dispatch\_SD\_RT.***

**5)WFL\_EquipSales\_Dispatch\_SD\_RT:**

**a) Generate\_Idoc(Analysis):**

**Request\_Flow:**

1. We create idoc segements and map the data present in list of records which we recieved from batch, intially we use the first IT record to create Idoc segements like eg: ZE1EDKO1,E1EDK03\_027(Uses the SalesOrderDate) , E1EDK03\_023(Uses the InstallDate)

Segments creation:

IDOC\_CONTROL\_REC\_40 params:

E1EDK01 - > Segment is created using the first IT Record found in the Order, if the IT Record is for Bill or Deduct , and be assigned different constants for it which will be used in Segement

E1EDK14\_008 -> using the first IT Record , we assign the Company Code , if no company code is found we assign a default constant .

E1EDK14\_007 -> use the constant E1EDK14\_ORGID\_DIST\_CHAN=10, to create the segment

E1EDK14\_006 -> use the constant E1EDK14\_ORGID\_DIVISION=99, to create the segment

E1EDK14\_012 -> use the E1EDK14\_ORGID\_ORDER\_TYPE=ZESO,to create the segment

E1EDK03\_027 -> create a segment by using the first IT Record’s SalesOrderDate

E1EDK03\_023 -> create a segment by using the first IT Record’s InstallDate

E1EDKA1\_ShipTo -> segment created using the first IT Records ShipTo

E1EDKA1\_SoldTo-> segment created using the first IT Records SoldTo & OrderNo

E1EDKA1\_ServiceProvider -> We do validation for the SP\_ACN of the first It record ( if the SP\_ACN is not null or empty and having 999 (999 is for dummy record)in it, then only we make the segment using the SP\_ACN

E1EDK02 -> Segment created using the CustomerPO of First IT record, if the CustomerPO is 0 then CustomerPO used for making the segment will have empty value.

E1EDK02\_2 -> segment created using the first IT records ProjectNumber

We now traverse the entire list of records for the OrderNo:

For Non-IT records:

E1EDP01\_ES->record’s ShippedQuantity & record’s UnitofMeasure

ZE1EDP01\_ES -> record’s EquipmentType, EquipmentSubType, SerialNumber

E1EDP05\_ZPRC -> record’s ExtendedPrice if greater than 0 ,else we use ExtendedCost

E1EDP19\_ES -> record’s ICCode

E1EDPT1\_ES -> using const “001”,”EN”

E1EDPT2\_ES -> record’s OrderLineNumber

For IT Records with Processed\_Indicator as Success

E1EDP01\_IT -> record’s ShippedQuantity and UnitofMeasure

ZE1EDP01\_IT -> records EquipmentType,EquipmentSubType & SerialNumber

E1EDP19\_IT -> using constant’s E1EDP19\_QUALF\_IT\_ICCODE & E1EDP19\_IDTNR\_IT

E1EDPT1\_IT -> using const “001”,”EN”

E1EDPT2\_IT -> record’s OrderLineNumber

These below 4 segments will be created if the TotalLabor >0

E1EDP01\_ZLBR -> we use minutesInLabor, we do using the fields we got from Thirsty namly TotalLaborHours & TotalLaborMinutes and following those caluclations assign it minutesInLabor

ZE1EDP01\_ZLBR -> records EquipmentType, EquipmentSubType & SerialNumber

E1EDP05\_ZLBR -> using costByMinLabor , this is another constant created by dividing Thirsty’s TotalLabor and minutesInLabor

E1EDP19\_ZLBR -> using constant’s E1EDP19\_QUALF\_IT\_ICCODE & E1EDP19\_IDTNR\_IT

Next 4 segments will be created if the TotalParts>0:

E1EDP01\_ZPRT-> record’s ShippedQuantity,UnitofMeasure

ZE1EDP01\_ZPRT -> records EquipmentType,EquipmentSubType &SerialNumber

E1EDP05\_ZPRT -> using the records TotalParts( collected from Thirsty)

E1EDP19\_ZPRT -> using constants E1EDP19\_QUALF\_IT\_ICCODE, E1EDP19\_IDTNR\_ZPRT

Next segments will be created if TotalMisc >0:

E1EDP01\_ZMSC -> record’s ShippedQuantity,UnitofMeasure

ZE1EDP01\_ZMSC -> records EquipmentType,EquipmentSubType &SerialNumber

E1EDP05\_ZMSC -> using the records TotalMisc(collected from Thirsty)

E1EDP19\_ZMSC -> using constants E1EDP19\_QUALF\_IT\_ICCODE, E1EDP19\_IDTNR\_ZPRT

b) We then check if the SAP RTC is enabled , and depending of the size of *idoc\_inbound\_asynchronous* we insert a log into ***TXN\_DETAIL\_LOG*** table , and then route to SAP RTC Agent.

c) if not enabled , we create a sd\_response with a flag( isSuccessful) as false and route back to batch as wrapped in ConsumeCycleUdr.Action

**Response\_Flow:**

a) If agent recives a RfcErrorUDR , simialar to what we did in Request\_Flow in steps b and c , we log the error into ***TXN\_DETAIL\_LOG*** table and also response error msg's are mapped to the *sd\_response.* and send back to batch.

b) else , the *sd\_response* flag as true and also log the success to ***TXN\_DETAIL\_LOG*** and send back to batch.

<-------------------------------------------------------------------------------------------------->

***Back to batch* WFL\_EquipSales\_Dispatch\_SD**

**b)Route\_SD\_Response(Analysis):**

1)If we recieve the ConsumeCycleUDR and if the *sd\_response* flag is true then we update the ProcessedIndicator with *msg = Successfully sent JDE Equipment Sales Order as IDOC to SD.*and route(list Of Records) to location: ***/usr/sap/interfaces/MZP/EquipmentSales/distribution/idoc/success***

2) if the flag is false, we updated the ProcessedIndicator with *msg=Failed to send IDOC to SD.* then the list Of Records is routed to reprocessing location:

***/usr/sap/interfaces/MZP/EquipmentSales/reprocessing/output***

3) we then convert the xml records back to csv , then stored in location

***/usr/sap/interfaces/MZP/EquipmentSales/processing/merge\_jde***

**6)WFL\_EquipSales\_Forward\_JDE:**

a) Collect the files from ***/usr/sap/interfaces/MZP/EquipmentSales/processing/merge\_jde***

**b) Wrap\_ASCII:**

1) We basically create header for csv using the filename of the xml and also the trailer(in drain block) using filename and record\_counter(count of records) and route them to JDE\_ARCHIVE(***/JDE/BRIM/ES/ARCHIVE****)*

<-------------------------------------------------------------END-------------------------------------------------------------------->