USE CASE 2 – BATCH HIRE INTEGRATION FROM PEOPLESOFT TO WORKDAY FINANCIALS

Develop a batch application that will take hires/re-hires from PeopleSoft per a given date/time and then call the necessary Workday Web Services to get worker data, hire/rehire and finally, add the worker account. Due to the nature of our business, there are a number of distinct requirements which will be outlined in the following sections.

Overall Processing/InputParms
- Job needs to allow for the possibility of processing hundreds and possibly thousands of hires on any given run. Due to this, it should be a “batch” job so that threading or other optimizing techniques can be utilized to process a large population in as short a time as possible.
- Job needs to be able to receive optional input parms, From_Date_Time and Thru_Date_Time that will passed in from initiating scheduling system to allow process to re-run for a given date/time period (explained below how these are utilized in dynamic PeopleSoft SQL).
- Job also needs to accept an optional input parm string of CSV Emplids (will be explained below).
- Job needs to have multiple end-points with two examples being:
  - One endpoint needs to be a Ping flow to see if the job is active. It should return a JSON payload indicating the service is alive;
  - One endpoint needs to be the one to initiate the job;

Input Data
Hire/re-hire data obtained from Oracle PeopleSoft via a complex view that is accessed via dynamic SQL in the batch program. Sometimes, due to the volume of transactions, it may take upwards of 45 minutes for a response from the DB call to PeopleSoft so the program needs to able to handle this and not time out. This is due to times when we process a large number of hires/re-hires like at the beginning of the school year. This is also why the process needs to be handled as a batch/high-volume process and not as an API.

The data returned from the DB call needs to be able to be easily accessible and stored in variables utilized during the life of the process. The data returned should be able to be processed as records such that if we receive 100 hire/rehires from PeopleSoft, the job will allow us to process them one-by-one although behind the scenes threading will be utilized which may chunk data. Examples of data include:

- Emplid
- First Name
- Last Name
- DOB
- Department

The SQL for the PeopleSoft call to obtain the population to process must be dynamic so that the SQL string can be dynamically built allowing for various input parms in order to select the correct population. These parms are optional and include:

- One to many Emplids sent into the program as a CSV string (e.g. 1289888880, 1234578932, 2321234542). If these values are supplied, they need to be dynamically included in the SQL like:
WHERE EMPLID IN " + Input Emplids + "

- From Date Time and Thru Date Time (format: yyyy-mm-dd hh:mm:ss). If these values are supplied, they will be included in the SQL dynamically in similar fashion to the example below.

  
  
  \[
  \begin{align*}
  ((\text{EFFDT BETWEEN TO\_DATE}(" + "'" + \text{input\_fromDate} + "'" + ", \ 'YYYY-MM-DD')") & \text{ AND (AUDIT\_STAMP BETWEEN TO\_DATE}(" + "'" + \text{input\_fromDtTmAudit} + "'" + ", \ 'YYYY-MM-DD HH24:MI:SS')") + "TO\_DATE"(" + "'" + \text{input\_thruDate} + "'" + ", \ 'YYYY-MM-DD')")
  \end{align*}
  \]

- If From and Thru Date Time are not supplied, the job must access an “Object Store” or persistent storage to obtain watermark date/times in order to build the SQL to get the hire data per last date/time the job ran. The program will build these values into the SELECT as seen above.

Values to obtain from Object Store need to be identified per a unique key and should include From Date Time, Thru Date Time and Audit Date Time.

**Note:** These watermark date/time values need to be updated at the end of the job so that the next time the job runs it will pick up new data and not re-process old data. Refer to ‘End of Job Processing’.

The data returned from the PeopleSoft DB SQL call should first be formatted and written out to a Logger as a JSON payload for use in Splunk reporting.

**Note:** If no rows are returned from the SQL call, the job should write out an appropriate message to the log file and stop processing.

**Batch Processing**

Once hire/rehire data is obtained from PeopleSoft, batch processing should begin for the returned population. These steps will be done for each person returned from the SQL. Batch processing includes:

- Build SOAP request for Workday Get_Workers web service to see if employee already exists in Workday. This request needs to include the EMPLID obtained from PeopleSoft as the ID in Workday Request References as a WorkerObject.
- Issue call to Workday for Get_Workers.
- Process Get_Workers Response from Workday and parse SOAP response to extract to variables key fields such as Formatted_Name, Hire_Date, Original_Hire_Date, Active, Rehire, Terminated. These fields and the SOAP response also be written out to a Logger as a JSON payload for use in Splunk reporting. See Appendix B for an example.
- Act upon data from Workday Get_Workers and compare with PeopleSoft data to make final determination as to how to handle the person. See Hire/Rehire Evaluate which depicts the evaluation. Basically, we need to determine if the person is a new hire, rehire, or a new hire in PeopleSoft that needs to be treated as a re-hire in workday.
Hire/Rehire Processing

- Call respective sub-flow to either Hire employee or Rehire. Processing is slightly different as the WD SOAP Request requires different data depending on whether you are hiring or rehiring.
- Build SOAP request for Workday Hire_Employees Web Service. This will involve being able to interrogate data previously obtained and to build the request for this Web Service.
- Process response from Hire_Employees Web Service call. Key data elements need to be extracted and transformed from XML into a JSON payload and written out to log file for reporting/error tracking. See Hire_Employees Response in Appendices.

Add Workday Account

- If the person is hired successfully, we need to call one final Workday Web Service to create the Workday account. One exceptional item to note for this call follows:
  - In hiring an employee via WS Hire_Employees, there is a slight delay in when the Web Service is called and the person is actually available in Workday. Due to this, we needed to build in a “pause” in processing before actually calling the Add_Workday_Account Web Service to create the account. If not, then the WS call will fail due to the employee not being found. Just as a note, to accomplish this, we utilized a Groovy script to sleep for 30 seconds.
- Build Workday Web Service call for Add_Workday_Account and call to add WD account.
- Process response from Add_Workday_Account as done for other WS calls where data from response is formatted into JSON payload and written out to log file. If response is successful, continue on to End of Job Processing; if not and an error or exception is encountered, it needs to be logged and an exception count incremented.

End of Job Processing

End of job processing happens after all employees/rows have been processed. It is not processed for every row from the input step, but only at the end of job.
- Update Object Store/Persistent Storage with latest From and Thru Date Times and other relevant fields to watermark for next processing run.
- Write out totals in XML payload including processing time, # of records processed and any custom counts maintained by the program. See ‘End of Job Totals’.

Additional Items
This section will highlight a few additional items relevant to the processing requirements.

Error Processing
The integration needs to incorporate Exception Handling so that exceptions in the process can be caught, evaluated and properly handled as explained below. Specific exceptions to catch include any from Workday Web Service calls like Get_Workers or Hire_Employees, but also we need to be able to catch any errors that may happen internally such as an HTTP Timeout, Authentication denied on endpoint, etc., org.apache.cxf.binding.soap.SoapFault, etc.

When processing exceptions, it is desired that the exception can be caught so we can have the option to execute a separate sub-flow to do additional processing, and also write out the exception details with other key fields/variables via a logger.
We include in the Appendices an example for an exception returned from the Get_Workers WS indicating that the ID sent in is invalid. We “catch” this situation/SOAP Fault and report it in a logger with additional information that is used in reporting and troubleshooting. Refer to Exception Logger.

**Count Details**
The job needs to be able to maintain counts relevant to the process that will then be reported at the end of the job and written out to a log file via a logger. Count examples include:

- Records selected from PeopleSoft DB SELECT
- Number of Hires sent to Workday
- Number of Rehires sent to Workday
- Number of successful Hires processed by Workday (Determined by SOAP response from Workday Hire_Employees call)
- Number of successful Re-Hires processed by Workday (Determined by SOAP response from Workday Hire_Employees call)
- Number of exceptions encountered

**Logger**
As noted in various places above, we need to be able to write out detailed logger statements to a log file for the job that can include variables, payload, exception details etc. Log statements should be able to include a unique ID per each batch job so that they log statements can be matched up with the particular run.
USE CASE 2 - HIRE APPENDICES

Use Case 2 Appendix A – Hire/Rehire Example Evaluate

- Multiple conditions checked via evaluate statement. Data checked includes variables set based on prior processing.

1. **Not in Workday - HIRE**
   - Process Hire

2. **In Workday as Tenured - REHIRE**
   - Process Rehire

3. **SOAP Exception caught from Workday Call**
   - Log Exception

4. **Check for Unique PS-HIR Mismatch**
   - Change HIRE to REHIRE
   - Log Exception

Bulk Batch Processing

SOAP WS
Call to WD to get XML for Person

Evaluate/Choice
Use Case 2 Appendix B – Get_Workers JSON Response


"Get_Workers_Response": [

{}],

"Employee": [

{}],

"Assisted": "",

"Name": "Felix",

"Hire_Date": "2019-08-12",

"Original_Hire_Date": "2019-08-12",

"Active_Status": "2019-08-12",

"Active": "1",

"Rehire": "0",

"Terminated": "0",

"FromDate": "2019-08-12 21:15:43",

"ToDate": "2019-08-12 06:00:32"

]
Use Case 2 Appendix C – Exception Logger

InstanceID=04872220-bd02-799-a93-d334e77a  host=muleprod-app03  source=logs/mule-workday-prod/logs/au-techn/hire-term-inbound.log  sourceType=mule


47 FirstName=Daniel  MiddleName=M  LastName=Smith

AuroraError=Exception encountered: hire. HireEmployeeSoapFault: Validation error occurred. Invalid ID value. '999999999' is not a valid ID value for type = 'Applicant_ID' soapException=SoapFault: Validation error occurred. Invalid ID value. '999999999' is not a valid ID value for type = 'Applicant_ID' exception=SoapFault: Validation error occurred. Invalid ID value. '999999999' is not a valid ID value for type = 'Applicant_ID'

instanceID=04872220-bd02-799-a93-d334e77a  host=muleprod-app03  source=logs/mule-workday-prod/logs/au-techn/hire-term-inbound.log  sourceType=mule
Use Case 2 Appendix D – Hire_Employees Response

2019-08-11 21:51:44,118 [BatchJob:au-pghr2working=hire-termin=hrbound=hr=work=manager=20] INFO edu.au.uto.pghr2working.httpmessage="Weekday Response: Hire_Employees_Reponse" httpStatus=200 EmpId=828 FirstName=Camila MiddleName=Marie LastName=Ritter AspiriteID=828 payload={
  "Hire_Employee_Event_Response": {
    "Event_ID": "695",
    "UserID": "",
    "EmpId": "",
    "EmpID": "",
    "Emp_ID": 0,
    "Name": null,
    "FirstName": null,
    "MiddleName": null,
    "LastName": null,
    "Effective": "2019-08-12T00:00:00",
    "Action": "HIRE",
    "FromDate": "2019-08-11 21:12:38",
    "ToDate": "2019-08-12 21:15:43",
    "ID": [],
    "WDP": "48a",
    "EmpID": "",
    "Applicant_ID": {
      "ID": [],
      "Applicant_ID": "T"}
  }
}
Use Case 2 Appendix E – End of Job Totals

```
<object>
  <script>
    if (window['usecase']) {
      console.log(usecase);
    }
  </script>
</object>
```

Rev 07-02-18