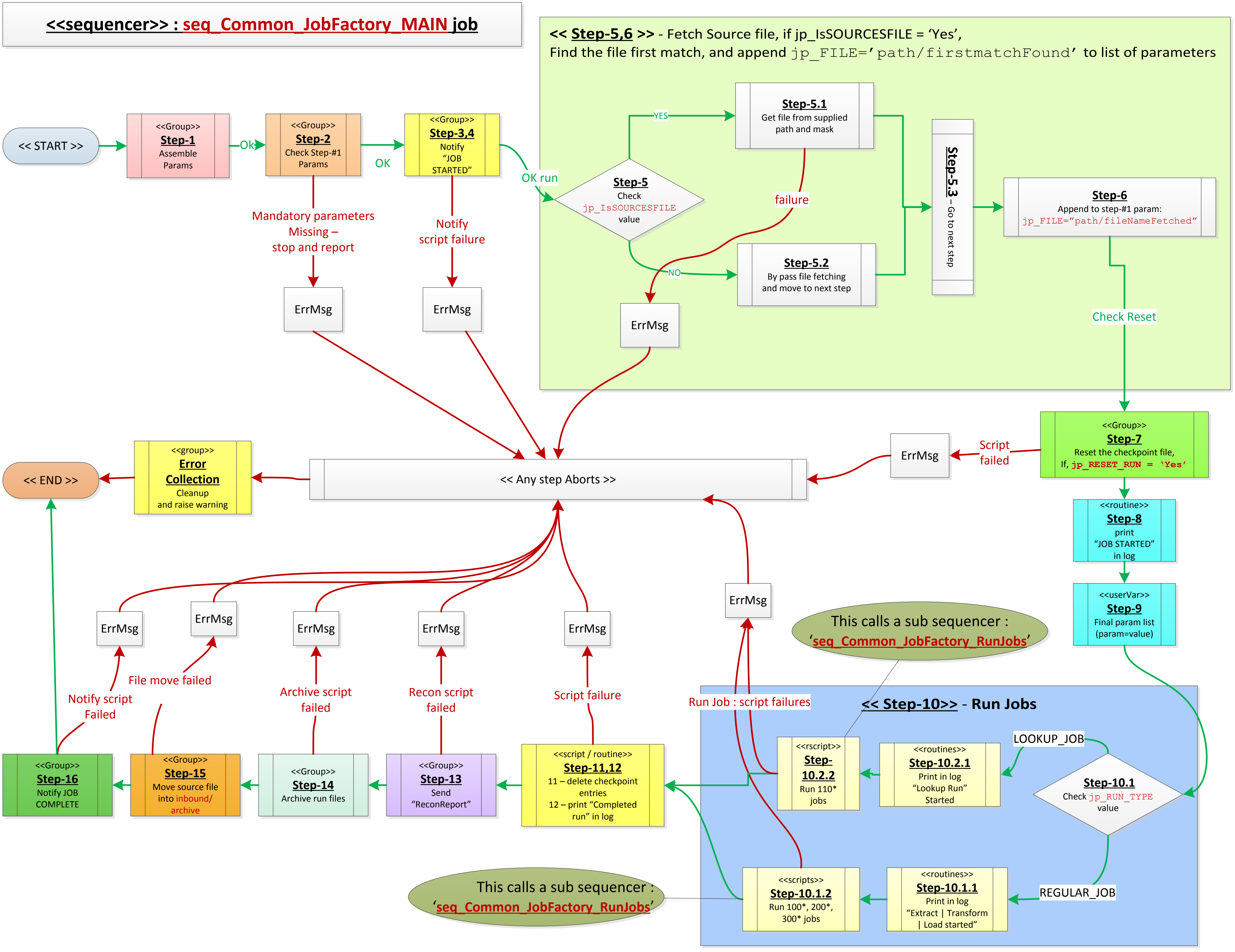
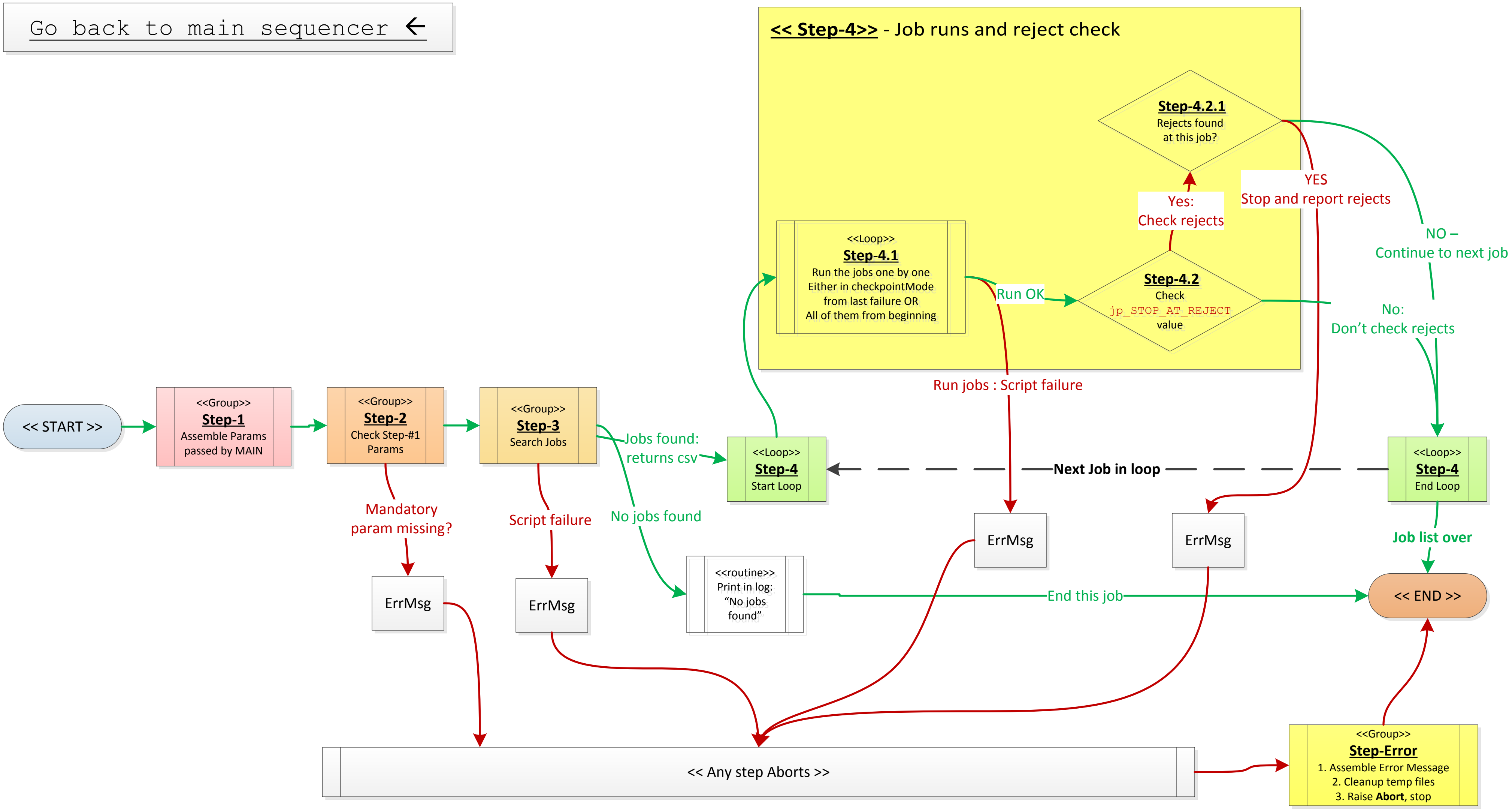
<<sequencer>> : seq_Common_JobFactory_MAIN job


<< Step-5,6 >> - Fetch Source file, if jp_IsSOURCESFILE = 'Yes',
Find the file first match, and append jp_FILE='path/firstmatchFound' to list of parameters
<< START >>
<<Group>> Step-1 Assemble Params
Ok
<<Group>> Step-2 Check Step-#1 Params
OK
<<Group>> Step-3,4 Notify "JOB STARTED"
OK run
Mandatory parameters Missing – stop and report
Notify script failure
ErrMsg
ErrMsg
Step-5 Check jp_IsSOURCESFILE value
YES
NO
Step-5.1 Get file from supplied path and mask
failure
Step-5.2 By pass file fetching and move to next step
ErrMsg
Step-5.3 – Go to next step
Step-6 Append to step-#1 param: jp_FILE="path/fileNameFetched"
Check Reset
<<Group>> Step-7 Reset the checkpoint file, If, jp_RESET_RUN = 'Yes'
Script failed
ErrMsg
<< Any step Aborts >>
<<group>> Error Collection Cleanup and raise warning
<< END >>
<<routine>> Step-8 print "JOB STARTED" in log
<<userVar>> Step-9 Final param list (param=value)
ErrMsg
This calls a sub sequencer : 'seq_Common_JobFactory_RunJobs'
ErrMsg
ErrMsg
ErrMsg
ErrMsg
ErrMsg
File move failed
Notify script Failed
Archive script failed
Recon script failed
Script failure
Run Job : script failures
<< Step-10>> - Run Jobs
LOOKUP_JOB
<<rscript>> Step-10.2.2 Run 110* jobs
<<routines>> Step-10.2.1 Print in log "Lookup Run" Started
Step-10.1 Check jp_RUN_TYPE value
REGULAR_JOB
<<scripts>> Step-10.1.2 Run 100*, 200*, 300* jobs
<<routines>> Step-10.1.1 Print in log "Extract | Transform | Load started"
This calls a sub sequencer : 'seq_Common_JobFactory_RunJobs'
<<Group>> Step-16 Notify JOB COMPLETE
<<Group>> Step-15 Move source file into inbound/ archive
<<Group>> Step-14 Archive run files
<<Group>> Step-13 Send "ReconReport"
<<script / routine>> Step-11,12 11 – delete checkpoint entries 12 – print "Completed run" in log

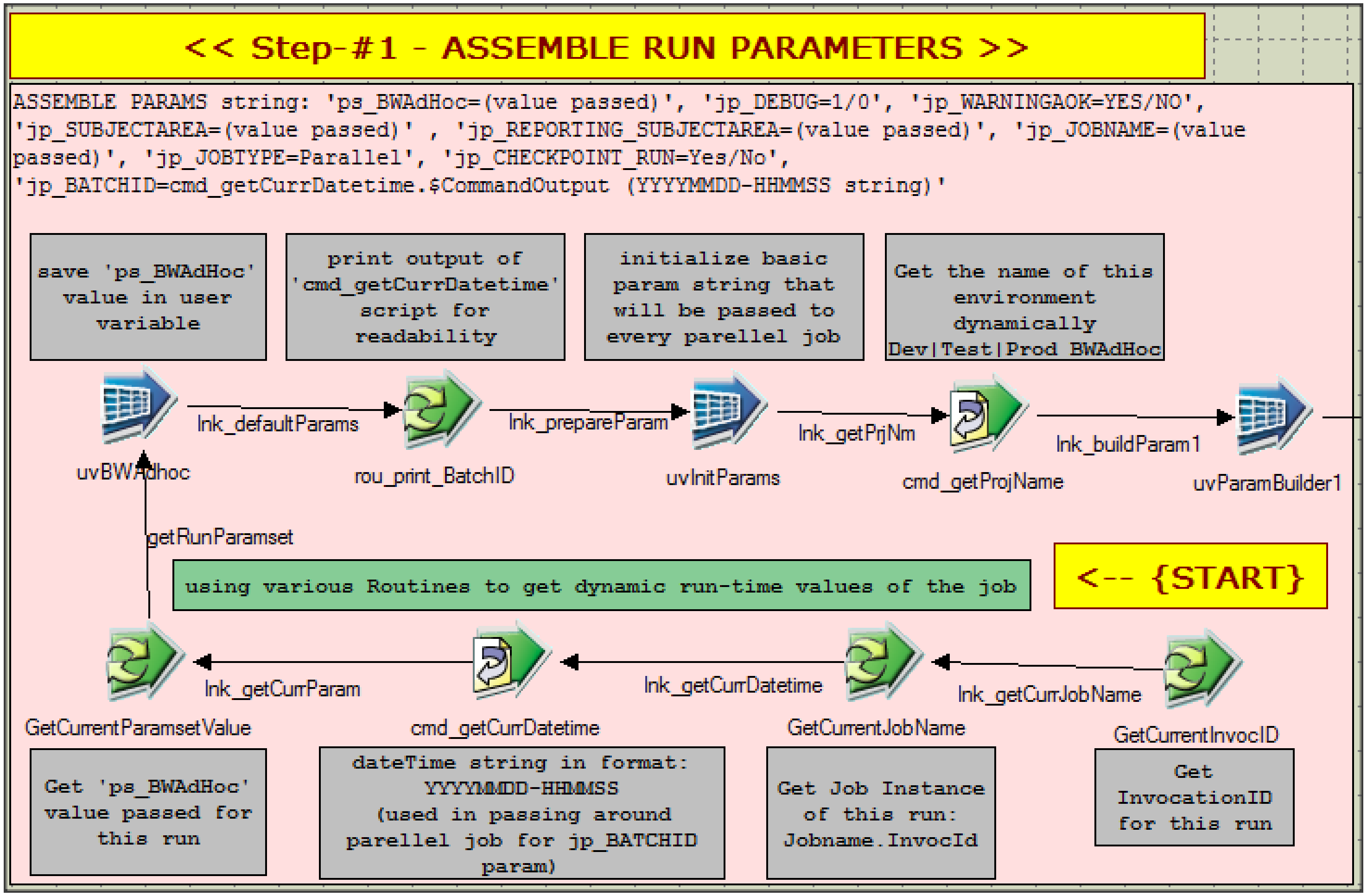
<<sequencer>> : seq_Common_JobFactory_RunJobs.[InvocationID] job

Go back to main sequencer ←



Go back to main sequencer ←

1) Run various routines to get dynamic run time values
1) Assemble run params in below mentioned format:
"paramname1 = value1, paramname2 = value2, etc"



<< Step-#1 - ASSEMBLE RUN PARAMETERS >>

ASSEMBLE PARAMS string: 'ps_BWAdHoc=(value passed)', 'jp_DEBUG=1/0', 'jp_WARNINGAOK=YES/NO', 'jp_SUBJECTAREA=(value passed)' , 'jp_REPORTING_SUBJECTAREA=(value passed)', 'jp_JOBNAME=(value passed)', 'jp_JOBTYPE=Parallel', 'jp_CHECKPOINT_RUN=Yes/No', 'jp_BATCHID=cmd_getCurrDatetime.$CommandOutput (YYYYMMDD-HHMMSS string)'

save 'ps_BWAdHoc' value in user variable

print output of 'cmd_getCurrDatetime' script for readability

initialize basic param string that will be passed to every parellel job

Get the name of this environment dynamically Dev|Test|Prod BWAdHoc

using various Routines to get dynamic run-time values of the job

<-- {START}

Get 'ps_BWAdHoc' value passed for this run

dateTime string in format: YYYYMMDD-HHMMSS (used in passing around parellel job for jp_BATCHID param)

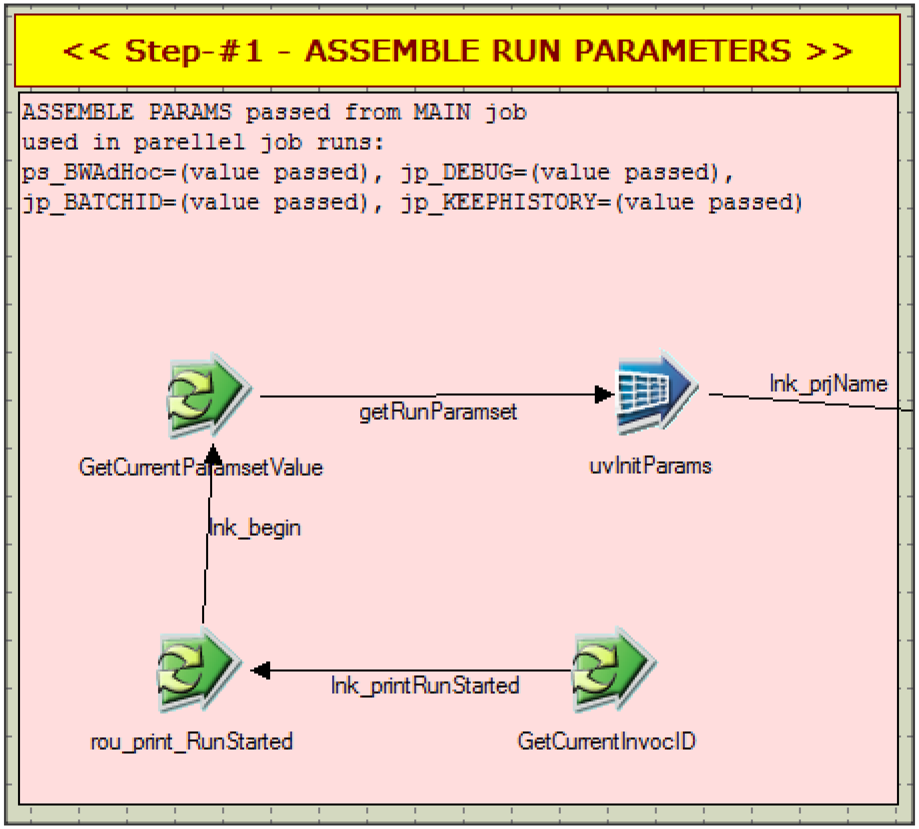
Get Job Instance of this run: Jobname.InvocId

Get InvocationID for this run

Go back to RunJobs sequencer ←

1) Run various routines to get dynamic run time values

1) Assemble run params passed from the main sequencer:
   "paramname1 = value1, paramname2 = value2, etc"

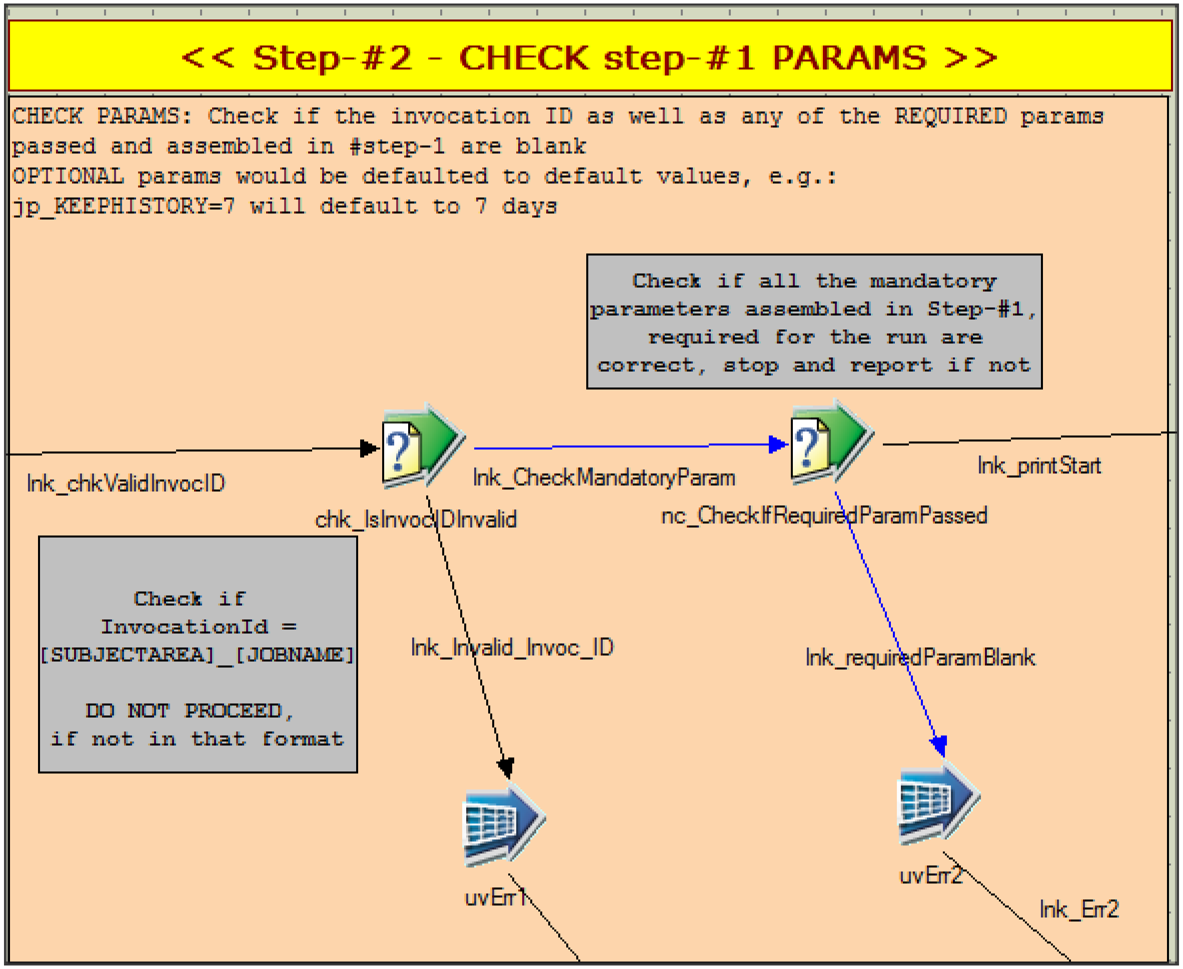


<< Step-#1 - ASSEMBLE RUN PARAMETERS >>

ASSEMBLE PARAMS passed from MAIN job
used in parellel job runs:
ps_BWAdHoc=(value passed), jp_DEBUG=(value passed),
jp_BATCHID=(value passed), jp_KEEPHISTORY=(value passed)

Go back to main sequencer ←

1) Check if "InvocationID" = "[jp_SUBJECTAREA]_[jp_JOBNAME]",
   -> stop and report bad Invocation if not
2) Check if all the required parameters are passed with correct value
   -> stop and report bad parameters if not



<< Step-#2 - CHECK step-#1 PARAMS >>

CHECK PARAMS: Check if the invocation ID as well as any of the REQUIRED params passed and assembled in #step-1 are blank
OPTIONAL params would be defaulted to default values, e.g.:
jp_KEEPHISTORY=7 will default to 7 days

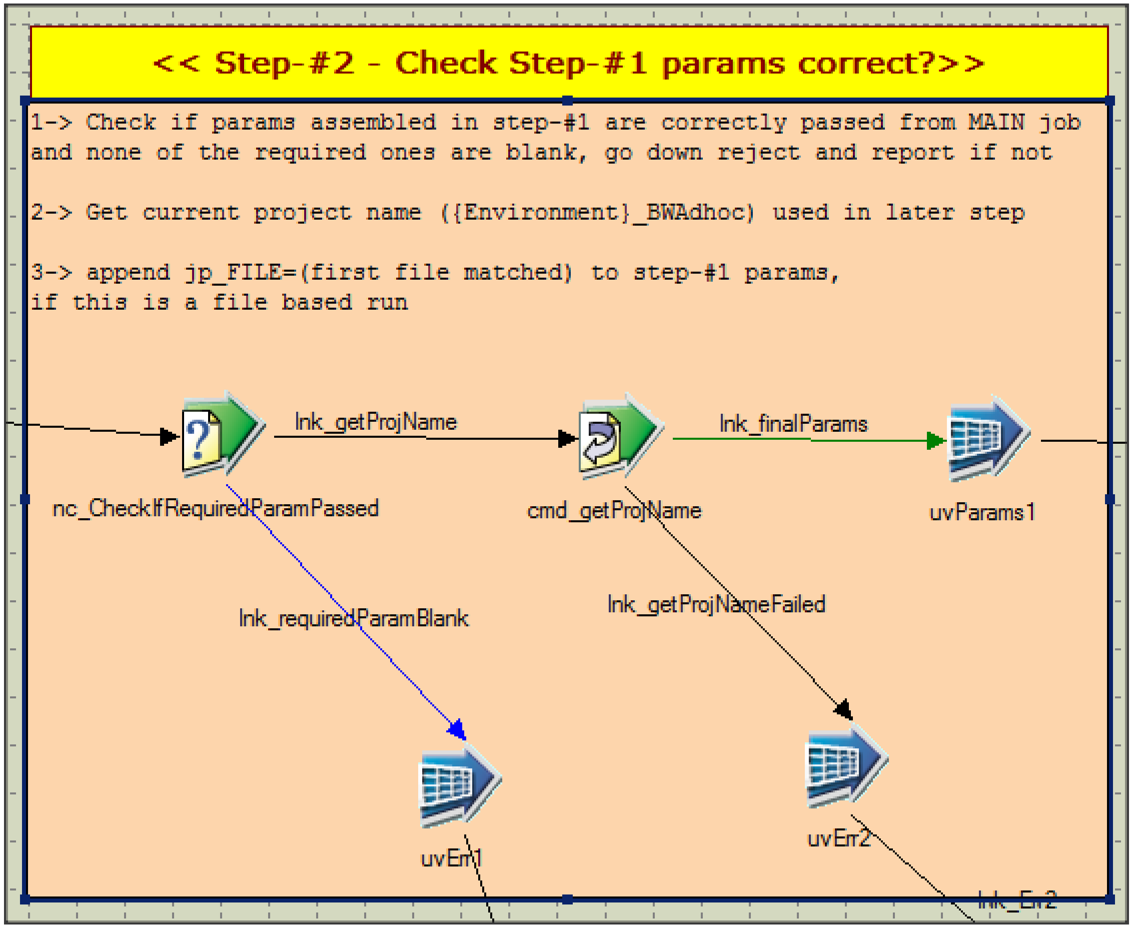
Check if all the mandatory parameters assembled in Step-#1, required for the run are correct, stop and report if not

Check if InvocationId = [SUBJECTAREA]_[JOBNAME]

DO NOT PROCEED, if not in that format

Go back to RunJobs sequencer ←

1) Check if the required params were passed correctly
   If NO, report the missing param and stop



<< Step-#2 - Check Step-#1 params correct?>>

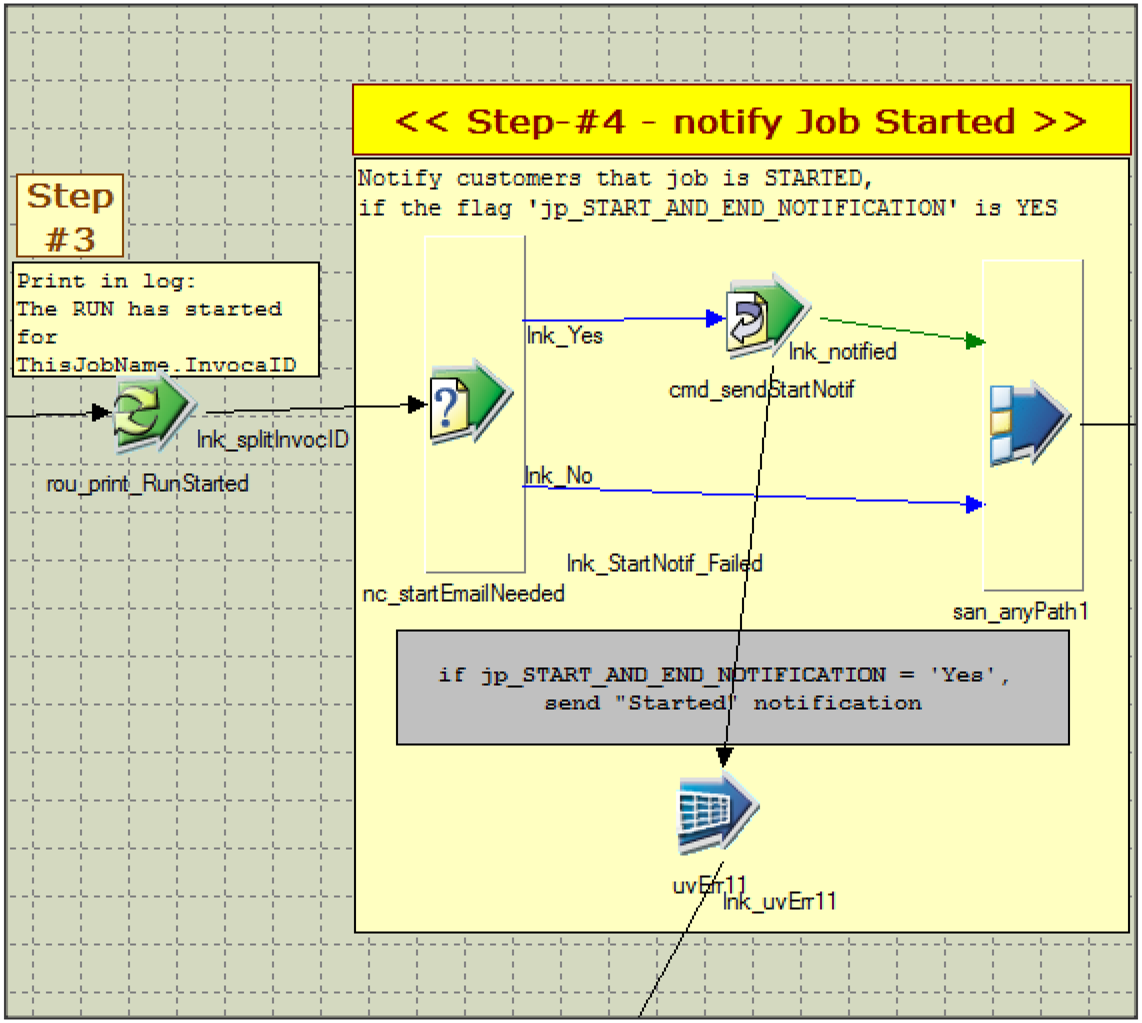
1-> Check if params assembled in step-#1 are correctly passed from MAIN job and none of the required ones are blank, go down reject and report if not

2-> Get current project name ({Environment}_BWAdhoc) used in later step

3-> append jp_FILE=(first file matched) to step-#1 params,
if this is a file based run

Go back to main sequencer ←

1) Print in job log "Started Run for : {seq_Common_JobFactory_MAIN.InvocationID}"
2) if jp_START_AND_END_NOTIFICATION = "Yes",
   then send informational email to users stating: "{InvocationID} Run has STARTED"



Step #3

Print in log:
The RUN has started
for
ThisJobName.InvocaID

<< Step-#4 - notify Job Started >>

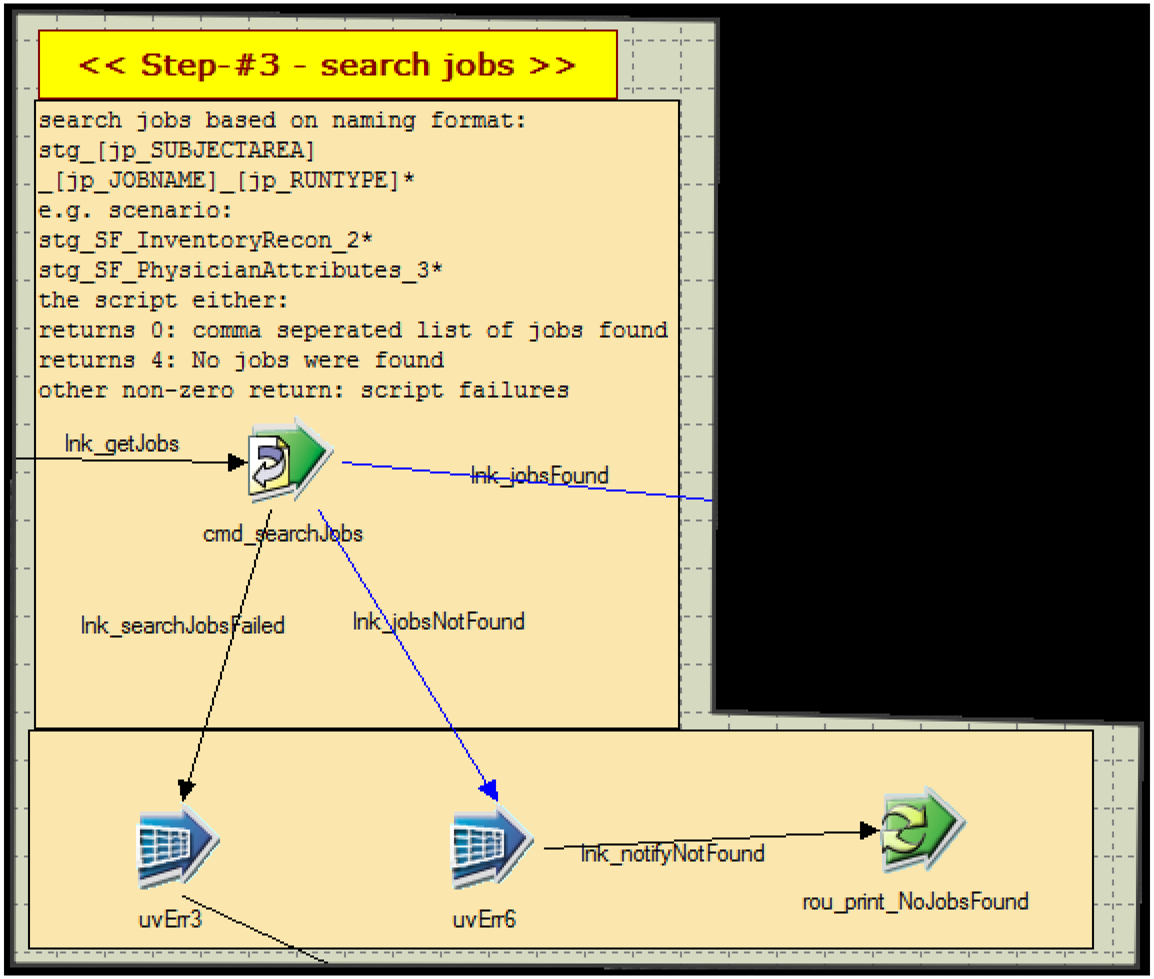
Notify customers that job is STARTED,
if the flag 'jp_START_AND_END_NOTIFICATION' is YES

if jp_START_AND_END_NOTIFICATION = 'Yes',
send "Started" notification

Go back to RunJobs sequencer ←

```
1) Search job based on following predicate for:
1A) Extract       : stg_[jp_SUBJECTAREA]_[jp_JOBNAME]_1*
1B) Transform     : stg_[jp_SUBJECTAREA]_[jp_JOBNAME]_2*
1C) Load          : stg_[jp_SUBJECTAREA]_[jp_JOBNAME]_3*
1D) LookupExtract : stg_[jp_SUBJECTAREA]_Lookup_*_110*
```

If **found**, returns a comma separate list of job names, that is used in loop
if **not found**, prints in log "NO JOBS FOUND" and ends

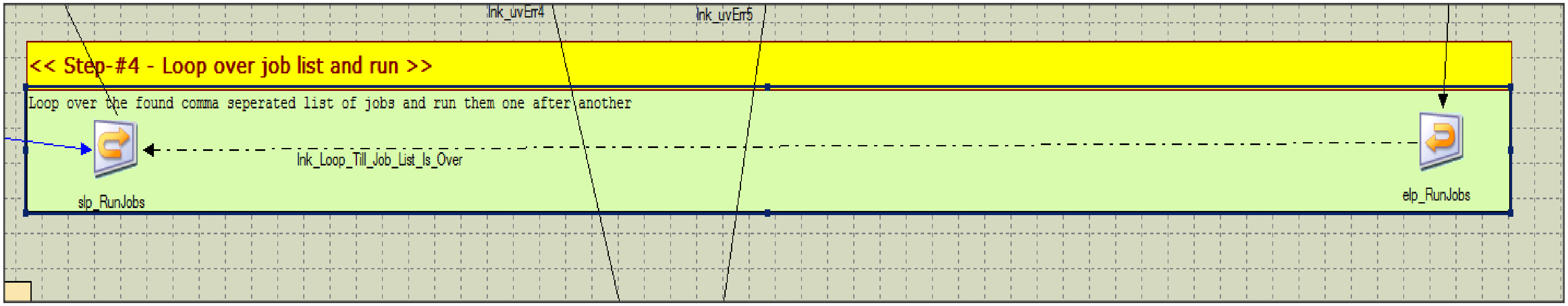


<< Step-#3 - search jobs >>

```
search jobs based on naming format:
stg_[jp_SUBJECTAREA]
_[jp_JOBNAME]_[jp_RUNTYPE]*
e.g. scenario:
stg_SF_InventoryRecon_2*
stg_SF_PhysicianAttributes_3*
the script either:
returns 0: comma seperated list of jobs found
returns 4: No jobs were found
other non-zero return: script failures
```

Go back to RunJobs sequencer ←

1) For each jobs returned in comma seperated list in step-#3:
1A) Run them one after another, either in checkpoint mode, or all of them from beginning without checkpoint

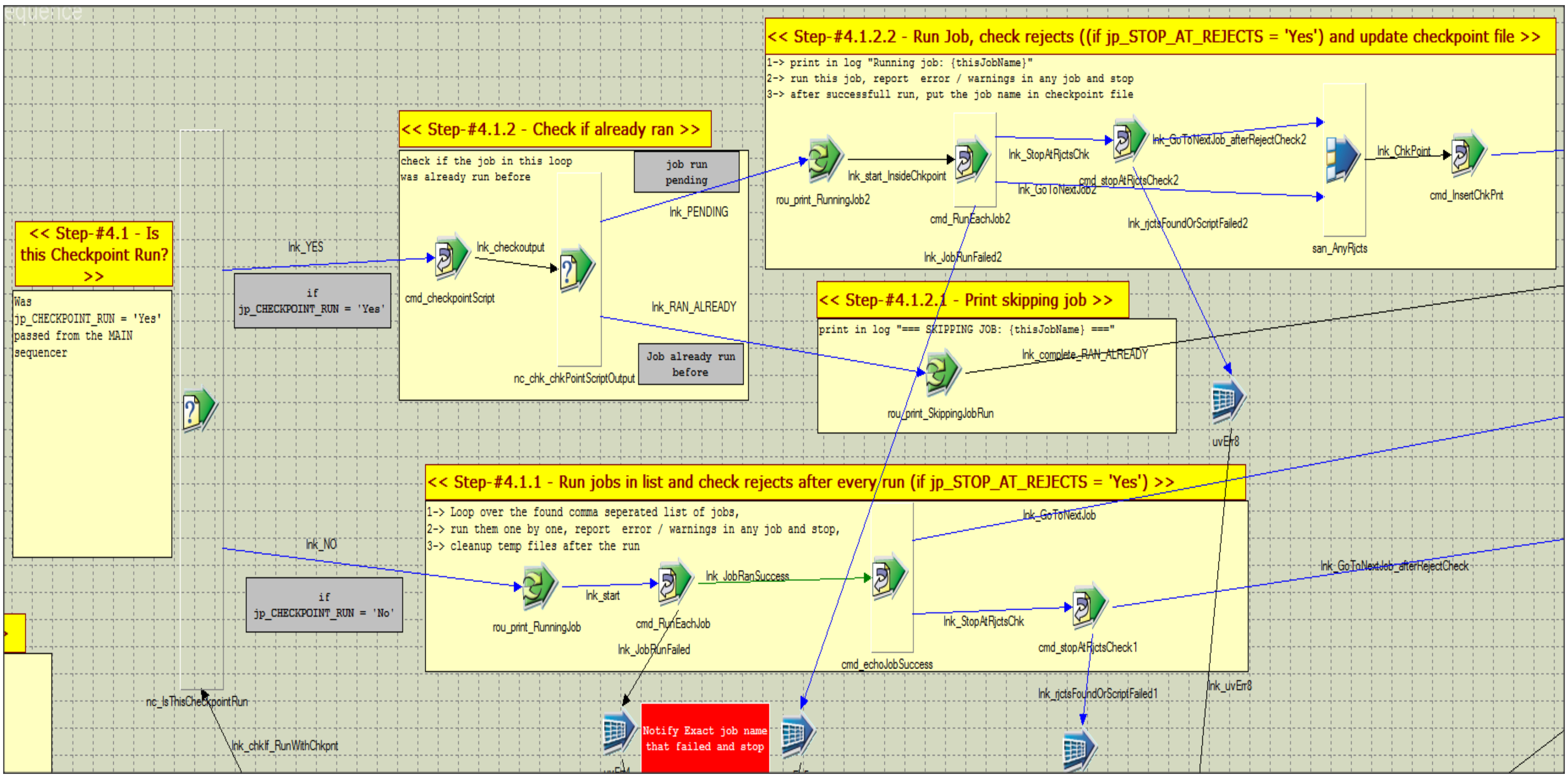


<< Step-#4 - Loop over job list and run >>

Loop over the found comma seperated list of jobs and run them one after another

Go back to RunJobs sequencer ←

```
4.1) Check value of jp_CHECKPOINT_RUN that was passed from MAIN sequencer
      4.1.1) If jp_CHECKPOINT_RUN='No',
            4.1.1.1) Run the current job in csv job list
            4.1.1.2) Check jp_STOP_AT_REJECTS value
                              if 'Yes' => check job's reject files, if rejects > 0, report and stop
                              if 'No' => don't check any rejects and move to next job
      4.1.2) If jp_CHECKPOINT_RUN='Yes', check if the job in loop is found in checkpoint file through script
            4.1.2.1) if script output: RAN_ALREADY, print in log "=== SKIPPING JOB – {thisJobName} ==="
            4.1.2.2) if script output: PENDING, then do the following =>
                  4.1.2.2.1) Run the current job  in csv job list
                  4.1.2.2.2) check jp_STOP_AT_REJECTS value
                                    if 'Yes' => check job's reject files, if rejects > 0, report and stop
                                    if 'No' => don't check any rejects and move to next step
                  4.1.2.2.3) Put job name in checkpointfile:
                                 "processing/[jp_SUBJECTAREA]_[jp_JOBNAME].chk" file
```



<< Step-#4.1 - Is this Checkpoint Run? >>

Was jp_CHECKPOINT_RUN = 'Yes' passed from the MAIN sequencer

<< Step-#4.1.2 - Check if already ran >>

check if the job in this loop was already run before

<< Step-#4.1.2.2 - Run Job, check rejects ((if jp_STOP_AT_REJECTS = 'Yes') and update checkpoint file >>

1-> print in log "Running job: {thisJobName}"
2-> run this job, report error / warnings in any job and stop
3-> after successfull run, put the job name in checkpoint file

<< Step-#4.1.2.1 - Print skipping job >>

print in log "=== SKIPPING JOB: {thisJobName} ==="

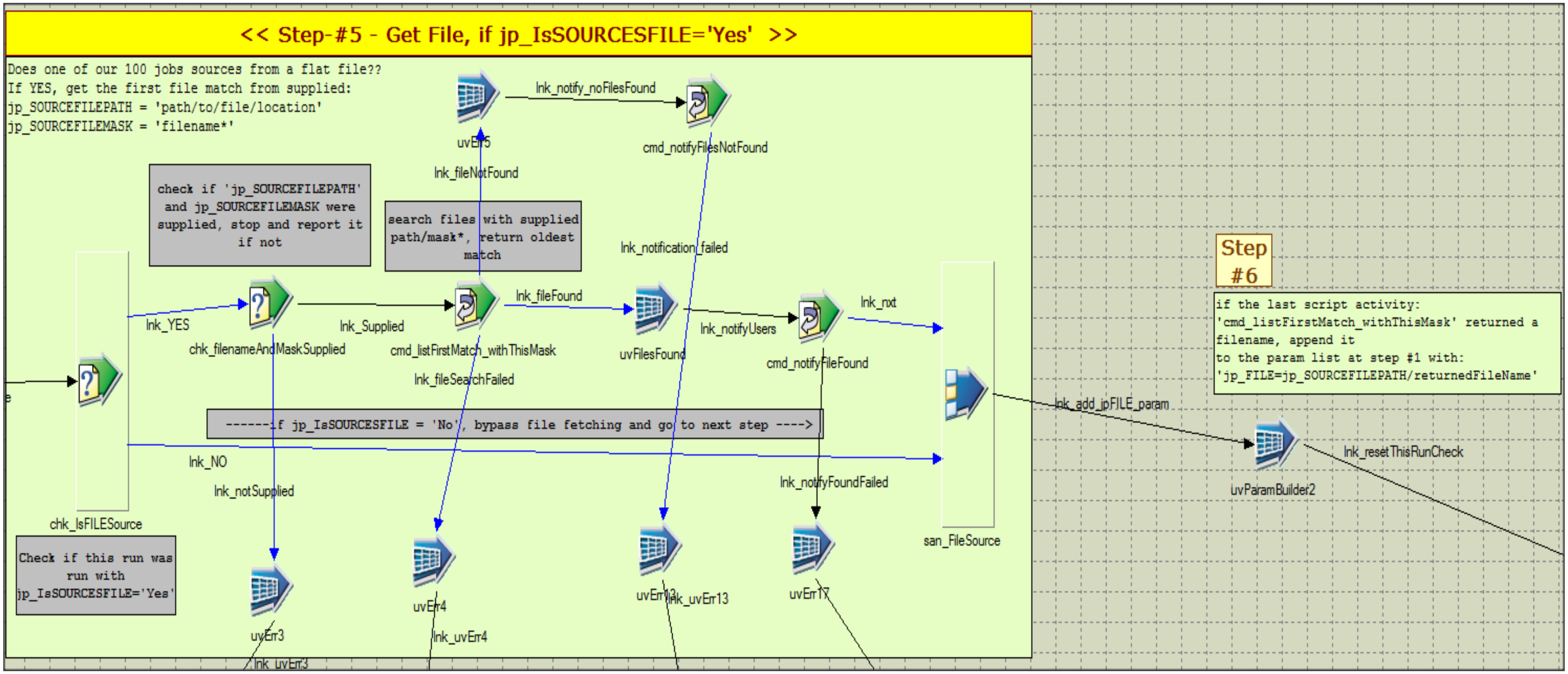
<< Step-#4.1.1 - Run jobs in list and check rejects after every run (if jp_STOP_AT_REJECTS = 'Yes') >>

1-> Loop over the found comma seperated list of jobs,
2-> run them one by one, report error / warnings in any job and stop,
3-> cleanup temp files after the run

Notify Exact job name that failed and stop

Go back to main sequencer 🡐

1) If jp_IsSOURCEFILE = 'Yes', check if values for jp_SOURCEFILEPATH and jp_SOURCEFILEMASK was supplied
   -> if not supplied, report users and stop
2) if supplied, get the last file name match of supplied jp_SOURCEFILEPATH and jp_SOURCEFILEMASK
3) Append to step-#1 parameters one more param value:
   jp_FILE="path/to/drop/location/returnedFilenameLastMatch"



<< Step-#5 - Get File, if jp_IsSOURCESFILE='Yes' >>

Does one of our 100 jobs sources from a flat file??
If YES, get the first file match from supplied:
jp_SOURCEFILEPATH = 'path/to/file/location'
jp_SOURCEFILEMASK = 'filename*'

check if 'jp_SOURCEFILEPATH' and jp_SOURCEFILEMASK were supplied, stop and report it if not

search files with supplied path/mask*, return oldest match

------If jp_IsSOURCEFILE = 'No', bypass file fetching and go to next step ---->

Check if this run was run with jp_IsSOURCESFILE='Yes'

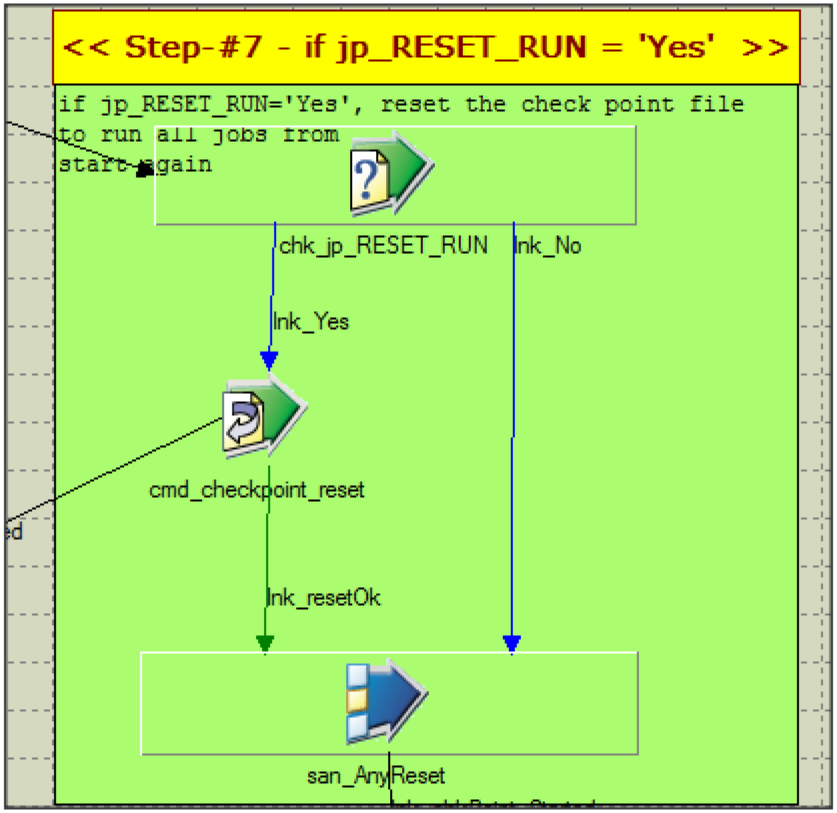
Step #6

if the last script activity:
'cmd_listFirstMatch_withThisMask' returned a filename, append it
to the param list at step #1 with:
'jp_FILE=jp_SOURCEFILEPATH/returnedFileName'

Go back to main sequencer ←

1) If jp_RESET_RUN = 'Yes', Clear the check point file of previous job states, so that it can run from first job again

!! WARNING !! – production CPS schedule should never be run with JP_RESET_RUN = 'Yes', this is for one time run scenarios

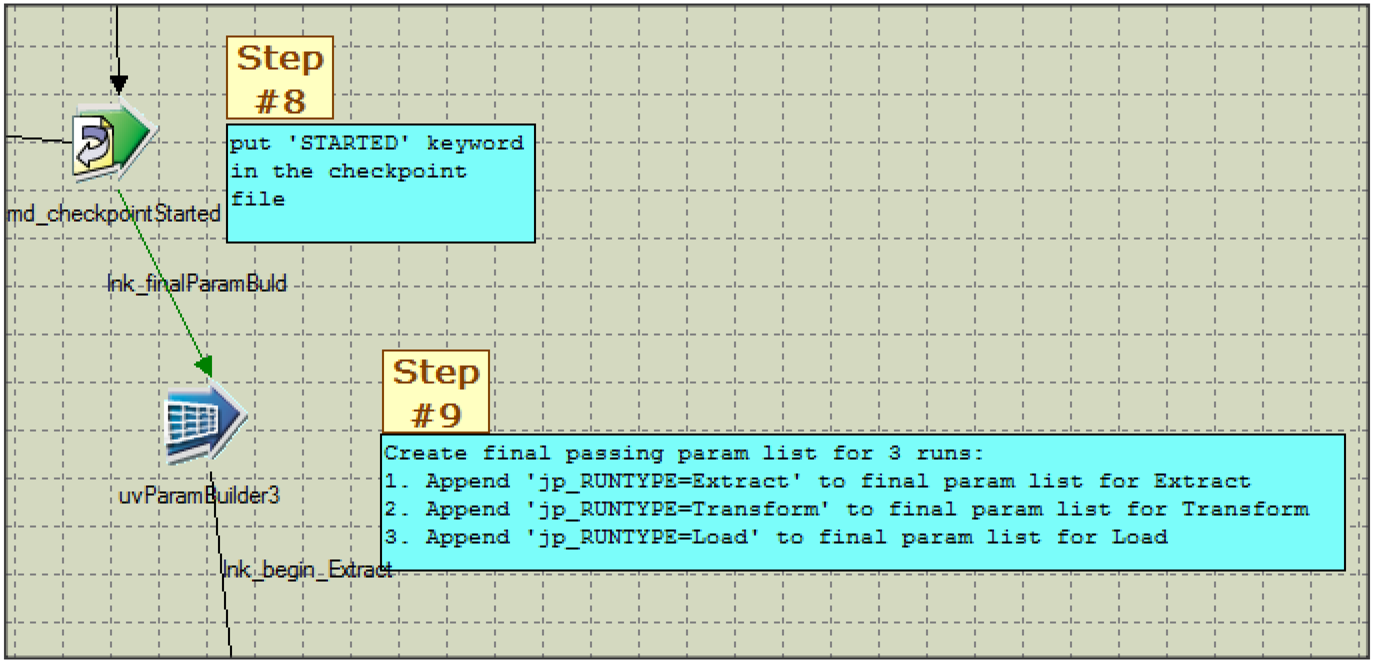


<< Step-#7 - if jp_RESET_RUN = 'Yes' >>

if jp_RESET_RUN='Yes', reset the check point file to run all jobs from start again

Go back to main sequencer ←

1) Put "=== Started run for : seq_Common_JobFactory_MAIN.{thisJobName} ===" in the log
2) Create a final list of params for three runs: Extract (100*), Transform (200*), Load (300*)
3) Create a final list of params for Lookup run: (110) jobs in your [jp_SUBJECTAREA]

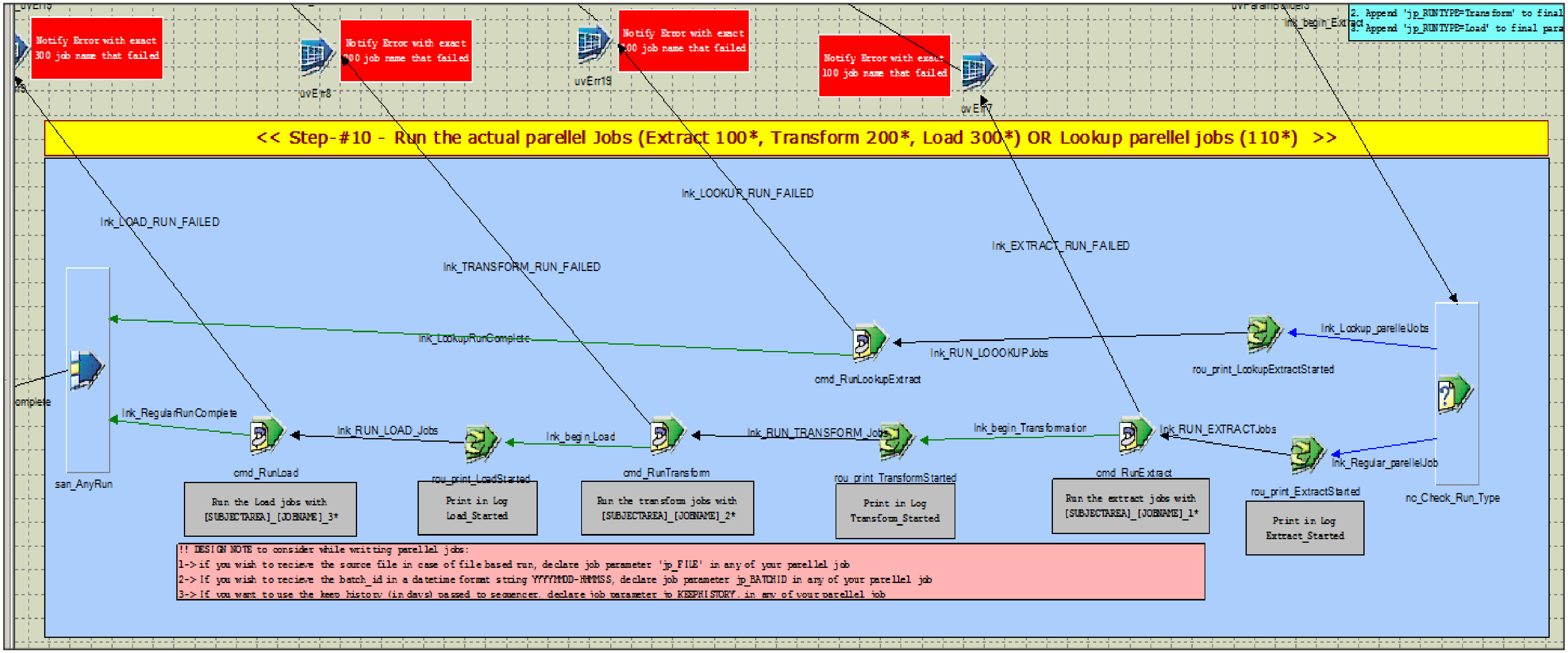


put 'STARTED' keyword in the checkpoint file

Create final passing param list for 3 runs:
1. Append 'jp_RUNTYPE=Extract' to final param list for Extract
2. Append 'jp_RUNTYPE=Transform' to final param list for Transform
3. Append 'jp_RUNTYPE=Load' to final param list for Load

Go back to main sequencer ←

```
1) Check the jp_RUN_TYPE value?
      1.1) If [REGULAR_JOB]
            1.1.1) Print in Log "Started_[jp_SUBJECTAREA]_[jp_JOBNAME]_Extract | Transform | Load started"
            1.1.2 ) Run the job run scripts for 100*, 200*, 300* jobs in sequence
      1.2) If [LOOKUP_JOB]
            1.2.1) Print in Log "Started_[jp_SUBJECTAREA]_Lookup Extract has Started"
            1.2.2) Run the job script for 110* jobname matches
```



Notify Error with exact 300 job name that failed

Notify Error with exact 00 job name that failed

Notify Error with exact 00 job name that failed

Notify Error with exact 100 job name that failed

2. Append 'jp_RUNTYPE=Transform' to final

3. Append 'jp_RUNTYPE=Load' to final para

<< Step-#10 - Run the actual parellel Jobs (Extract 100*, Transform 200*, Load 300*) OR Lookup parellel jobs (110*) >>

Run the Load jobs with [SUBJECTAREA]_[JOBNAME]_3*

Print in Log Load_Started

Run the transform jobs with [SUBJECTAREA]_[JOBNAME]_2*

Print in Log Transform_Started

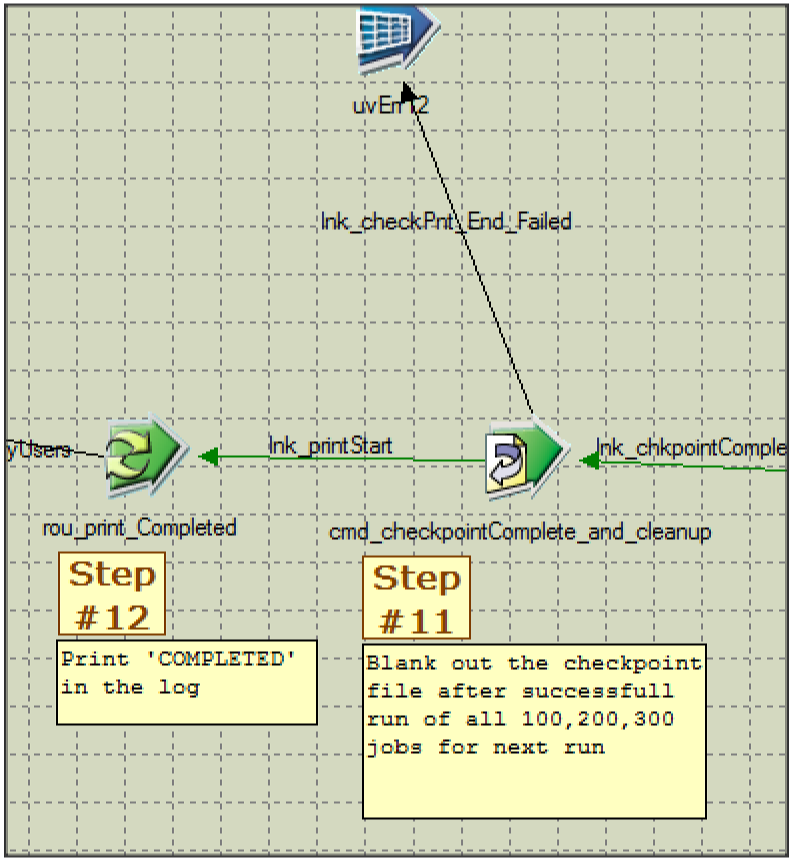
Run the extract jobs with [SUBJECTAREA]_[JOBNAME]_1*

Print in Log Extract_Started

!! DESIGN NOTE to consider while writting parellel jobs:
1-> if you wish to recieve the source file in case of file based run, declare job parameter 'jp_FILE' in any of your parellel job
2-> If you wish to recieve the batch_id in a datetime format string YYYYMMDD-HHMMSS, declare job parameter jp_BATCHID in any of your parellel job
3-> If you want to use the keep history (in days) passed to sequencer, declare job parameter jp KEEPHISTORY, in any of your parellel job

Go back to main sequencer ←

1) Delete the checkpoint file so next job can begin from start, if jp_CHECKPOINT_RUN = 'Yes'
2) Print in log "====Completed Run for : seq_Common_JobFactory_MAIN.[jp_SUBJECTAREA]_[jp_JOBNAME]===="



Step #12

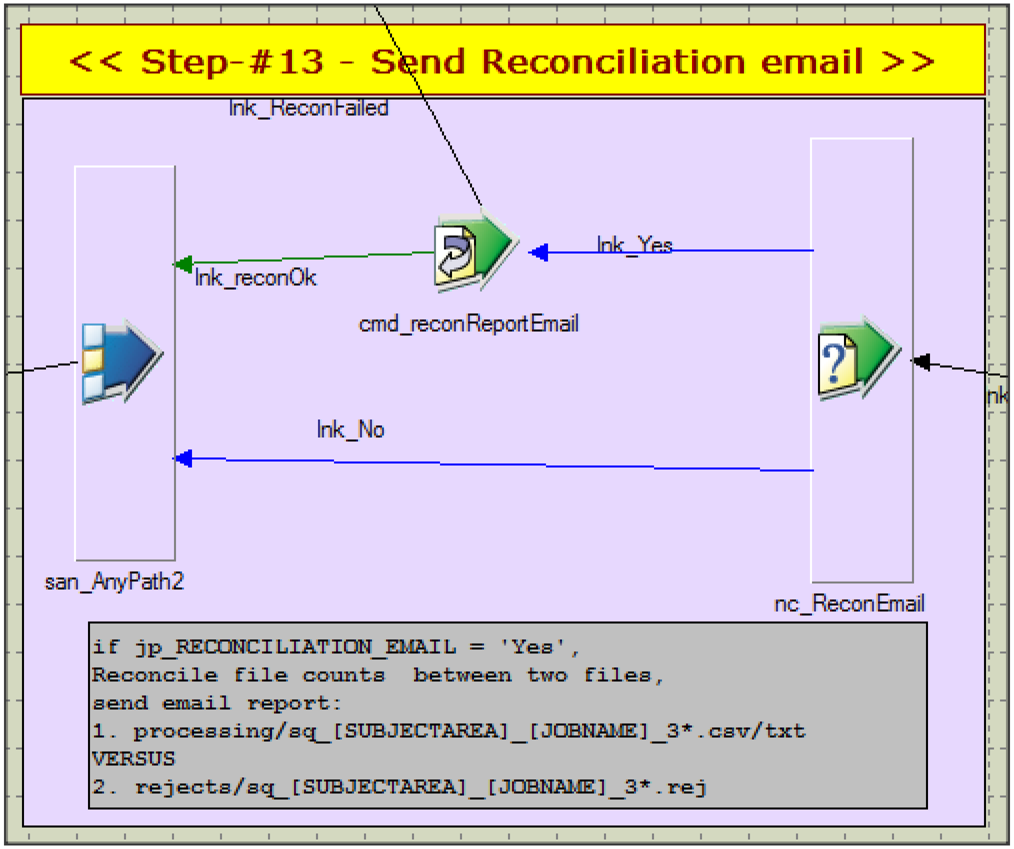
Print 'COMPLETED' in the log

Step #11

Blank out the checkpoint file after successfull run of all 100,200,300 jobs for next run

Go back to main sequencer ←

1) Check if jp_RECONCILIATION_EMAIL = ‘Yes’,
1A) if NO, do nothing, go to next step - #14
1B) if YES, run script that compares the following files and sends a recon report email:
‘processing/sq_[jp_JOBNAME]_*.csv|txt’ with its
‘rejects/sq_[jp_JOBNAME]_*.rej’



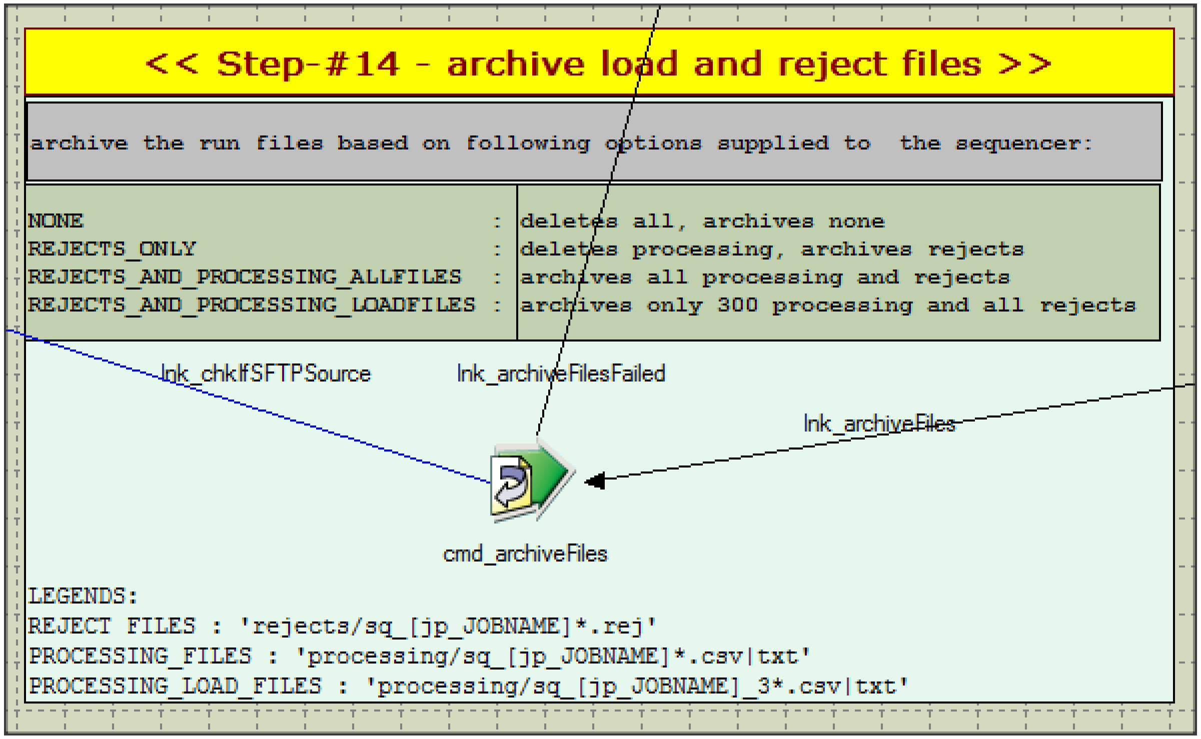
```
if jp_RECONCILIATION_EMAIL = 'Yes',
Reconcile file counts  between two files,
send email report:
1. processing/sq_[SUBJECTAREA]_[JOBNAME]_3*.csv/txt
VERSUS
2. rejects/sq_[SUBJECTAREA]_[JOBNAME]_3*.rej
```

Go back to main sequencer ←

1) Check the jp_ARCHIVAL_OPTIONS passed:
1A) if 'NONE' : Deletes all processing and reject files
1B) if 'REJECTS_ONLY' : Archives only reject files, deletes all the processing files
1C) if 'REJECTS_AND_PROCESSING_ALLFILES' : Archives both all the processing as well as reject files
1D) if 'REJECTS_AND_PROCESSING_LOADFILES' : Archives only 300* processing files and all reject files, delete remaining processing files

LEGENDS:

processing files	: 'processing/sq_[jp_JOBNAME]_*.csv|txt'
processing load files	: 'processing/sq_[jp_JOBNAME]_3*.csv|txt'
reject files	: 'rejects/sq_[jp_JOBNAME]_*.rej'



<< Step-#14 - archive load and reject files >>

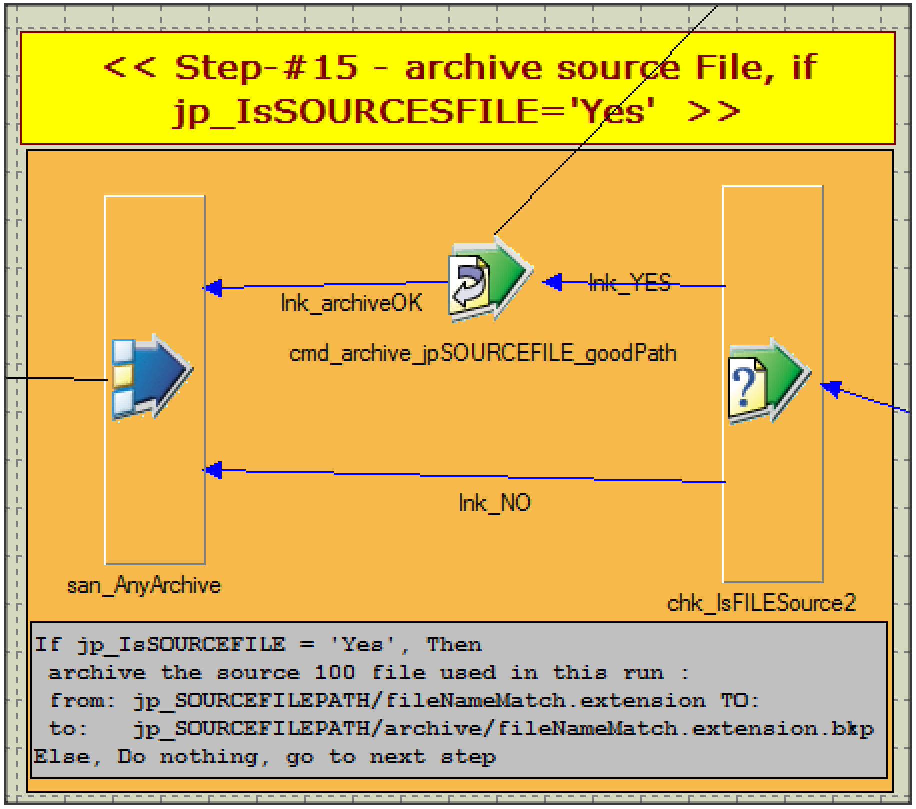
archive the run files based on following options supplied to the sequencer:

NONE :	deletes all, archives none
REJECTS_ONLY :	deletes processing, archives rejects
REJECTS_AND_PROCESSING_ALLFILES :	archives all processing and rejects
REJECTS_AND_PROCESSING_LOADFILES :	archives only 300 processing and all rejects

LEGENDS:
REJECT FILES : 'rejects/sq_[jp_JOBNAME]*.rej'
PROCESSING_FILES : 'processing/sq_[jp_JOBNAME]*.csv|txt'
PROCESSING_LOAD_FILES : 'processing/sq_[jp_JOBNAME]_3*.csv|txt'

Go back to main sequencer ←

1) If the job was run with jp_IsSOURCESFILE = ‘Yes’:
move the ‘jp_SOURCEFILEPATH/firstMatch.**extension**’ **to**
‘jp_SOURCEFILEPATH/**archive**/firstMatch.**extension.bkp**’

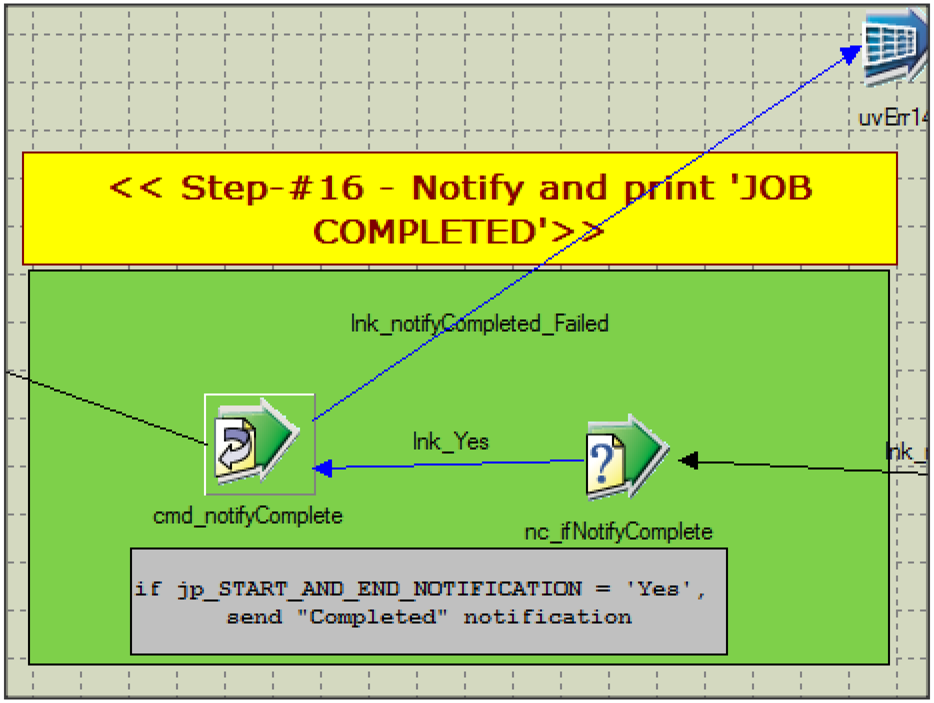


<< Step-#15 - archive source File, if jp_IsSOURCESFILE='Yes' >>

```
If jp_IsSOURCEFILE = 'Yes', Then
 archive the source 100 file used in this run :
 from: jp_SOURCEFILEPATH/fileNameMatch.extension TO:
 to:   jp_SOURCEFILEPATH/archive/fileNameMatch.extension.bkp
Else, Do nothing, go to next step
```

Go back to main sequencer ←

1) If jp_START_AND_END_NOTIFICATION = 'Yes', then send email to users stating :
'The job run : **[jp_SUBJECTAREA]_[jp_JOBNAME]** has **COMPLETED**'



<< Step-#16 - Notify and print 'JOB COMPLETED'>>

if jp_START_AND_END_NOTIFICATION = 'Yes', send "Completed" notification

Go back to main sequencer ←

1) This is random failure collection, do the following on reaching here:

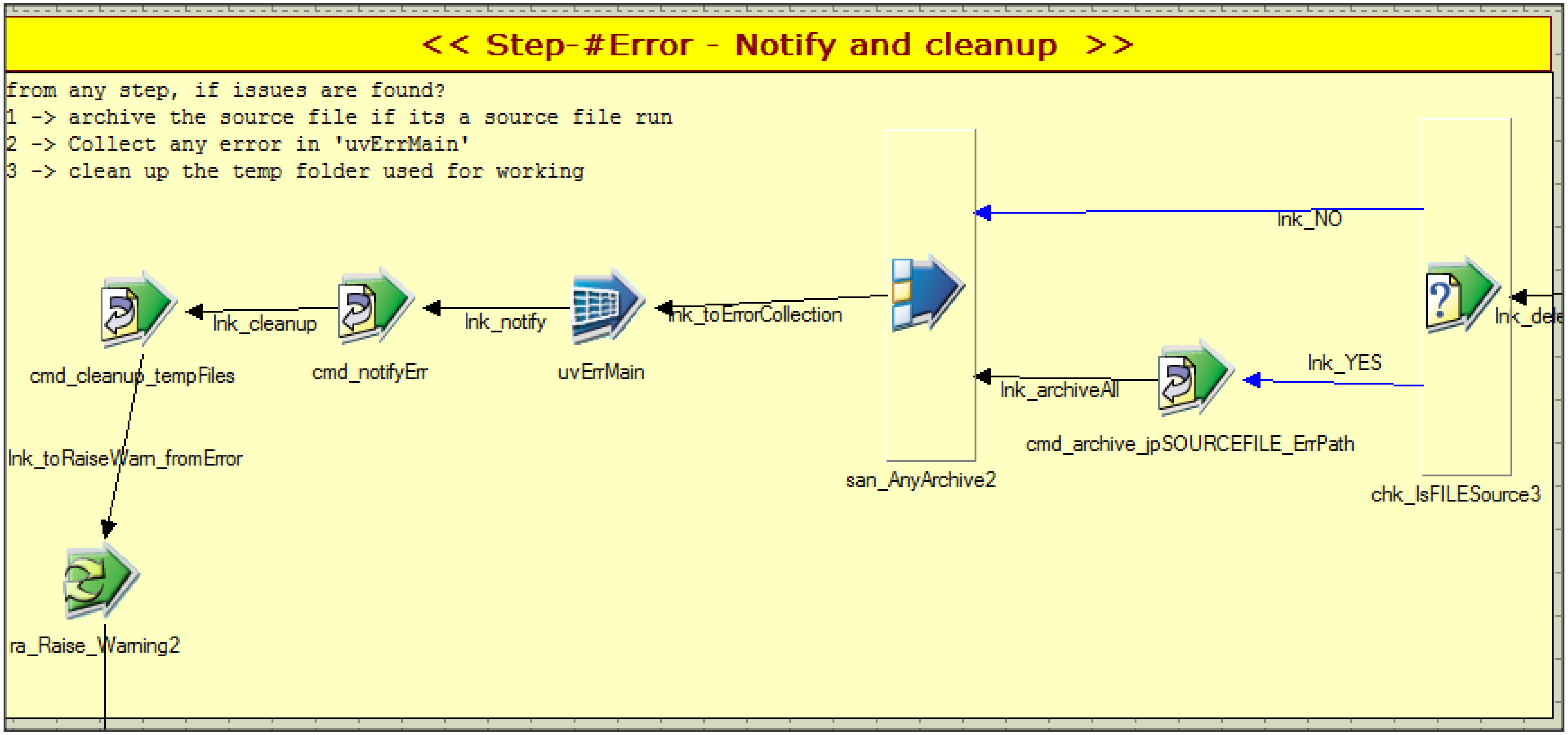
1.1)If **jp_IsSOURCESFILE='Yes'**:

  1.1.1) send the source file from: jp_SOURCEFILEPATH/firstFileMatch to: jp_SOURCEFILEPATH/**archive**/firstFileMatch**.bkp**

1.2) Assemble the main error message from all the various errors at various step (uvErrMain)

1.3) cleanup the temporary files created during running of various scripts

1.4) Raise warning and stop

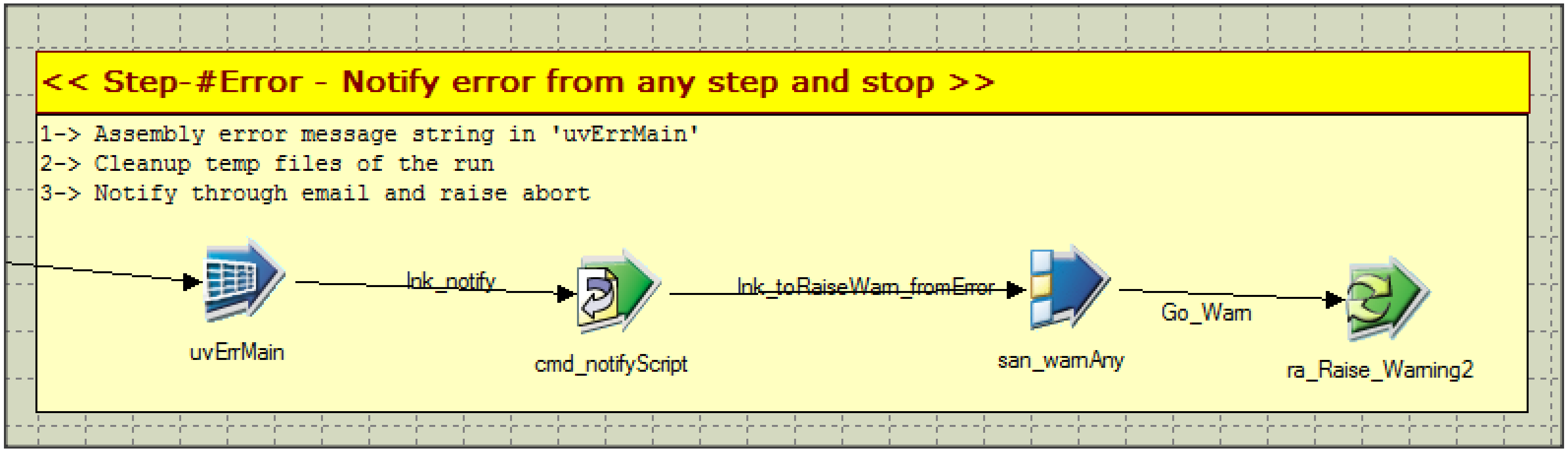


<< Step-#Error - Notify and cleanup >>

from any step, if issues are found?
1 -> archive the source file if its a source file run
2 -> Collect any error in 'uvErrMain'
3 -> clean up the temp folder used for working

Go back to RunJobs sequencer ←

1.) Assemble error messages into a 'uvErrorMain'
2.) notify users of the failure
3.) Raise **Abort** and stop



<< Step-#Error - Notify error from any step and stop >>

```
1-> Assembly error message string in 'uvErrMain'
2-> Cleanup temp files of the run
3-> Notify through email and raise abort
```