# Restart-ability in Matillion Jobs

Restart-ability refers to the ability of a Batch (or Process) to resume processing from point of failure. Failures may be caused by data, logic or environmental issues. DataStage provides built-in ability to restart from beginning (Reset) or resume a batch from the last point of failure.

Since, Matillion jobs replicate the exact same functionality of DataStage jobs, it will still be possible to restart a Matillion batch from the beginning (Reset). However, Matillion doesn't support resuming processing from the last point of failure, unlike DataStage. This poses a major problem for support teams since some large batches, may take 1-2 hours to complete and a failure in between could mean significant loss of time. Also, multiple failures could make the matters worse.

This note discusses proposed implementation of restart-ability for Matillion jobs.

## Matillion Implementation

Exact replication of DataStage functionality to resume from point of failure is not feasible in Matillion. However, a close approximation can be achieved by use of 'Checkpoints'. A checkpoint is a shared component that can be placed anywhere in any orchestration job. Any number of checkpoints can be placed within a batch or even within a single orchestration job. Checkpoints are self-configuring components that need no other input from the developers.

When a batch runs, checkpoints create a unique ID for themselves and log a Checkpoint event using ABC Framework. They also dump the Batch context (state of all job variables) into the EVENT_DETAIL field of the log. In the event of a failure, support teams can check the event log for the failed batch and note down the last successful checkpoint ID. They can then rerun the failed Batch by specifying the checkpoint ID from where processing is to be resumed.

Checkpoints have built in ability to skip the processing up to next checkpoint, if they are earlier in the sequence than the start checkpoint specified at run time. The start checkpoint on the other hand, reads the logs from ABC database and restores the context by assigning job variable values, available at the time of failure.

Checkpoints perform following functions:

- Increment Checkpoint number (ID) by 1 to generate a unique ID for that run.
- IF current ID is GREATER than starting ID. — true for normal run and post restart checkpoint
    - Capture state
    - Log a Checkpoint-Complete event
    - Continue processing.

- IF current ID is LESS than starting ID. — will be true only for restart before restart checkpoint
    - Log a Checkpoint-Skipped event
    - Proceed to next checkpoint.
- IF current ID is EQUAL to starting ID. — will be true only at the restart checkpoint
    - Read EVENT LOG for the specified batch run and start checkpoint combination
    - Extract State JSON from EVENT_DETAILS
    - Assign Job Variable values from the State JSON.
    - Log a Checkpoint-Resumed event
    - Continue processing.

## Illustrative Example

Diagram below shows a batch with main sequence and 5 sub-sequences (orchestrations). Batch Start and Batch End components of ABC Framework are added to enable Audit and logging of events. Four check points are placed prior to Process1 through to 4. Following parameters are passed to the Batch at run time:

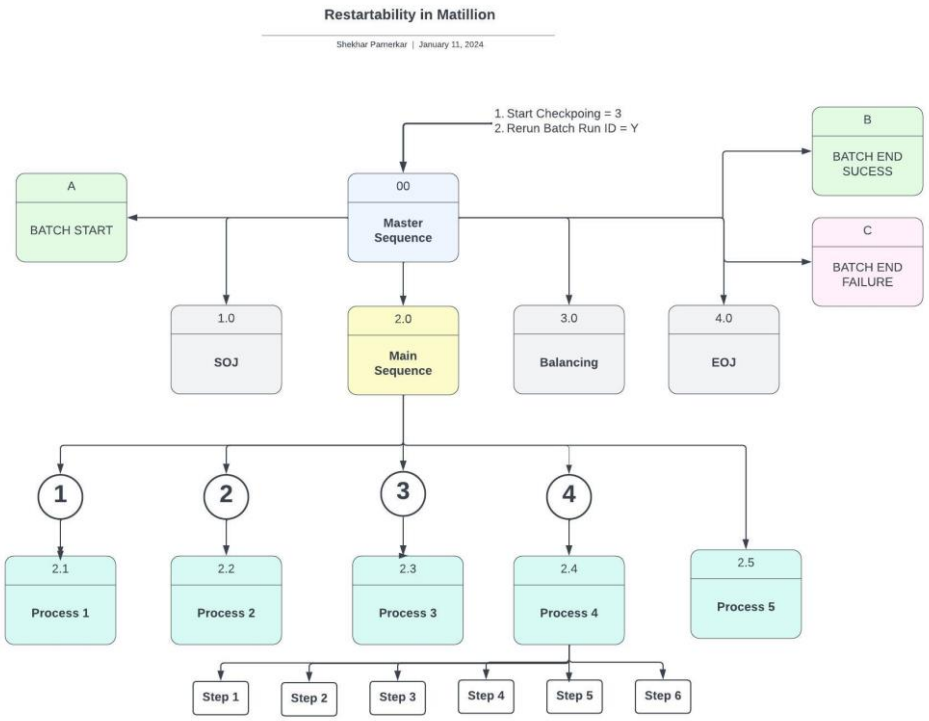
1. Start Checkpoint
2. Rerun Batch Run ID

Both will be set to 0 for initial run by HWA and greater than zero to specify a restart.

When the batch starts, Master sequence will call Batch Start, SOJ and then pass control to Main Sequence. Main Sequence calls Checkpoint 1, which in turns decides if Process1 should be excited or skipped. Then control moves to the next component in Main Sequence. In this diagram to Checkpoint 2, which decides if Process2 should be executed. If we remove Checkpoint 2, Process2 will always be executed regardless.

If the batch fails (say during Process4), a Batch End Failure event will notify the support team of a failure. They can then restart the batch by specifying the start checkpoint (4) and failed batch run id. In this case checkpoints 1,2 and 3 will bypass Process 1,2 and 3 but Checkpoint 4 will resume processing from Process 4.

Process 5, Balancing and EOJ components will be executed as usual followed by Batch End Success event. Should a failure occur again in Process 4 or later jobs, restart will still be required from Checkpoint 4 only, since no other restart point is available after that. To cover Process 4 and 5, a checkpoint can be placed just before Balancing, which will configure itself as Checkpoint 5 - following the order of execution.



## Idempotency Considerations

Idempotency refers to the ability of a process to produce the exact same output, no matter how many times it is run. When restarting a batch either from beginning or from any checkpoint, two factors play an important role:

1. Batch Context - state of all job variables associated with it.
2. Data being processed - input, output and intermediate.

The batch context is saved and restored by the Checkpoints, when batch resumes - therefore that will not be an issue. But data is a much bigger consideration. For a Batch rerun to produce expected output from the failed run, following conditions must be satisfied:

1. Input data should not change or there should be a definitive mechanism to select it for a given run.
2. In case of input files, they should remain intact and unchanged for the restart
3. Output data should be restored to status prior to initial run by deleting incomplete data loaded by the failed run.
4. Temporary data in files and tables should be removed before writiing into them for the first time.

In other words, the batch should be designed to be idempotent. To enable restart from a checkpoint, the processes between checkpoints should also be idempotent - they should clear the temporary data and reverse the previous updates made to persistent tables. Where application logic doesn't provide this function, support team (BAU) needs to analyse the failure and do the same manually. The decision to reset or resume is also made by the support team considering the point of failure and effort required to clean the data, if needed.

For examples, in the above example, let us say, a failure occurred at Process 4: Step 4. When Process 4 resumes, Steps 1 through to 3 will be executed again. Let's assume that these steps write data into a temporary table. In this case Process 4 should be designed in such a way that temporary table is cleared prior to, or at Step 1, else, data will be duplicated in the temporary tables.

Idempotency considerations requires careful thought when designing new applications. Where application fails to provide this ability, the Support team needs to analyse the failure and clean up the data manually to avoid corrupting the database.

## Next Steps

As part of migration project, DataStage applications are being migrated to Matillion without any change in their functionality or design. Therefore, they will behave in the same way when it comes to restart. However, following additional steps shall be taken in the development process towards idempotency:

1. Development teams shall touch base with respective SMEs to identify potentially time-consuming jobs and place checkpoints at appropriate points.
2. Add unit test cases to simulate failures at some key checkpoints
3. Cause the applications to fail at those points
4. Restart application from those checkpoints.
5. Note the result in Unit Test Record.

Development teams are already advised to clear temporary tables before inserting data into them for the first time.