Example showing how to run parallel jobs repeatedly in a job chain

Availability

- FEATURE AVAILABILITY STARTING FROM RELEASE 1.11.4
- See How to Execute Jobs in a Job Chain in Parallel for more information.

Example: Multiple parallel processes in a job chain

- The job chain is to start with a job with the name *truncate_export_table*.
- After this job has been completed four jobs with the name *table partition* are to be run in parallel.
- A single job that indexes the new partition tables is then to run.
- Finally, a further four jobs that test the individual partition tables are to start in parallel.
  This job chain is shown schematically in the diagram in the Diamond section below.

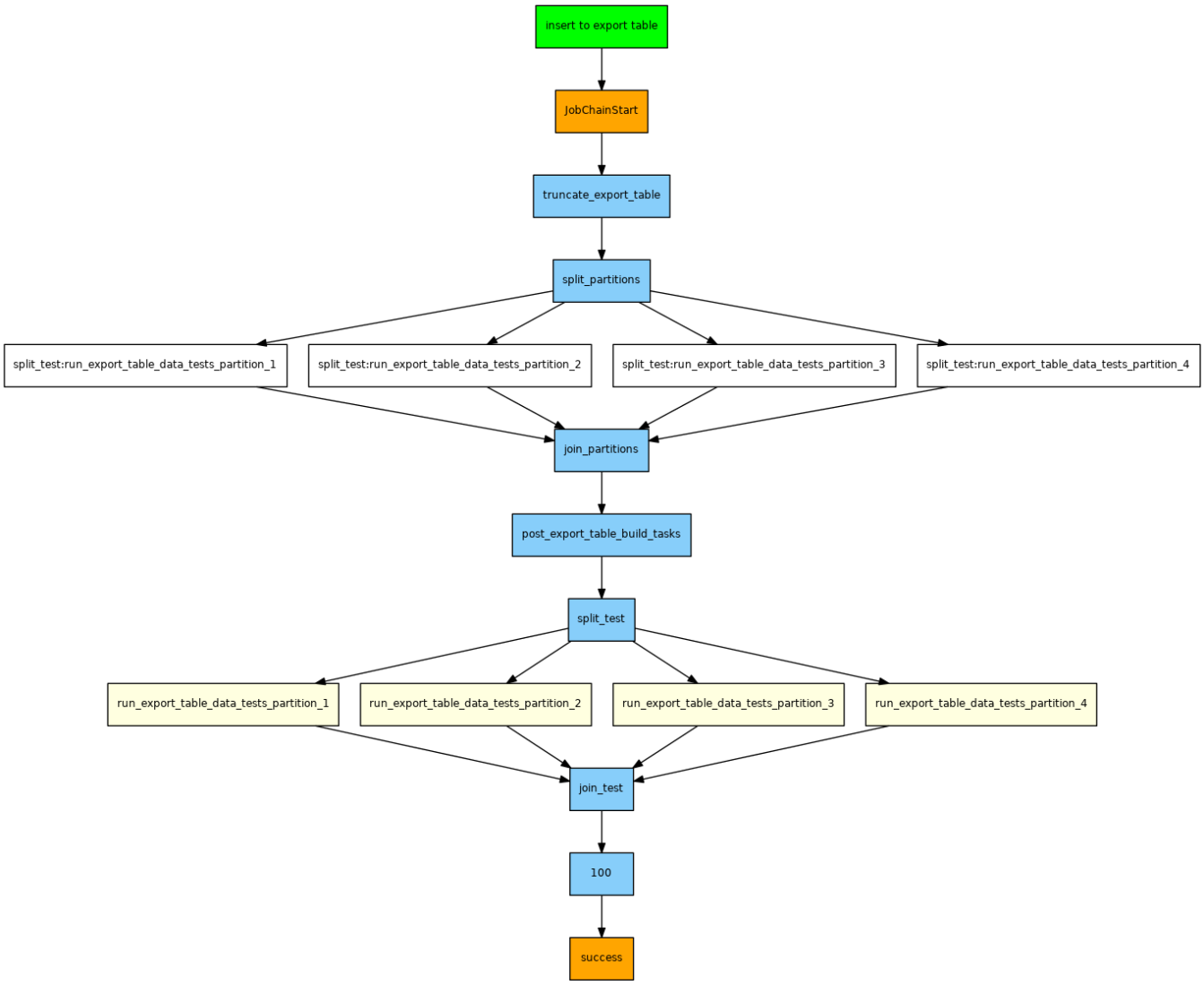
Writing the job chain

The following steps have to be followed to realize a job chain that meets the requirements listed above:

1. A "splitter" job has to be included for each "series" of job nodes that are to be processed in parallel. The splitter generates an order for each series of parallel job nodes starts these orders as soon as it itself has been started.
2. In order to do this the splitter job has to "know" the names of the states corresponding to the first node of each job node series. These names are specified in the splitter job's *state_names* parameter (see How to set and read job and order parameters).
3. The parallel processing normally ends at a specific node in the the chain with processing then continuing serially. This join-up node is implemented using the Join-Job.

The "diamond" job chain structure

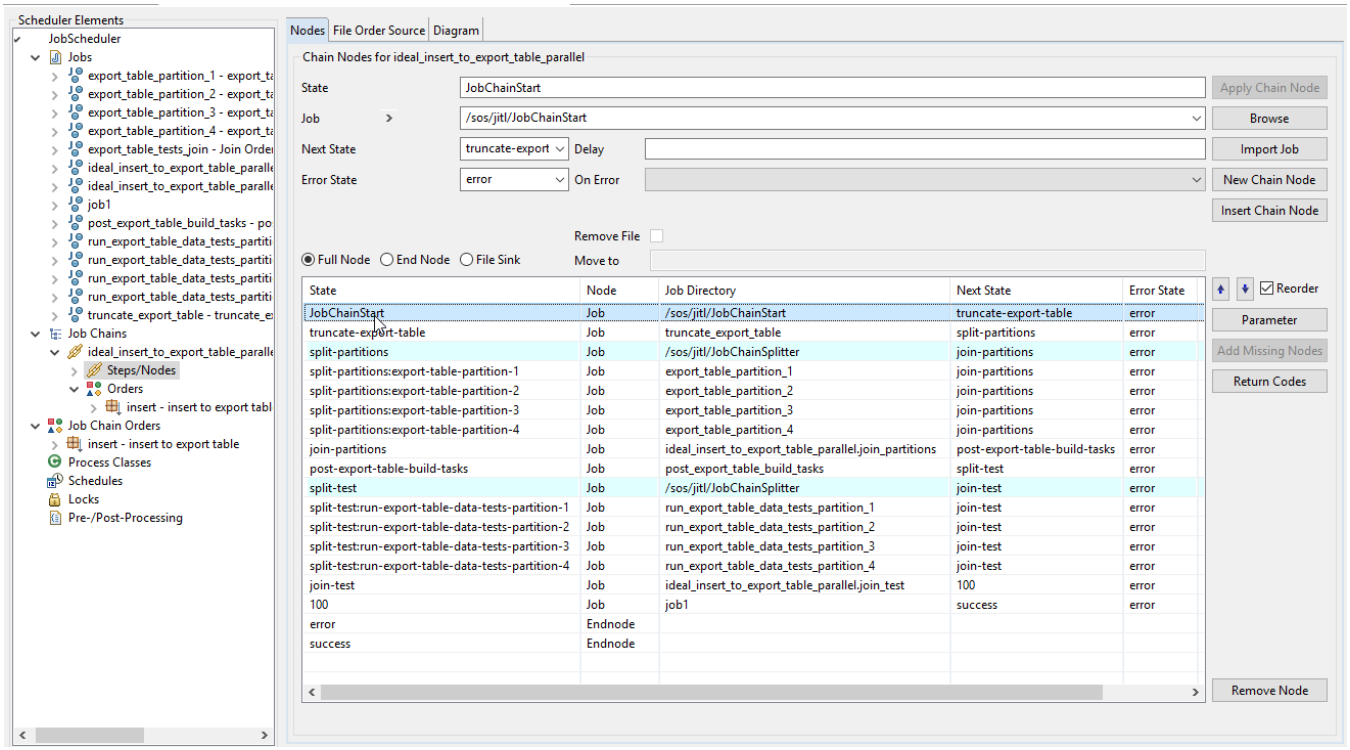
The example job chain will look like this:



We refer to the pattern that results in this type of job chain as a "diamond" pattern. Such patterns can occur more than once in a Job Chain: both sequentially, as shown in the diagram above, in parallel and nested. They can also be combined with other job chain patterns such as emerald or cross-over patterns (see Example showing the synchronization of multiple job chains).

Job chain list view

The next illustration shows a list view of the job chain as created by JOE:



The "Splitter" job

A generic splitter job is delivered with the JobScheduler JITL jobs. This job can be found in the *.//ive/sos/jitl/* directory.

We recommend that the following syntax is used for the names of job nodes that are to be processed in parallel:

"node name of the splitter job" ":" "job name". For example *split_partitions:export_table_partition_1*.

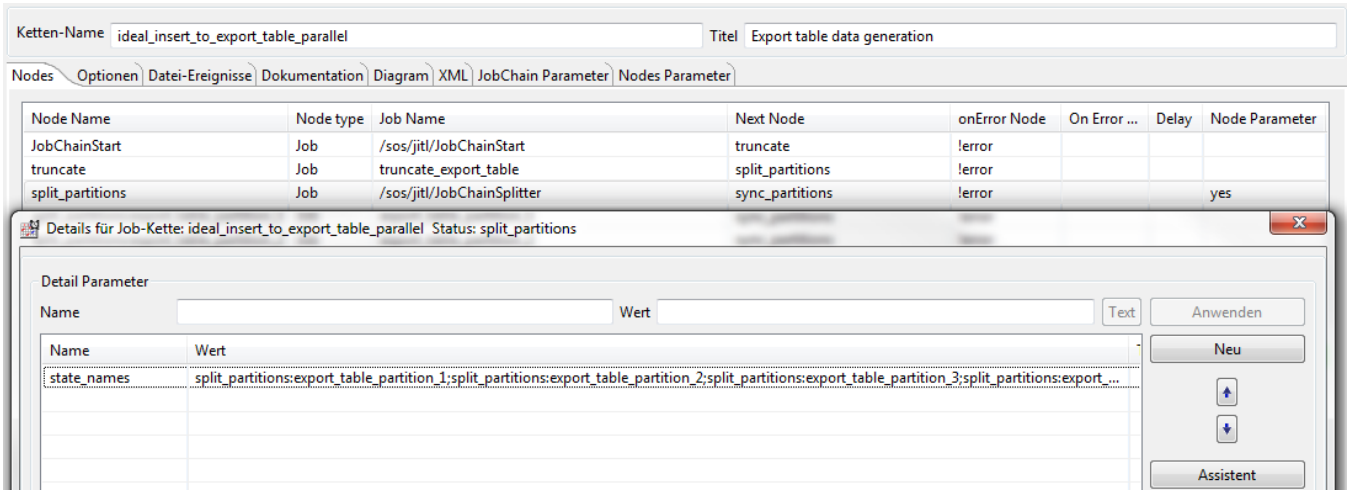
Splitter job parameters

See documentation of the JobChainSplitter job.

The *state_names* parameter

- The splitter job *state_names* parameter is used to specify the node names of the jobs that are to be first started in parallel (see How to set and read job and order parameters).
- The node names are to be separated by semi-colons.
- In jobs chains with this diamond pattern structure, the parameters are specified for the job chain and referred to as node parameters. Node parameters can be used to specify parameters for more than one splitter in a job chain, independently of one another, as in our example, without creating conflicts.

The parameters for the *split_partitions* splitter job - as shown in JOE - are:



The "Join" job

A unique join-up job is required at the end of every group of processes running in parallel (see Example showing how to run parallel jobs in a job chain) when the nodes in the job chain after the join node are only to be processed after all the jobs (tasks) that are to be carried out in parallel have been completed without errors.
Each join-up job has to be unique within a JobScheduler instance - and within a job chain.

For more information see the documentation for the JobSchedulerJoinOrders job.

Related Downloads

- You can download the example from: parallel_execution.zip

See also

- Documentation of the JITL jobs:
  - JobChainSplitter
  - JobSchedulerJoinOrders
- A overview of the use of Split & Join Jobs:
  - How to Execute Jobs in a Job Chain in Parallel.
- Job Dependencies in the JobScheduler:
  - Job dependencies