Example showing how to set up a Join Orders Job

Scope

The *Join Orders* Job (JobSchedulerJoinOrders) is a JITL Job and is used to join up parallel executing child segments in a Job Chain. It then continues processing with a single Order in a single thread once processing of the parallel threads has been completed. It is used in two Job Chain Patterns - *Split and Join* and "*Y*".

This article describes how the *Join Orders* Job can be used in a "Y" pattern - the How to Execute Jobs in a Job Chain in Parallel article describes its use in a *Split and Join* pattern.

The *Join Orders* Job is only available for JobScheduler versions 1.11.4 and newer. FEATURE AVAILABILITY STARTING FROM RELEASE 1.11.4

- See the How To Synchronize Job Execution Across Job Chains article for information about using the *Sync* Job (JobSchedulerSynchronizeJobChains) to synchronize Jobs in *different* Job Chains and to join up parallel executing child Job Chain segments in JobScheduler versions 1.11.3 and older.
- The *Join Orders* Job provides a plug-in replacement for the *Sync* Job for sync/join operations *within* a Job Chain. As the *Join Orders* Job is significantly faster than the *Sync* Job, users of JobScheduler 1.11.4 or newer wishing to improve performance of the *Sync* Job can simply replace their *Sync* Job with the *Join Orders* Job.

Join Patterns

The example described in this article shows the use of a single instance of the *Join Orders* Job within a single Job Chain. Multiple instances of the *Join Orders* Job can also be used within a Job Chain. See the Configuration section of the JobSchedulerJoinOrders documentation for more information.

split and join Pattern

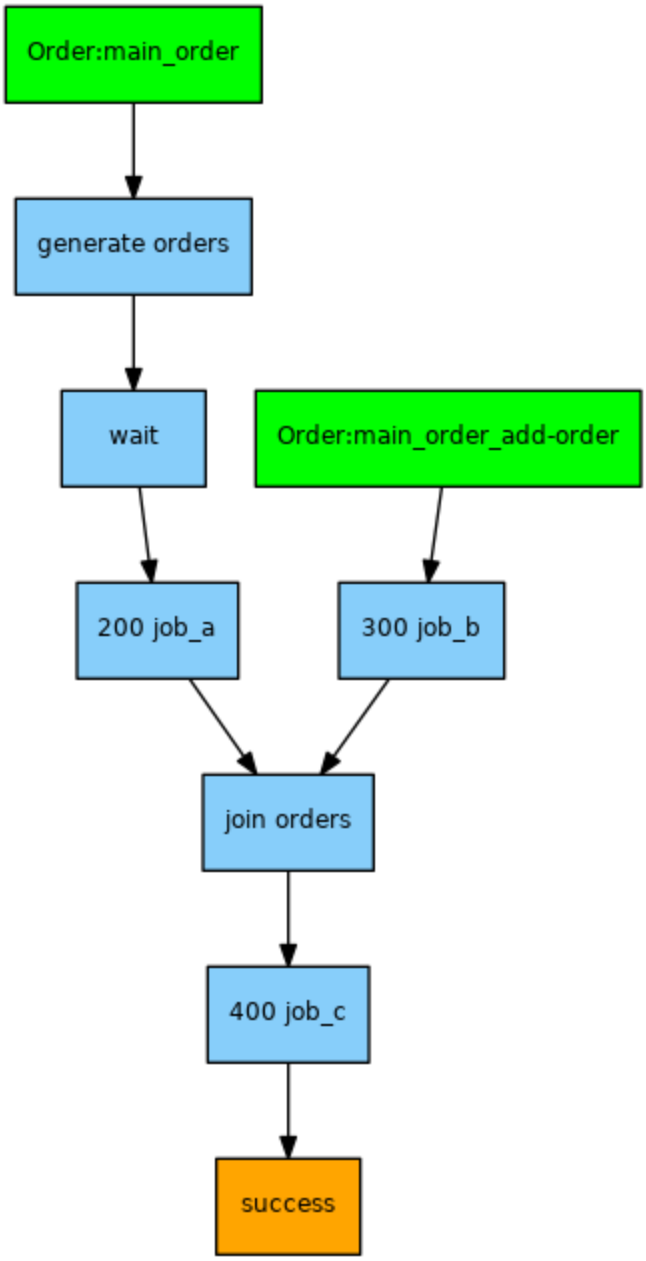
The configuration of the *Join Orders* Job for use in the *Split and Join* pattern is described in the How to Execute Jobs in a Job Chain in Parallel article.

Relevant for the current article is that in the *Split and Join* pattern, the *Split* Job generates the configuration information required by the *Join Orders* job. This information must be configured separately for the "Y" pattern. The information generated by the the *Split* Job is:

- a parameter - `required_orders` - which is forwarded to the *Join* Job. This parameter defines the number of orders that the *Join Orders* Job has to receive before processing the Job Chain continues with processing of the *Join Orders* Job itself
- an Order for each of the child Job Chain segments that is to be processed in parallel. The end state of each of these Orders is the state of the *Join Orders* Job.

'Y' Pattern

The following diagram shows the "Y"-pattern Job Chain used in the example download archive that is linked from this article.



- Note that the *generate* and *wait* Jobs in this example have been included to make the example work and are not required for its operation in "normal" use.
- All Jobs in the "Y"-pattern are configured as a single Job Chain.
- The parallel arms of the example are effectively *job_a* to the *Join Orders* Job and *job_b* to the *Join Orders* Job.

Download and Configure the Example

- Download the example y_join.zip.
- Extract the archive to a folder `./config/live` of your JobScheduler Master installation.
- The archive will extract the files to a folder `y_join`.
- The `y_join` folder can be renamed if required, the solution does not require the use of specific folder or Job names.
- Change the path to the example Job Chain in line 14 of the *generate_orders* Job below to the correct setting.

Example Description

To use the example, first start the Parent Order (in this example *main_order*). This Order has been configured with three parameters, one of which - *required_orders* - is required by the *Join Orders* Job. The other two parameters - *generate_orders* and *duration_wait_job* are only used to provide a convenient way of making the example work.

- The *required_orders* parameter is read immediately by the *Join Orders* Job, which will then wait in the *suspended* state until the number of Orders specified in the parameter (here it is 10) have been processed.

- The *main_order* moves to the *generate_orders* Job, which generates a total of 8 Child Orders.
    - This number is specified in a second parameter for the *main_order - generate_orders*.
    - 4 of these child orders be processed in one arm of the "Y" - i.e. starting at node 200 - and 4 will be processed in the other arm - i.e. starting at node 300. All the Child Orders will end at the *Join* Job.
        - The branch of the Job Chain the Child Orders have been executed on is not important for the *Join Orders* Job.
    - All these Child Orders will be counted towards the *required_orders* parameter.
    - Child Orders are identified by the *Join Orders* Job when they either:
        - Have an Id based on the Parent Order ID plus an underscore plus a string.
            - For example, the first generated Child Order in the example will be given the ID *main_order_0*.
            - This is the method used in the example.
        - They carry a *join_session_id* parameter, specifying the ID of the Parent Order.
- The *main_order* itself then moves to the *wait* Job where it waits for a time specified in the *main_order*'s *duration_wait_job* parameter (here 30 seconds).
    - The sole purpose of this delay is to demonstrate that the *main_order* can reach the *Join Orders* Job *after* Child Orders.
    - This delay is not necessary for the functioning of the *Join Orders* Job and the example will work with the *duration_wait_job* parameter set to its minimum value of 1 (0 will cause an error).
- When the *main_order* reaches the *Join Orders* Job it will be counted towards the number of Orders specified in the *required_orders* parameter, making a total of 8 + 1 = 9 Orders after all the generated Child Orders have been completed.
    - Note that the *main_order* is the only Order that will be counted that does not have to have the state of the *Join Orders* Job as its end state.
- The *main_order* will now be suspended at the *Join Orders* Job (without the Job being processed) until:
    - a further Order that has the state of the *Join Orders* Job as its end state is completed.

The *main_order_add-order* Order can now be used to increase the the total number of Orders counted by the *Join Orders* Job by 1.

- In the current example, running the *main_order_add-order* Order once will cause the number of Orders counted to reach the value set in the *required_orders* parameter (10).
    - The *Join Orders* Job will now process the *main_order* which will then proceed along the Job Chain - in this example to the *Job C* with the state 400.
- The ID of this Order has to follow one of the conventions used to identify Child Orders. Here the ID of the parent Order plus an underscore plus a string has been used.
    - Note that this string may not contain an underscore character ("_") and therefore the string "add-order" has been used.

Note that Child Orders such as the generated Orders or the manually started *main_order_add-order* Order in this example will only be recognized as such when they are started after the Parent Order has been started.

The Job Chain, Jobs and Orders

The Job Chain

The Job Chain is shown in the diagram near the top of this article and is listed in the code block below.

The y_join Job Chain

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<job_chain  title="Y Join">
    <job_chain_node  state="100" job="generate_orders" next_state="150" error_state="error"/>
    <job_chain_node  state="150" job="wait" next_state="200" error_state="error"/>
    <job_chain_node  state="200" job="job_a" next_state="join" error_state="error"/>
    <job_chain_node  state="300" job="job_b" next_state="join" error_state="error" delay="10"/>
    <job_chain_node  state="join" job="join" next_state="400" error_state="error"/>
    <job_chain_node  state="400" job="job_c" next_state="success" error_state="error"/>
    <job_chain_node  state="success"/>
    <job_chain_node  state="error"/>
</job_chain>
```

The Jobs

The *Join Orders* Job

The *Join Orders* Job basically counts incoming Jobs. This makes it significantly faster than the *Sync* Job mentioned in the Scope section above, which checks the incoming Jobs for completeness.

The configuration of the *Join Orders* Job can set using the JITL Job Wizard in JOE and is shown in following code block. Relevant for users in the following listing is the *show_join order list* parameter which may be optionally set (default is *false*). Setting this Parameter to *true* causes a list of all the orders counted by the *Join Orders* Job to be written to the Parent Order log file.

The Join Orders Job

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<job  order="yes" stop_on_error="no" idle_timeout="60" title="Join Job">
    <params >
        <param  name="show_join_order_list" value="true"/>
    </params>
    <script  language="java" java_class_path="" java_class="com.sos.jitl.join.
JobSchedulerJoinOrdersJSAdapterClass"/>
    <run_time />
</job>
```

Note that the *Join Orders* Job only counts Orders that have the state of the *Join Orders* Job as their end state (here *join*).

The generate_orders Job

The configuration of the *generate_orders* Job is shown in the next code block along with the script responsible for the generation of the Child Orders.

Note that this Job is only used to create the current example and is not required for the *Join Orders* Job itself. In a working scheduling environment, the orders for the parallel parts of the "Y" would come from other sources such as Schedules or File Order Sources.

Note also that the Job Chain path (set in line 14 of the script) must be modified to suit the actual path used.

The generate_orders Job

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<job  order="yes" stop_on_error="no">
    <script  language="java:javascript">
        <![CDATA[
function spooler_process(){

    // merge parameters from task and order
    var params = spooler_task.params;
    params.merge( spooler_task.order.params );

    // set variable
    var generate_orders = params.value( 'generate_orders' );
    var jobChain = spooler.job_chain('/test/join/y_join/y_join');

    // log parameter
    spooler_log.info( 'generate_orders = ' + generate_orders);

    // generate orders
    for (i=0;i<generate_orders;i++){
       var order = spooler.create_order();
       order.id = spooler_task.order.id + "_" + i;
       order.params.merge(spooler_task.order.params);
       order.end_state="join";
       if((i%2)==1) {
          order.state="200";
       } else {
          order.state="300";
       }
       jobChain.add_order(order);
    }

    return true;
}
        ]]>
    </script>

    <run_time />
</job>
```

The Order starts the first Job (generate) in the y_join Job Chain:

- This Job contains a script that generates the Child Orders (see lines 20 - 31 of the listing).
    - The Orders are alternated between the two branches of the Job Chain (even numbered Orders start at the Job corresponding with the Order state 150 and odd numbered Orders at the Job corresponding with the Order state 160). All Child Orders terminate at the *Join Orders* Job.
    - The total number of orders generated is determined by the *generate_orders* parameter.

The wait Job

The wait Job is configured to read the *duration_wait_job* parameter and execute a simple script (i.e. ping local host for the length of time specified in the parameter). This script causes the Job to wait for the number of seconds specified in the parameter.

The Orders

The Parent Order

The Parent Order, in this example, with ID *main_order* has the following 3 parameters:

- *required_orders*
    - This parameter (Default 10.)
    - This parameter is required for the *Join* Job.
- *duration_wait_job*
    - this is the time in seconds that the wait Job will wait before the main Order moves to the *Join Orders* Job. (Default 30 secs.)
    - The *duration_job_a* parameter is only used in the *wait* Job as part of this example and is not necessary for the functioning of the *Join Orders* Job.
- *generate_orders*
    - this is the number of Orders that are to be generated by the *generate_orders* Job. (Default 8.)
    - The *generate_orders* parameter is used in the *Generate* Job as part of this example to specify the number of Child Orders that should be generated. This parameter is not necessary for the functioning of the *Join* Job as the Jobs counted by the *Join* Job could come from any number of sources..

The main_order Order

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<order >
    <params >
        <param  name="required_orders" value="10"/>
        <param  name="duration_job_a" value="30"/>
        <param  name="generate_orders" value="8"/>
    </params>
    <run_time />
</order>
```

The Child Orders

The generated Child Orders

Have the ID of the Parent Order (*main_order*) + "_" + * where * is a string - in the current example simple numbers are used a string.

Have either:

- `<order state="200" end_state="join">` or
- `<order state="300" end_state="join">`

depending on which branch of the Job Chain they should be executed on.

The main_order_add-order Order

Has the ID of the Parent Order (*main_order*) + "_" + * where * is a string - in the current example "add-order" is used.

This Order is configured with:

- `state = 300` (The state in the Job Chain where the main_order_add-order Order starts processing. Here this is job_b) and
- `end_state = join` (The state corresponding to the *Join* Job) This means that this Order will be registered by the *Join* Job as counting towards the required orders.

The main_order_add-order Order

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<order  state="300" end_state="join">
    <run_time />
</order>
```

Logging

A parameter can be set for the *Join* Job - *show_join_order_list*. When this parameter is set to *true* the all the Child Orders counted by the join job will be listed in the Parent Order log file.

The default setting for this parameter is *false*.