**WebSphere Application Server**

# Throttling WebSphere Liberty Batch Jobs

This document can be found on the web at:
www.ibm.com/support/techdocs
Search for document number **WP102600** under the category of "White Papers"

*Version Date*: June 3, 2016
See "Document change history" on page 9 for a description of the changes in this version of the document

**IBM Software Group**
**Application and Integration Middleware Software**

Written by:
**David Follis**
IBM Poughkeepsie
845-435-5462
follis@us.ibm.com

**Don Bagwell**
IBM Advanced Technical Sales
301-240-3016
dbagwell@us.ibm.com

Many thanks go to the WebSphere Batch team for getting all this work done.

# Table of Contents

## Introduction

If you've looked through the “Creating a Job Class for WebSphere Liberty Batch” under the WP102600 Techdoc, then you already know how to use parameters specified when a job is submitted to control which Executor server the job runs inside. If you haven't read it, this might be a good time to go take a look. Everyone else, just read slowly until they catch up.

The example presented in that paper is pretty simple. We have two batch applications deployed into two servers and we use a job parameter called jobClass to control which server runs any given job. That's just great, but what happens when you have a lot more applications and more than just one parameter you want to use to control job routing? That selector is going to become very complicated and easy to mess up.

And what if you want to control how many of a particular job can run in a server at a time? There must be a way to manage that. Yes there is, and it is actually pretty easy to set up.

For the purposes of this paper we're going to stick to a somewhat simple example, but we'll show how you can configure your server in a way that scales up without becoming insanely complex.

## Our Sample Scenario

Over in WP102600 we had two applications: SleepyBatchletSample-1.0 and BonusPayout-1.0. We also had two jobClass values: A and B. In our configuration for that paper we had one executor configured to run either application when the jobClass was 'A' or not set. The other server also ran either application, but only for a jobClass of 'B'.

For this paper we want the first server to run up to five instances of the SleepyBatchlet application in jobClass 'A' and up to two instances of BonusPayout in jobClass 'A'. Over in the second server we want up to ten instances of SleepyBatchlet in jobClass 'B' and up to one hundred instances of BonusPayout in jobClass 'B'.

This is kind of what we had before with jobClass 'A' in the first server and jobClass 'B' in the second server, but we now want limits on how many jobs for each application can run in each server.

| | Server One / jobClass A | Server Two / jobClass B |
|---|---|---|
| **SleepyBatchlet** | 5 | 10 |
| **BonusPayout** | 2 | 100 |

To achieve the throttling we're going to use an existing configuration variable that is part of the activation specification we put in the server configuration. But there is a problem. We want different throttling values for different applications in the same server. How do we separate those out?

## Multiple Activation Specifications

As you grow the number of applications in any given server or you increase the complexity of the parameters being used to route work, the message selector you specify in the Activation Specification gets more and more complicated.

Imagine a server with six different applications where routing decisions for each application are made based on a couple of the parameters passed to those applications. Remember that you can route based on any job parameter, so jobs accessing one datasource might go to one server and jobs using another datasource go to another. Or routing based on an accounting string. Or some other combination of parameters.

That single selector string is going to become very very complex and quite easy to ruin when you try to make a small change. Wouldn't it be nice if you could just specify the routing parameters (message selector) for each application separately? Turns out you can.

Remember that when we configure a Liberty Server as a Batch Executor we have to include the configuration element *batchJmsExecutor*. That element points us to the queue and the activation specification elements. The queue reference gives us the queue manager and queue name (among other things) from which we get jobs to dispatch.

The activation specification contains the message selector and other configuration. Can you have more than one *batchJmsExecutor* element in a server configuration? You certainly can. For our example we will have two in each server:

```
<batchJmsExecutor activationSpecRef="batchActivationSpec1"
        queueRef="batchJobSubmissionQueue"/>

<batchJmsExecutor activationSpecRef="batchActivationSpec2"
        queueRef="batchJobSubmissionQueue"/>
```

Yes, those are exactly the same except the activation specification we point to in each one is slightly different (the '1' and '2' at the end of the reference name). You can define as many of these elements as you need. For each one you will, of course, have to have a matching activation specification definition. They will start out like this:

```
<jmsActivationSpec id="batchActivationSpec1" >
```

and

```
<jmsActivationSpec id="batchActivationSpec2" >
```

And each one is exactly the same except for the message selector. In Server one the selectors look like this:

```
messageSelector="com_ibm_ws_batch_applicationName =
        'SleepyBatchletSample-1.0'
        AND (jobClass = 'A' OR jobClass IS NULL)"
```

and

```
messageSelector="com_ibm_ws_batch_applicationName = 'BonusPayout-1.0'
        AND (jobClass = 'A' OR jobClass IS NULL)"
```

And for Server two we have two batchJmsExecutor elements that point to the same queueRef but different activation specifications. And the message selectors in those two are like this:

```
messageSelector="com_ibm_ws_batch_applicationName =
        'SleepyBatchletSample-1.0' AND jobClass = 'B'"
```

and

```
messageSelector="com_ibm_ws_batch_applicationName = 'BonusPayout-1.0'
        AND jobClass = 'B'"
```

Those are pretty simple and straight-forward, right? And you could easily just add more batchJmsExecutor elements with more activation specifications for other applications or other specific routing rules.

## What about Throttling?

This actually isn't a new thing. It isn't even a batch thing. This is just part of the activation specification configuration. How you do it depends on whether you are using MQ or the integrated messaging engine (SIBus). What you need to do is tell the activation specification how many concurrent messages to allow.

The onMessage method (part of the batch container code) will be driven for each message and isn't counted as 'complete' until the method returns. And the batch container is coded to wait for the job to complete before returning. I should point out that the actual job runs on a different thread than the onMessage execution so you will be using two threads in the server per job you allow to run. Fortunately the Liberty executor thread pool expands automatically and will grow as needed to handle the number of jobs you allow to run.

What are the configuration parameters? For MQ you specify the *maxPoolDepth*. For the integrated messaging engine you specify *maxConcurrency*. The default for *maxPoolDepth* is ten and the default for *maxConcurrency* is five.

If we are using MQ, then the full activation specification definition for SleepyBatchlet in Server One could look like this:

```
<jmsActivationSpec id="batchActivationSpec1" >
  <properties.wmqJms
    destinationRef="batchJobSubmissionQueue"
    messageSelector="com_ibm_ws_batch_applicationName =
         'SleepyBatchletSample-1.0' AND (jobClass = 'A' OR
         jobClass IS NULL)"
    maxPoolDepth="5"
    transportType="CLIENT"
    channel="SYSTEM.DEF.SVRCONN"
    destinationType="javax.jms.Queue"
    queueManager="MQS1"
    hostName="wg31.washington.ibm.com"
    port="1414">
  </properties.wmqJms>
</jmsActivationSpec>
```

And that's all there is to it. If you have multiple applications or complex routing property rules you can set up separate batchJmsExecutor elements for each one and specify its rules in a separate jmsActivationSpec element. And having done that you can separately throttle the number of jobs for that message selector using either *maxPoolDepth* or *maxConcurrency* depending on which messaging engine you are using.

## A word about dots vs. underscores

My first pass at exploring this capability just involved SleepyBatchlet. I thought that it would be fun to throttle based on how long the batchlet was going to sleep. The sleep time is based on a job parameter called *sleep.time.seconds*.

I tried to configure the server with two activation specifications, both for SleepyBatchlet. One was for long running (sleeping) batchlets so I wanted to limit it to one job at a time if the sleep time was greater than 60 seconds. The other specification was for short(er) batchlets and would allow up to ten jobs that slept less than 60 seconds.

And it didn't work. The problem is that you can't use sleep.time.seconds as a routing parameter because MQ doesn't allow dots in the property names. There is a note in the Knowledge Center about this, but you might have missed it.

The rule is that any message properties have to start with a character for which Character.isJavaLetter( ) returns true and can consist only of characters for which Character.isJavaLetterOrDigit returns true. Which means no dots.

## Document change history

Check the date in the footer of the document for the version of the document.

| | |
|---|---|
| *June 03 2016* | Initial Version |

**End of WP102600**