WebSphere Application Server

WebSphere Liberty Batch: Job Purge Processing

This document can be found on the web at:
www.ibm.com/support/techdocs
Search for document number **WP102712** under the category of "White Papers"

Version Date: December 11, 2019
See "Document change history" on page 11 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 whole batch team, but especially
Chris Gianfrancesco for making this easier to write.

© 2016, IBM Software Group, Application and Integration Middleware Software, Poughkeepsie, NY

Contents

Introduction 4
Purge and the JSR-352 Specification 4
The Two Parts of Purge 5
Which Command Line Interface Can You Use? 5
Using the Java CLI 6
Purging Single Jobs with the CLI 6
Purging Multiple Jobs with the CLI 7
Purging Just the Repository 9
Using the REST Interface Directly 9
Purging Jobs That Never Ran 10
Document change history 11

© 2016, IBM Software Group, Application and Integration Middleware Software, Poughkeepsie, NY

Introduction

Running batch applications in any environment, using any technology, produces artifacts of the execution of the batch processing. There might be some records of when the job ran and whether or not it was successful. The batch job might have produced a log with messages chronicling its execution. Over time as more and more jobs are executed all this tends to pile up, just like those cute photos of your dog or cat on your phone. At some point you just have to clean it up to make space and declutter.

Unlike those photos, your emotions generally don't get involved in deciding whether to purge the output of a job or not. And often those jobs run on a regular schedule and so it might make sense to have a similar regular schedule that gets rid of the output. But you may also have some manually submitted jobs that produce output an automated purge process might not know about and you need to clean that up too.

What tools does WebSphere Liberty Batch make available to help you with this? Just how does purge processing actually work? Do you need to care about that? Are there any complexities or special cases you need to care about?

Hopefully we'll answer all of these questions and more. Let's start..

Purge and the JSR-352 Specification

WebSphere Liberty Batch is just the JSR-352 Java Batch standard at its heart. So before we get too deep into our options we should look to the specification to see what options it gives us regarding purging of job data.

Not a word.

Go look. Search the specification for 'purge'. You won't find a single reference. Why is that? Well the specification doesn't mention job logs so it has no reason to talk about how you would get rid of them. That's a WebSphere Liberty Batch extension to the specification.

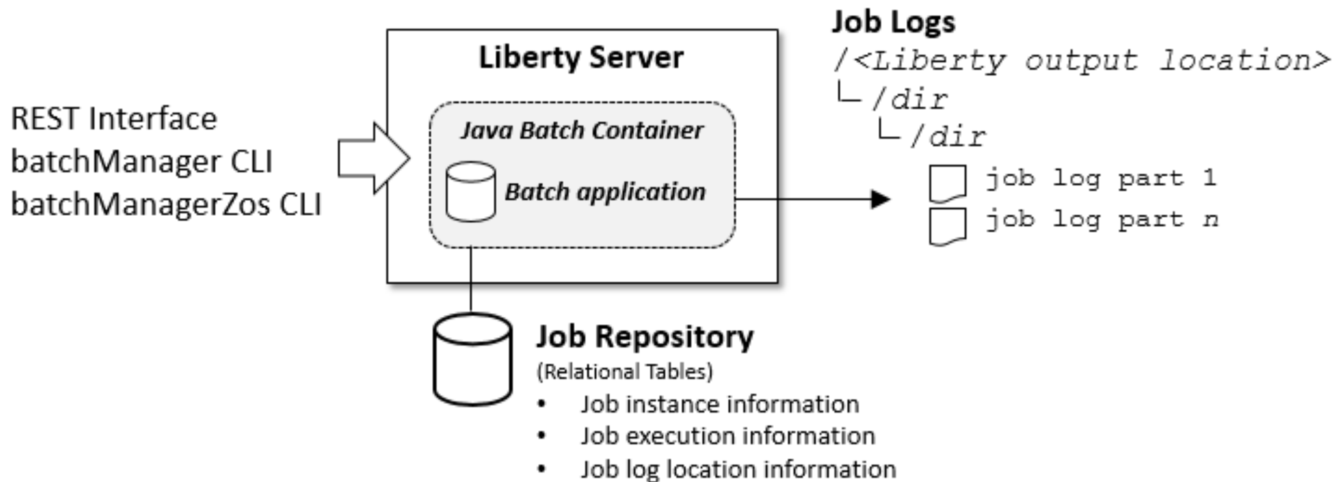
And the specification also doesn't detail the implementation of the Job Repository (where information about jobs is kept) so it doesn't detail how you would go about getting stuff out of the Job Repository. It is assumed to be part of however you implement the repository.

So there is no help for us there. Let's move on then.

© 2016, IBM Software Group, Application and Integration Middleware Software, Poughkeepsie, NY

The Two Parts of Purge

Before we get into how you purge a job, we should look in more detail at what actually happens as part of purge processing.



First of all, we need to get rid of all the job log pieces. When the job ran any messages issued by the application or by the batch container may have been written into the file system for the server that ran the job. If there were a lot of messages then it may have wrapped into several files. If the job had partitions or split/flows then those will also create separate job log files.

Getting rid of those files requires being able to access the server where the job ran and asking it to clean up the relevant files in its file system. If the server is down or otherwise inaccessible then this part of purge processing isn't going to work.

The other half of purge processing is cleaning up various rows from the Job Repository tables. All sorts of information about what happened to a job (including all of its Executions) are kept in tables in the database that contains the Job Repository. Purge processing needs to clean up all the relevant rows from various tables in the repository.

One of the things we keep in the Job Repository is where the job logs live. So that means we need to clean up the Job Log files before we clean up the Repository. And if for some reason we can't clean up the Job Logs, we don't want to clean up the Repository or we'll forget where the logs are and we can't go back and get them later. So Job Log cleanup comes first, followed by Repository clean up if Job Log cleanup was successful.

Which Command Line Interface Can You Use?

WebSphere Liberty Batch provides a REST interface and two command line interfaces (CLIs). One CLI is written in Java and exploits the REST interface. The other CLI is written in C and runs on z/OS, exploiting Liberty's Optimized Local Adapter (WOLA) interface.

© 2016, IBM Software Group, Application and Integration Middleware Software, Poughkeepsie, NY

The native CLI (batchManagerZos) exists because starting a Java program to run the CLI to submit a Java batch job inside Liberty is more overhead that you want on a platform sensitive to excess CPU usage (for cost reasons). Thus the intention of providing the native CLI was that it would be used to submit jobs into a Liberty Batch server.

Both CLIs are able to purge jobs in a single server environment. However, in a multi-server configuration with dispatchers and executors you should use the Java CLI because the REST interface allows purge requests to be properly forwarded to executors to purge any job logs.

Using the Java CLI

Before we start purging jobs with the Java Command Line Interface we should cover some basics. First of all, issue the command:

```
batchManager help purge
```

That will give you the help and the complete syntax possibilities for purging. You will notice that there are three required parameters: user, password, and batchManager. The first two are, of course, the userid and password to authorize access to the server you are connecting to and to determine your authorization to purge jobs there.

The `batchManager` parameter provides the host:port (or a list of host:port values) of the target server. This could be a stand alone server or a dispatcher in a multi-server configuration. In a multi-server configuration you may have multiple dispatcher servers in which case you might want to list all of their host:port combinations as any dispatcher can purge jobs from that environment.

Some other parameters are common to all batchManager CLI processing and we won't spend any time on them here: controlPropertiesFile, trustSslCertificates, and httpTimeout.

Purging Single Jobs with the CLI

This is, more or less, the easy case. You submitted a job, it ran, and now you want to purge it. To do that you need to know the job instance ID. Each job instance gets a unique identifier and you can issue a `batchManger` purge command specify the `jobInstanceId` parameter to purge it.

For example:

```
batchManager purge --user=XXX --password=YYY
          --batchManager=my.host:8080 --jobInstanceID=1234
```

That seems pretty straight forward. What could go wrong? Well first of all you need to remember that job instance IDs are unique within the set of servers sharing a Job Repository. If you have a complex topology and not just one server that runs all your jobs then you need to make sure you are talking to the right server when you purge a job by Job Instance ID. Another server with a different Job Repository might still have a job using that ID, but it isn't the job you wanted to purge. Be careful.

Besides the scope of the ID you also need to think about how you will know what the ID actually is. When you submit a job part of the set of messages that get issued by either CLI includes one that has the Job Instance ID. Remember that this is different from a Job Execution ID. When you submit a job it gets both an Instance and an Execution ID. If it fails, you may be able to restart it getting a new Execution ID. Purge processing always uses the Instance ID. But Instance and Execution IDs are just numbers and it may be that an Execution ID could also be a valid Instance ID for a different job. Again, be careful.

Purging Multiple Jobs with the CLI

If you want to purge more than one job and you know the Job Instance IDs then you can save time and overhead by supplying `batchManager` with a list (or range) of ID values on the `jobInstanceId` parameter.

Some examples (with user, password, and batchManager keywords left out):

```
batchManager purge –-jobInstanceID=12,14,15

batchManager purge –-jobInstanceID=”<100”

batchManager purge –-jobInstanceID=1000:1100
```

The first example purges jobs with those specific Job Instance IDs. The second example purges any job with an instance ID less than 100. Note the quotation marks around the string. The parameter has to be a string, you can’t specify `–-jobInstanceID<100`. The third example purges all jobs with instance ID values between 1000 and 1100.

You can also use other `batchManager` purge parameters to control what jobs get purged without needing to know the Job Instance ID values.

For example, you can purge based on exitStatus or instanceState. The instanceState is one of COMPLETED, FAILED, or STOPPED. The exitStatus is a string set by the application itself. The possible values depend on what the application sets the status to.

More examples (again, leaving out user, password, and batchManager keywords):

```
batchManager purge –instanceState=FAILED

batchManager purge –exitStatus=12
```

```
batchManager purge –exitStatus=SUCCESS*
```

The first example purges jobs based on their batch status value. Only jobs that are COMPLETED, FAILED, or STOPPED can be purged. This example, obviously, purges all failed jobs.

The second example purges based on the exit status value of the job. The exit status value by default is the batch status (e.g. instance state) of the job, but the application can set it to any string value it likes. In this case we know that some applications set an exit status value of 12 and we want to purge any jobs that completed that way.

But the exit status can be any string so you might have applications setting non-numeric values like “SUCCESS”. In this case we're purging those jobs and we have a trailing asterisk to wild card any other text that might follow that word.

You can combine keywords too. This example (again, skipping user, password, and batchManager) purges any FAILED jobs with instance IDs between 1000 and 2000:

```
batchManager purge –jobInstanceID=1000:2000 –instanceState=FAILED
```

You can also purge based on creation time. You can specify a specific date to purge jobs created on that day. Or you can specify a range of dates to purge jobs created between those dates. You can also purge jobs relative to a number of days ago. For example you can purge jobs with a create time greater than 3 days ago. Remember when using relative purges that time runs forward. So create times greater than 3 days ago are create times MORE RECENTLY than three days ago. That is, a job where the value of the create time date is bigger than a date value of three days ago.

While these are all useful, you can also combine these together to do more useful things like purging all FAILED jobs that were created on a specific day. Or you can purge all STOPPED jobs in a Job Instance ID range.

Now it is possible that your filters might result in the need to purge a lot of jobs. You might want to just take it in pieces and purge a subset of the jobs at a time. To help with that, `batchManager` supports the `page` and `pageSize` keywords. The `pageSize` defines how many jobs will be purged at a time from all the jobs that match your filters. The `page` value defines which set will be purged.

It helps to think of the processing as building a list of all the jobs that match your criteria and then dividing it up into pages of `pageSize` jobs. Then you tell it which page (the first, the twelfth, the twentieth) that you want to purge. There is an assumption that the list doesn't really change between calls. But, of course, once you have purged a set of jobs, that set no longer appears in the list. So if you purge the first page of jobs those jobs are no longer in the list. On the next call you really want to purge the first page again since the new first page is the jobs that used to be in second page.

Confused? Be safe, use the `batchManager listJobs` operation which supports all the

same keywords as purge, including `page` and `pageSize`. This operation will show you all the jobs that would be purged by the equivalent purge command. Even if you build purge processing into an automated script it might not be a bad idea to capture the `listJobs` results before doing a purge just to have some record of what happened in case something goes awry.

Purging Just the Repository

There is one more purge option we haven't talked about. That's the `purgeJobStoreOnly` option. This option tells the server to just clean up information about the job (or jobs) in the Job Repository and not to worry about cleaning up the Job Logs.

Why would you want to do that? There are a number of possibilities. Perhaps you have some other automated process that cleans up the server file system and removes the logs for you. Perhaps you aren't even keeping the job logs around.

Or you may simply have the log files deleted by some frequent automated process because the logs get sent to the CLI and kept with the output of the job or script that ran the CLI. Or perhaps you are in a cloud environment where the server file systems come and go and the job log files don't survive.

Regardless of why, you might find it easier to use the `purgeJobStoreOnly` option to just clean up the Job Repository and handle the log files another way.

Using the REST Interface Directly

What if none of this does quite what you want? You can always build your own REST client that works exactly the way you need it. For example, suppose you want to purge all the jobs for a particular application that used a specific JSL file. You could use the REST API to return a list of ALL the jobs in the Job Repository and then examine the results looking for ones that match your criteria. You could build up a list and then purge specifying a list of Job Instance ID values.

You might be able to use some of the supported criteria (such as `instanceState` or `createTime`) to narrow down the list of jobs returned, but you would still have your own code to sift out the jobs you wanted to purge.

The URL for purge and for list: `/ibm/api/batch/jobinstances`. Newer versions inserting, for example, a ‘`v3’` between ‘`batch’` and ‘`jobinstances’`. The newer versions support additional filters. The difference between getting information about a job, or list of jobs, and purging that job or jobs is the choice of Verb: GET will get information and DELETE will purge.

© 2016, IBM Software Group, Application and Integration Middleware Software, Poughkeepsie, NY

Purging Jobs That Never Ran

In a dispatcher/executor configuration jobs are submitted to the dispatcher, a message representing the job is put on a queue, and an executor picks up the message and runs the job. Which executor picks up which messages is controlled by the message selector string that is part of the executor's Activation Specification configuration.

Let's consider an example. Suppose you have a configuration with two executor servers pulling job messages from a queue. The message selector is based on a message property called 'JobClass'. One server takes messages where JobClass=A and the other where JobClass=B. When a job is submitted into this environment a job parameter of 'JobClass' is specified with a value of 'A' or 'B' to control where the job runs.

But what if somebody makes a mistake and submits a job with a job parameter of 'BobClass=A'. Nothing is looking for a message with a property of 'BobClass' and so no executor will pick up the job. It will just sit there. Forever.

How do you clean this up? Well, you could look at the properties of the message on the queue (MQExplorer will help you do this) or look at the job parameters the job was submitted with to understand why the job isn't running. Then you could, just for this one job, configure a server to accept jobs with 'BobClass=A' and let it run. You'd remove the configuration after the job ran, unless your user can't seem to type 'Job' without hitting a 'B' instead.

But what if you just want to get rid of it? Can you purge it? Well, no. To purge a job it has to have completed in some way. This job hasn't even started yet, so it can't be purged. First you have to stop it. The REST API and Command Line Interfaces support a 'stop' operation. This will move the job into a 'stopped' state. And a stopped job is purgeable.

That will remove the job from the Job Repository, and there are no job logs to clean up because the job never started executing so it never created any.

But what about the message on the queue? That will remain there. If, somehow, a batch executor server does get defined that can pick up the job, it won't find matching information in the Job Repository and the message will be discarded.

If you'd like to have messages like this automatically removed, MQ supports setting a CAPEXPRTY value on the queue that defines the amount of time (in tenths of a second) that a message can be on a queue before it expires. You could set this to some fairly large value (maybe a week?). The EXPRYINT value determines how often MQ wakes up to look for expired messages.

Be careful with this. It might be that jobs regularly get submitted with JobClass=Z and the servers that process those jobs only get started on alternate Tuesdays after midnight. You don't want to accidentally remove the messages for perfectly good jobs that are just waiting for automation to start up their executor. Maybe set CAPEXPRTY to a couple of weeks (the max value is 999,999,999 which is…um…just over 3 years…I think).

Document change history

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

June 9,2017	Initial Version
June 13, 2017	Add document number
December 11, 2019	Added information about cleaning up jobs that never ran

End of WP102712

© 2016, IBM Software Group, Application and Integration Middleware Software, Poughkeepsie, NY