

WebSphere Application Server

# WebSphere Liberty Batch: z/OS Dataset Contention

This document can be found on the web at:  
[www.ibm.com/support/techdocs](http://www.ibm.com/support/techdocs)  
Search for document number **WP102667** under the category of "White Papers"

*Version Date:* September 26, 2016  
See "Document change history" on page 16 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](mailto:follis@us.ibm.com)

**Don Bagwell**  
IBM Advanced Technical Sales  
301-240-3016  
[dbagwell@us.ibm.com](mailto:dbagwell@us.ibm.com)

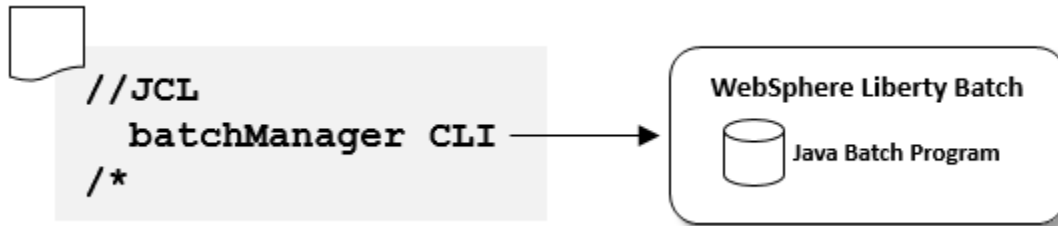
Many thanks go to the WebSphere Batch team for getting  
all this work done.

## Table of Contents

<b>Introduction</b> .....	<b>4</b>
<b>Problems? What Problems?</b> .....	<b>4</b>
<b>The Rules</b> .....	<b>6</b>
The Final Word.....	6
DISP to ENQ.....	6
Seize and Hold.....	6
Learning to Let Go: DSENQSHR.....	8
DD vs. Dynamic Allocation.....	9
You Don't Really Want To Do That.....	10
Jobs and JobGroups.....	11
<b>Scenarios and Solutions</b> .....	<b>12</b>
Using an Existing Dataset.....	12
Creating a New Dataset in JCL and using it Liberty Batch .....	13
Creating a new Dataset in JCL and using it later in the JCL itself and in Liberty Batch .....	13
Creating a Dataset in Liberty Batch to use later in the JCL.....	14
Using a Dataset DISP=SHR in JCL but exclusively from Liberty Batch .....	14
Dynamically Allocating a Dataset from the JCL job and using it in Liberty Batch .....	15
Using a Dataset from Liberty Batch and JCL – Error Handling.....	15
<b>Conclusion</b> .....	<b>15</b>
<b>Document change history</b> .....	<b>16</b>

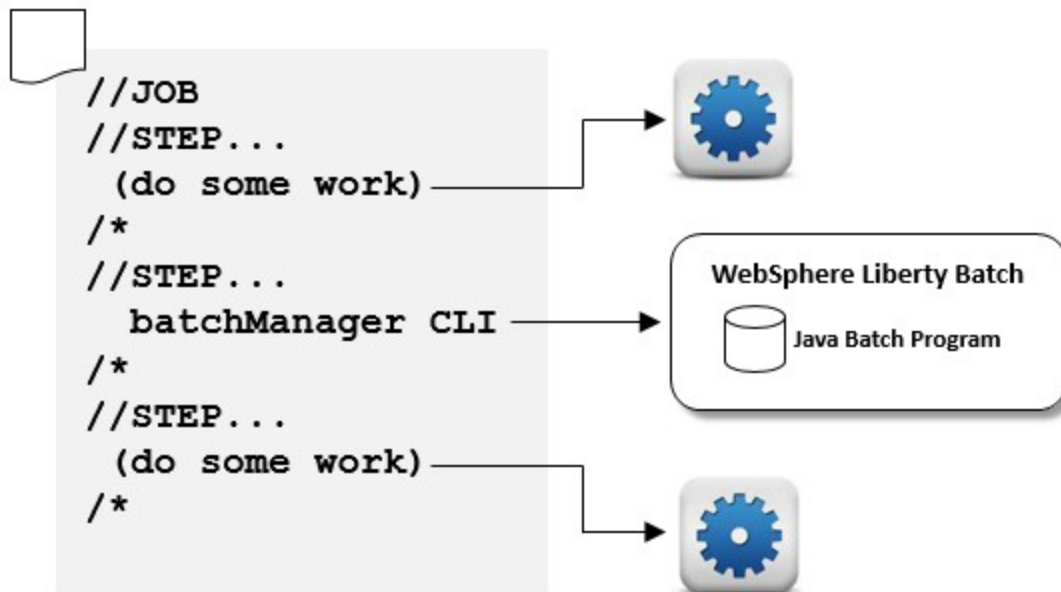
## Introduction

One of the ways to integrate WebSphere Liberty Batch jobs with traditional batch jobs on z/OS is to use standard JCL to invoke the batchManagerZos Command Line Interface (CLI). The CLI will connect to the specified Liberty server and your JSR-352 compliant Java Batch job will run in Liberty's Batch Container.



The JCL serves to integrate the Java Batch job into a more traditional batch environment that can be managed by traditional Enterprise Schedulers. If you do only this in the JCL then the traditional batch job serves as a proxy for the Java Batch job.

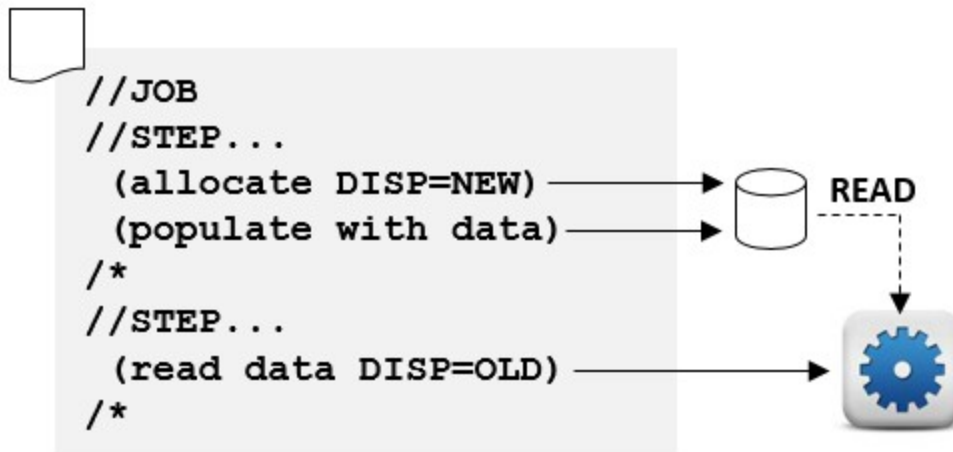
However, you might want to integrate a Java Batch job as a step inside a traditional JCL batch job that already does a bunch of other things. In that case you may have to be careful about how the processing in the JCL step that is the Java Batch job interacts with the processing in other steps of the JCL.



In this paper we'll take a look at some of the intricacies of dataset allocation on z/OS that can become a factor in creating multi-step JCL jobs that include steps that drive Liberty Batch jobs.

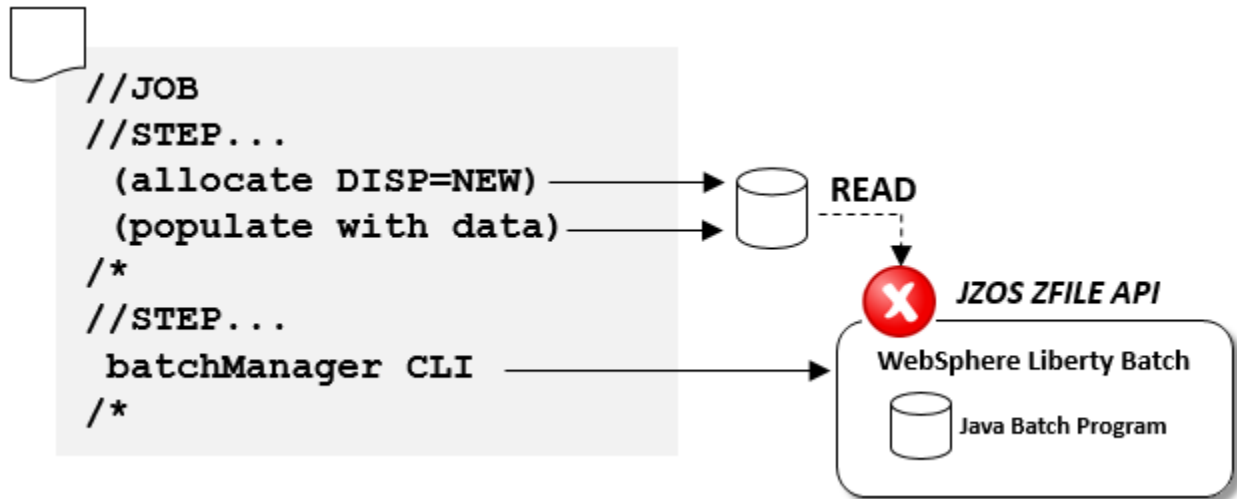
## Problems? What Problems?

To give you a hint about where we are going let's consider a simple scenario. Suppose you have a simple two step JCL job. The first step allocates a new dataset and writes content into it (from somewhere..doesn't matter where). The second step reads the data from the first step and does something with it (doesn't matter what).



In traditional JCL you might have a DD card in the first step that allocates the dataset DISP=NEW and in the second step you might have a similar DD card but specify DISP=OLD since the dataset will exist by the time we get to the second step. (You could also do this with a temporary dataset that is passed to the second step, but let's assume we want to keep the dataset around after the job for some reason).

Now we want to convert the second step into a single-step Java Batch job that runs inside WebSphere Liberty. The second step will no longer execute our application program but will instead call batchManagerZos to submit the Liberty Batch job. We no longer need the DISP=OLD DD statement in the second step because we aren't using the dataset in the code running in that step.



Instead over inside Liberty our Java Batch application will use the JZOS ZFILE API to allocate and open the dataset (presumably it knows the name or you passed the name as a job parameter when it was submitted).

But it doesn't work. Why not? Because the JCL job that created the dataset holds an exclusive ENQ on the dataset name and the Liberty Batch job can't open the dataset when another address space has it locked.

Oh dear.

## The Rules

Before we try to solve that problem we should be sure we understand the rules. In this section we will review some of the rules of dataset allocation and some features of JES and JCL that might help us out.

## The Final Word

This isn't it. The final word on the rules of handling dataset allocation is the official documentation about z/OS and JES2/JES3. When in doubt, always refer to the official documentation. What I'm trying to do here is just give a quick overview of the basics to make it easier to understand the problems and potential solutions. I'm glossing over a lot of detail.

For the most part, the documentation I'm leaning on for this section can be found in the JCL User's Guide (SA23-1386) in Chapter 13, "Data set resources – description".

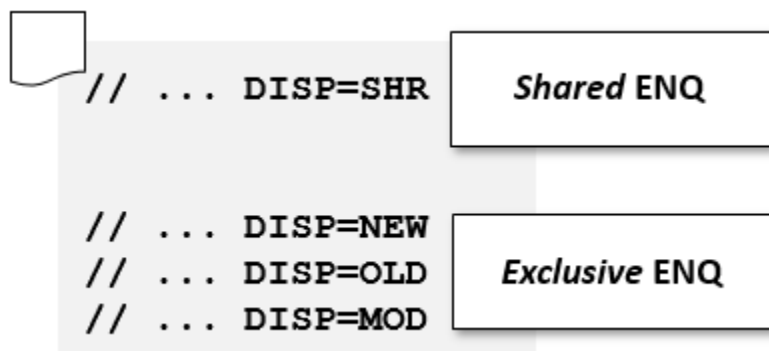
## DISP to ENQ

When we talk about JCL and dataset management we tend to be a bit loose in our terminology. We say things like "allocates a dataset exclusive". To really dig into our problems sharing datasets between Liberty Batch and traditional JCL jobs we need to look a bit deeper to understand how this actually works and to do that we have to be clear on our terminology.

The first thing to do is look at the actual text of the JCL DD statement and determine how that is reflected in the GRS ENQ that is used to synchronize access to the dataset

**Aside:** I used the word 'synchronize' where often z/OS folks might say 'serialize' because to Java developers 'serialize' has nothing to do with locking. As we integrate the traditional batch world with the Java world this is one of those words you have to be careful with..everybody knows what it means, but it means different things to different people.

How do we map a DISP value in JCL to an ENQ? It is pretty easy really. If you have a DD statement that specifies DISP=SHR then it maps to a *shared* ENQ. Anything else (NEW, OLD, MOD) will map to an *exclusive* ENQ.



But when do we get the ENQ and how does that relate to 'allocation'.

## Seize and Hold

There are two parts to what we tend to generically refer to as 'allocation' for a JCL job. The first part is getting locks on the resources needed by the job (the ENQs) and the second part is actually opening the dataset to read or write. JES (either one) wants to try to make sure that once it starts

running a job it can continue to run to completion without getting tangled up with other jobs using the same datasets.

To that end it divides datasets up into two categories: permanent and other. Permanent datasets are those whose life extends beyond the job. Other datasets are things like temporary datasets or an instance of a GDG dataset which would just be used by this job.

JES is mostly worried about permanent datasets. It wants to be sure that your job can get to all those datasets that might be used by other jobs before it even starts letting your job run. To that end, it looks through the JCL and figures out what level of ENQ (shared or exclusive) it will need for every dataset and it obtains those ENQs (or tries to) before it lets the job start.

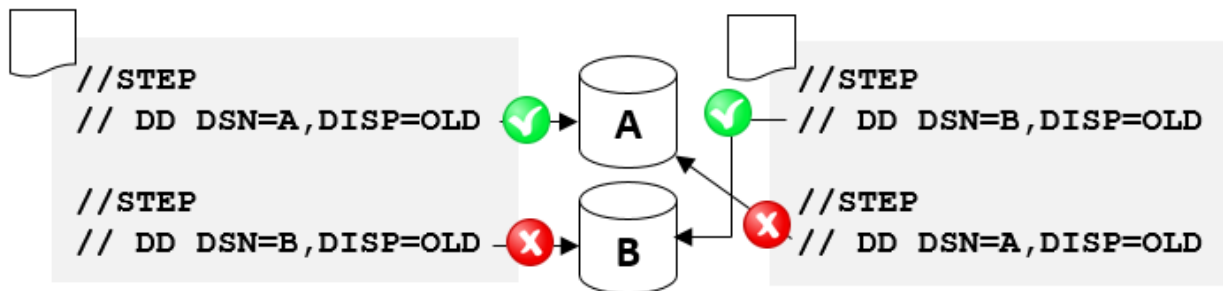
### Permanent Datasets



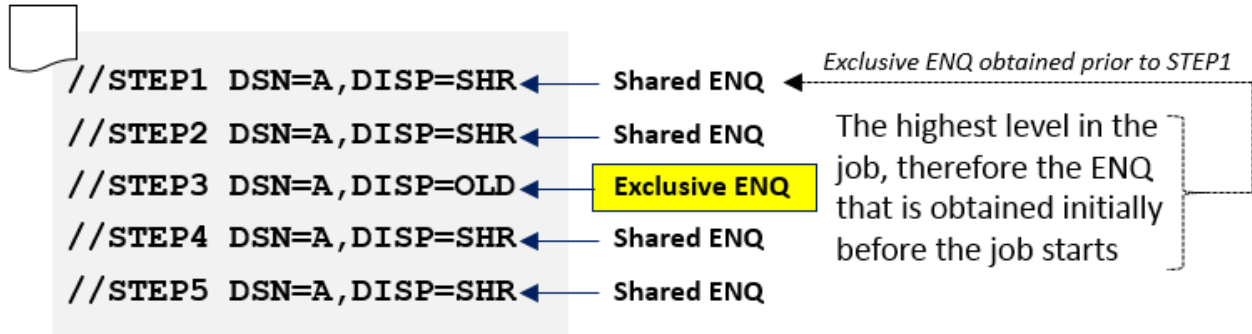
If it can't get an ENQ it needs, then you get those "JOB xxx WAITING FOR DATASETS" messages that let you know things are being held up. But it is considered better to hold up starting a job waiting for a dataset than to let the job start running and then get stuck waiting for an ENQ. You can imagine two jobs with step-scoped ENQs going after the same sets of datasets getting tangled up in a deadlock where neither job can proceed.

Let's look at the example below. On the left we have a job that allocates dataset "A" exclusively in the first step and then needs dataset "B" exclusively in the second step. The job on the right does the same thing, but in the reverse order. If JES allowed both jobs to start running, getting the ENQs as it goes, then the jobs would deadlock.

If you dynamically allocate datasets instead of using DD statements in the JCL then JES can't help you avoid deadlocks. Eventually the SMF wait time will be used to break this up with an abend.



But what if different steps need the same dataset but with different levels (remember our example from earlier where we need it DISP=OLD, or exclusive, in one step but later DISP=SHR, or shared, in another step)? The only safe thing for JES to do is obtain the highest level of ENQ it needs for the whole job and hang onto it until the last step that references that dataset.



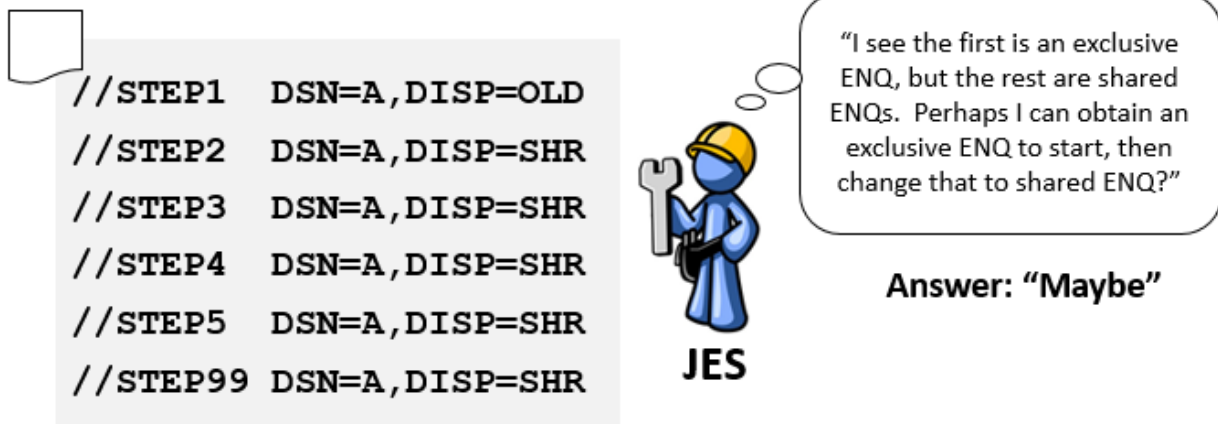
That means that if any step in the job has a DD that will need an exclusive ENQ, then that exclusive ENQ is obtained before the job starts and held for the entire job (except see the next section).

This is a key factor in the problems we're trying to solve sharing datasets between JCL batch jobs and Liberty Batch jobs running at the same time.

### Learning to Let Go: DSENQSHR

From the previous section you can see that it makes sense for JES to get an exclusive ENQ for a dataset that a job needs DISP=OLD before it even lets the job start, even though the step that uses that dataset may be a long way into the job. And that even if it needs the dataset DISP=SHR early on, it should get the ENQ exclusive from the start because it will eventually need the dataset DISP=OLD.

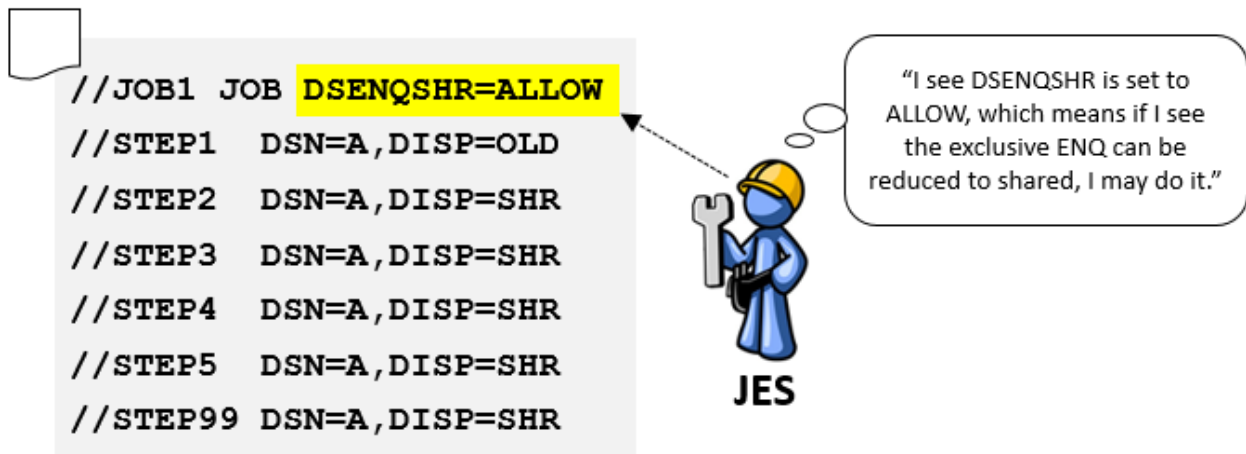
But what if it is the other way around? What if the job needs the dataset exclusively early on, but just needs it shared later in the job? Could we reasonably drop the ENQ from exclusive to shared after the exclusive step completes? Maybe.



JES gives you the option to let that happen. There is a JOB card keyword called DSENQSHR that you can set to one of three values. The first value is 'DISALLOW' which is the default and tells JES to work the way we have been describing. It means to disallow lowering the level of the ENQ.

The more interesting value is, of course, 'ALLOW' which tells JES to lower the level of the ENQ from exclusive to shared if it sees that the later steps in the job need the ENQ at a lower level. This could be a huge help to us, as we will see later.



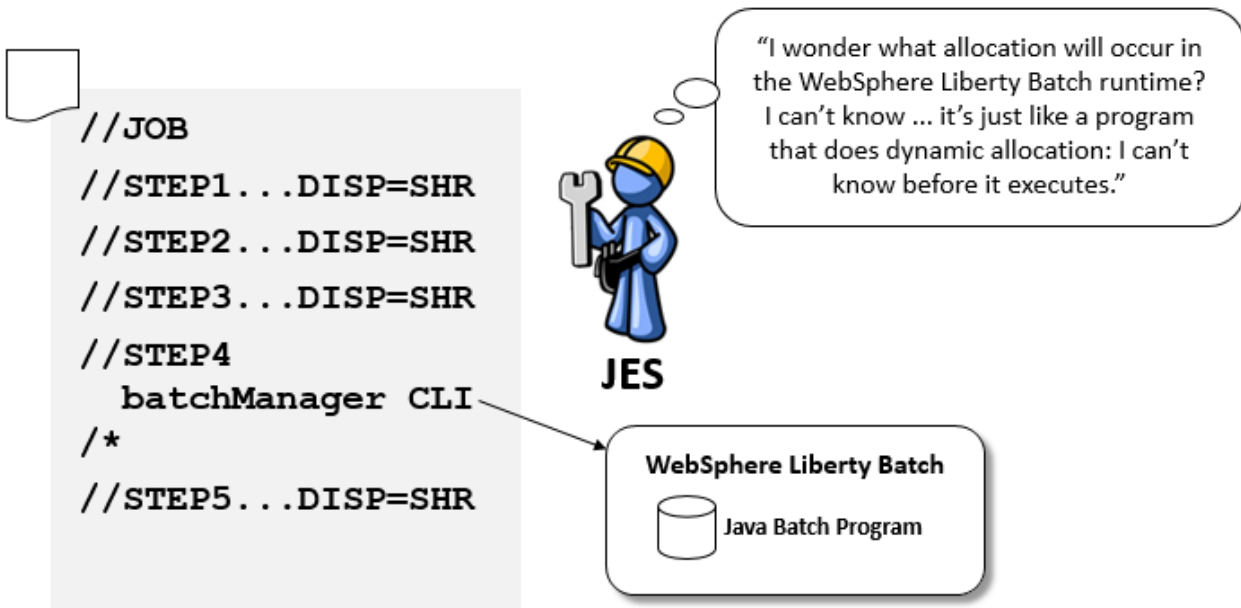


For completeness, there is also a third value, 'USEJC', which means that lowering the ENQ level might be allowed depending on how the JES jobclass is configured.

### DD vs. Dynamic Allocation

Everything we have talked about so far involves DD statements in JCL. What about applications that use Dynamic Allocation to allocate datasets on the fly during execution of the step?

Well pretty obviously JES has no idea you are going to do that and so it can't get the ENQ for you ahead of time. And, since our Liberty Batch jobs are running inside an already-executing Liberty server that is how any datasets are allocated inside a Liberty Batch job. JES has no visibility to what a Liberty Batch job will do with datasets.



As we go through various scenarios we'll see that doing the allocation yourself and not specifying things in JCL can be a solution. But if you take that path, remember that you are giving up the deadlock avoidance that JES is providing by getting the ENQs early. You have to take care not to create batch jobs that can deadlock each other over dataset usage. That might be hard to manage in a large organization with lots of applications being developed around the same data.


## You Don't Really Want To Do That

In this section we will talk about jobs that never run because the system thinks it is smarter than you are. The function I am talking about is a part of JES3 and is available from vendor products on JES2.

During the process of submitting a job the JCL is scanned looking for DD statements coded in a way that indicates the dataset already exists (e.g. OLD, SHR, MOD). For anything that is found like that, a check is made to be sure the dataset really does already exist. If it doesn't, and there is no sign in the JCL of a way that dataset might get created (e.g. DISP=NEW) then it seems unlikely the job could work and the submit will fail.

Remember that all the talking we did about ENQs doesn't have anything to do with the actual file on the disk. It is just ENQs trying to coordinate the synchronization of dataset across different jobs. There is no guarantee that just because you got the ENQ that the dataset will actually exist and get allocated when you need it.

```
//JOB
//STEP1 . . .DSN=A,DISP=OLD
//STEP2 . . .DSN=A,DISP=SHR
//STEP3 . . .DSN=A,DISP=OLD
:
//STEP99 . . .DSN=A,DISP=OLD
```



**JES3**  
Or vendor products  
with JES2

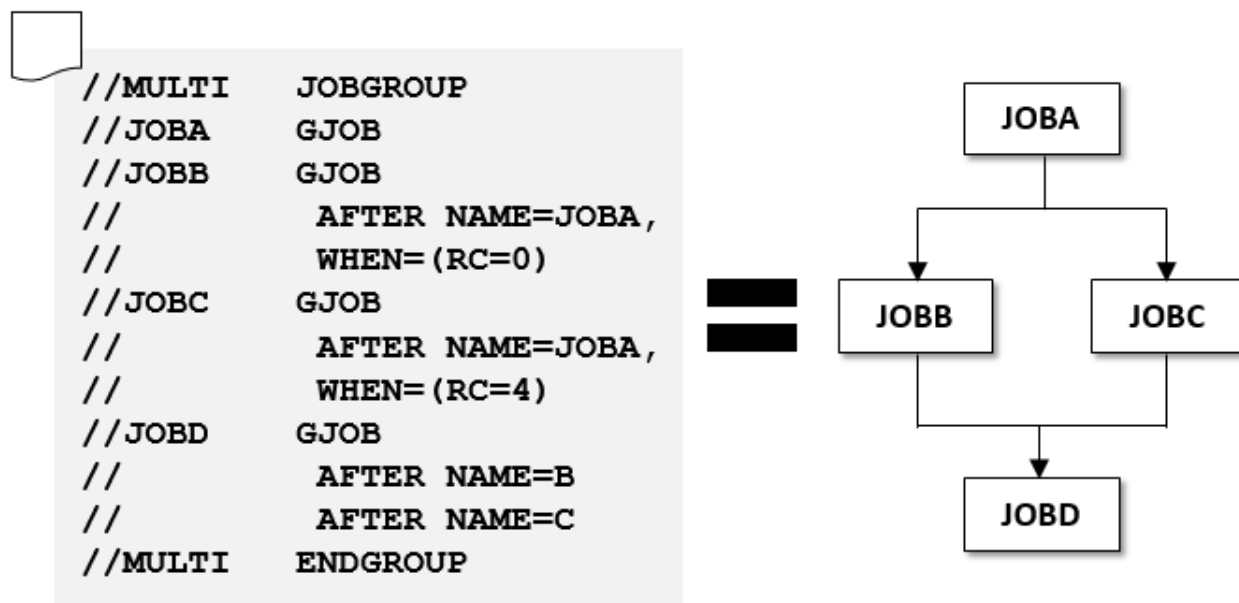
“All your dispositions assume the data set exists, but I see it does not. I don't see any DISP=NEW, so I know you're not creating it. I conclude this job is not likely to complete, so I won't allow it to start

This scanning of the JCL and preventing 'unworkable' jobs from being submitted sounds pretty cool, but we'll see a scenario where we are blending JCL and Liberty Batch into one thing where it gets us in trouble. And we'll find a somewhat sleazy way around it.

## Jobs and JobGroups

This is another capability of JES2, like DSENQSHR, that will help us out as we work through problem scenarios. A brief overview seemed useful at this point.

JobGroups are a new capability for JES2 in z/OS 2.2. You create new JCL that defines a JOBGROUP as a set of JOBS. There are an abundance of options but essentially you are defining to JES a flow of control between separate jobs. For example, JOBB and JOBC can only start after JOBA has completed, and then JOBD can only start after JOBB and JOBC have finished:



You can submit jobs JOBB and JOBC and JES won't let them start until JOBA has completed. There are all sorts of more clever ways to manage the flow.

How will this help us? Well, the scope of the ENQs we are concerned with is a job. Sometimes it might help to break a job up into smaller jobs to break up the ENQ scope. In the past that would mean adding steps to the end of one of the smaller jobs to submit the next smaller job and so forth to construct the chain using your own code. The ability to really keep the whole thing together as one large JobGroup makes it much easier to do that.

However, because the proper ENQs are still being obtained at the JOB level (not the JobGroup) you could end up with individual Jobs being held up waiting for datasets where before the entire larger Job ran to completion. Other jobs accessing the same datasets could find their way into the middle of your multi-Job JobGroup where before they could not. If that other job is actually a Liberty Batch job and that access is actually what you want, then that's great. But it might not be, so be careful.

---

## Scenarios and Solutions

### Using an Existing Dataset

#### The problem

In this scenario we have an existing dataset that will be specified in DD statements in JCL and allocated by Java code running inside a Liberty Batch job launched by another step in that same JCL.

If the JCL specifies DISP=SHR and the Java code can allocate the dataset shared then there is no problem. But if either one of them needs it exclusively for some reason then we have a problem.

Suppose we have a step early in the JCL that references the dataset in a DD coded DISP=OLD. We want to verify the dataset exists and the contents are useable. We originally coded it this way so that a later step could be sure the data had not changed since the first step verified the data.

But now the processing in the later step is part of a Liberty Batch application and runs as the result of a step using batchManagerZos to submit the Java batch job. Even if the Java code tries to allocate the dataset shared, it will fail because the JCL job is waiting for the Liberty Batch job to complete and will hold the ENQ until it finishes.

We have a few options in this scenario.

#### Solution 1: DSENQSHR

If we set DSENQSHR=ALLOW for the JCL job then the ENQ will drop from exclusive to shared after the early step completes. We will have to also code the Java application to allocate the dataset 'shared' instead of 'old'. This change will allow the Liberty Batch job to execute while the JCL job is incomplete. It does give the same protection as the original JCL since holding the ENQ shared is still enough to prevent another job allocating it exclusively and updating the data.

#### Solution 2: Don't wait

In the problem description we've assumed that batchManagerZos was told to wait for the Liberty Batch job to complete before proceeding. But that is optional. If we don't care we could just let the JCL job end after submitting the Liberty Batch job. That would cause the exclusive ENQ held by the JCL job to be released and allow the Liberty Batch job to proceed. That would require the Java code to be written to wait for access if it fails rather than simply failing. It also does create a window where another job could get in between the released ENQ from the JCL and the obtained ENQ from the Java file allocation.

#### Solution 3: JobGroup

We could also break the job into two separate jobs in a Job Group and have the validation step in the first job and the launching of the Liberty Batch job in the second job. This would cause the exclusive ENQ to be dropped when the first job completes, allowing the second job to run and the Liberty Batch job to do its processing. There's no special code in Java to wait for the file to be available, but we still have the window where a job could get between the two JCL jobs and change the data.

There are probably other possibilities, but this gives you an idea of how it can be approached and some of the pros and cons to consider. You might also want to consider how realistic it is that something might try to modify the data between steps in the original JCL and thus how important it is to protect against it.

## Creating a New Dataset in JCL and using it Liberty Batch

### The Problem

This is the scenario we described at the start of this paper. In this scenario we are going to create a new dataset instead of starting with an existing one. The original pure JCL job had a DD statement with DISP=NEW for a dataset and then a later step had a DD statement with DISP=OLD. We've changed the JCL so the later step uses batchManagerZos to run that processing in Java in a Liberty Batch job.

The problem, of course, is that the DISP=NEW causes the JCL job to get an exclusive ENQ for the dataset and so the Liberty Batch job can't allocate the dataset from the Liberty address space. JES is going to hang onto the ENQ for the entire JCL job. Our solutions are similar to the previous scenario.

### Solution 1: DSENQSHR

If we set DSENQSHR=ALLOW for the JCL job then the ENQ will drop from exclusive to shared after the first step completes. We will have to also code the Java application to allocate the dataset 'shared' instead of 'old'. This change will allow the Liberty Batch job to execute while the JCL job is incomplete. The lowering of the level of the ENQ doesn't introduce any problems or potential for deadlock as long as the Java application can live with shared instead of exclusive access.

### Solution 2: Don't wait

This is exactly the same proposal from the previous scenario.

### Solution 3: JobGroup

This is also exactly the same proposal from the previous scenario. As before, the danger is that another job could get between the two jobs in the Job Group and get exclusive control of the dataset which could cause problems for the Liberty Batch job (depending how the dynamic allocation is coded).

If it is unacceptable to have the Liberty Batch job allocate the dataset shared, then this is a good solution because solution #1 can only downgrade the ENQ to shared. We have to separate the JCL into two separate jobs to get the JCL-held ENQ to be completely dropped.

## Creating a new Dataset in JCL and using it later in the JCL itself and in Liberty Batch

### The Problem

This is a variation on the previous problem. The difference in this scenario is that not only do we need to access the new dataset in the Liberty Batch job, but after that job is completed, we need to access it again in the JCL job. Our options depend on the type of access we need in the later steps.

### Solution 1: DSENQSHR

If both the Liberty Batch job and the later step in the JCL can live with shared access then setting DSENQSHR=ALLOW in the JCL JOB will solve the problem. After the initial step that creates the dataset completes the ENQ will be lowered to shared and both the Liberty Batch job and the later step (which uses DISP=SHR) can run.

However, if either later processing requires exclusive access then this approach won't work.

### Solution 2: JobGroup

We would again break the job up into three separate jobs in a Job Group. The first job would create the new dataset and complete. This will release the exclusive ENQ for the dataset.

The second job would use `batchManagerZos` to run the Liberty Batch job. The Java code can now allocate the dataset exclusively. It MUST release the dataset when processing is complete.

When `batchManagerZos` returns when the Liberty Batch job is complete, the job completes. We now move on to the third job in the Job Group which runs the step with the DD statement allocating the dataset `DISP=OLD`.

Why can't we put the `batchManagerZos` step and the `DISP=OLD` step in the same job? Because JES will see the `DISP=OLD` allocation and get an exclusive ENQ for the dataset that will prevent the Liberty Batch job from running successfully.

## Creating a Dataset in Liberty Batch to use later in the JCL

### The Problem

Here we set up a scenario that is a reverse of the ones we have considered so far. In this case we want to create a new dataset from Java in a Liberty Batch job, perhaps put some data in it, and then use it from the JCL job that launched the Liberty Batch job.

From an ENQ perspective this is fine. The exclusive ENQ will be obtained as part of the dynamic allocation performed from Java inside Liberty. When the dataset is closed and unallocated the ENQ will be released. As long as that happens as part of the job processing, there should be no problem.

The issue turns up in the cleverness of JES3 and some vendor products. The JCL will be examined and a DD statement coded `DISP=OLD` or `DISP=SHR` is referencing a dataset that does not exist at the start of the job. That looks like it won't work, so the job fails to be submitted.

### Solution 1: A fake step

This is actually an old problem. Instead of having the dataset created in a Liberty Batch job, just imagine that an earlier step simply dynamically allocates the dataset as part of the JCL job. The JCL pre-processing will still not be able to see how the dataset gets created and fail the submit.

The way around this is just as old and is a bit sleazy. The trick is to insert a step into the JCL that *looks* like it will create the dataset but actually doesn't. You use conditional JCL logic to always skip over a step in the job that has a DD statement referencing the dataset and specifying `DISP=NEW`.

The step never actually runs, but the pre-processing doesn't know that and will see the DD statement and assume that is how the dataset will get created and lets the job run.

## Using a Dataset `DISP=SHR` in JCL but exclusively from Liberty Batch

### The Problem

In this case we have a JCL job with a DD statement referencing a dataset `DISP=SHR` but we know that we're going to allocate the dataset exclusively from Java code running in a Liberty Batch job launched by this JCL.

That's going to be a problem because the JCL job will get the ENQ for the dataset shared and that will prevent the Java code from allocating the dataset exclusively.

### Solution 1: Job Group

The only way out of this problem is to break the JCL job up into separate jobs in one Job Group and keep the step(s) that reference the dataset `DISP=SHR` in separate jobs from the one that launches the Liberty Batch job.

## Dynamically Allocating a Dataset from the JCL job and using it in Liberty Batch

### The Problem

Actually there isn't one. In this case a step in the JCL job dynamically allocates a new dataset and then launches a Liberty Batch job that will allocate the same dataset.

Since the JCL itself contains no references to the dataset, no ENQs will be obtained for it for the life of the job. An ENQ will be obtained while the dataset is allocated in the JCL step, but that will get cleaned up at the end of the step. That leaves the ENQ and dataset available for use by the Liberty Batch job.

I thought it would be nice to have a scenario that just worked.

## Using a Dataset from Liberty Batch and JCL – Error Handling

### The Problem

This is the reverse of the previous scenario and it just works too, if you are careful. In this case we dynamically allocate a dataset in a Liberty Batch job and then plan to also dynamically allocate it from code running in a step in the JCL job.

Again, since there are no references in the JCL to the dataset no ENQs will be obtained for the life of the job. This means the Liberty Batch job is free to allocate the dataset.

And once the Liberty Batch job has ended and released the dataset the JCL job can dynamically allocate it without a problem.

The trick is that the Liberty Batch job *has* to have code that releases the dataset in all cases. In a traditional JCL job if a step dynamically allocates a dataset and then the step ends without releasing it, z/OS will take care of it for you as part of step-end processing.

But when a Liberty Batch job completes a step that allocated a dataset, no such cleanup occurs. That's because the Liberty Batch container has no idea the application code has allocated a dataset that needs to be cleaned up. And from a z/OS perspective this is all just one big step that is the Liberty server itself.

This scenario isn't really a problem either, but it requires more care on the application developer's part to be sure that allocated datasets are freed. If you aren't careful, then there **will** be a problem.

---

## Conclusion

We covered a lot of scenarios but certainly not all of them. If you look closely hopefully your scenario can be simplified to match one of these. Or perhaps reading through this has given you an idea about how to handle your situation. If you come up with another way to handle these sorts of situations we would appreciate an email and we'll try to get the paper updated. Thanks!

## Document change history

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

---

*September 20, 2016* Initial Version

*September 26, 2016* Updated with document number

End of WP102667