Shared Spool Mods

For Jes2 1.7 and 1.8

Users Guide and Documentation

Table of Contents

General Information and Overview

What are the Shared Spool mods, and what can they do for you?

The Shared Spool Mods have traditionally provided additional job selection criterion that could be used by one job to indicated that it should be selected for execution on a system containing a specific resource, or that it should be selected or not selected based on the execution status or resource requirements of other related jobs. Please note that it is NOT necessary to be running a Multi-Access Spool Configuration, or MAS to take advantage of this package. Beginning with this release of the Shared Spool Mods the control of jobs has been extended to include operational controls that can be used to place limits on the total number of jobs that may run concurrently in each job class on each system within a MAS, regardless of the availability of JES2 or WLM initiators. We use this feature to provide operational controls that allow us to use WLM managed initiators for jobclasses that must be otherwise regulated, like jobs classes that require a finite number of tapes. Other system level controls can be used to limit concurrently executing jobs from a singe user, or group of users based on either jobname or userid.

The Shared spool mods enhance the job selection routines that JES2 uses when selecting the next job for execution from the input queue by adding new requirements and qualifications to submitted jobs. Throughout this document, whenever we refer to job selection, we mean the process of selecting a job from the input queue for assignment to an initiator and its immediate execution.

In addition to job selection enhancements, this version of the shared spool mods provides extended control features that may be used with WLM managed initiators. The additional controls allow your operations staff to stop, start, or limit the maximum number of jobs selection for individual job classes, or for all job classes. These same limits are extended to traditional JES2 initiators as well. You are allowed to limit the number of jobs in concurrent execution based on a userid mask or jobname mask, so that you can prevent a single user, or group of users from monopolizing the available initiators in the system.

The new requirements that can be used to qualify when a job is eligible to be selected to run, or on which system it can run if you are in a MAS (Multi-Access Spool configuration), are expanded by the shared spool mods beyond existing native JES2 mechanisms. The new selection capabilities are listed below.

- Only select this job for execution on the system where it was submitted.
- Only select this job if a named WLM Scheduling Environment is available.
- Only select this job if another named job is currently running.
- Only select this job ‘After’ a named job ends.
- Always select this job ‘Before’ another named job in the input queue.

- Allow or deny job selection based on other active jobs requirement for an arbitrary named resource.
- Single thread a group of individual jobs.
- Only select this job if another named job is not currently running on the same system.
- Only select this job if another named job is currently running on the same system.

Additionally job selection can be altered in the following ways -

- All jobs can be delayed from the time they are first read until they are eligible for execution by some number of seconds.
- Delay a job by some specified amount of time, up to 100 hours.
- Delay a job until some specified time of day occurs.

System specifications can be set to restrict jobs, even jobs using WLM managed initiators in the following ways -

- A limit may be set for the number of jobs that can run in each job class.
- A limit may be set that limits the number of active jobs based on characters in the jobname.
- A limit may be set that limits the number of active jobs based on characters in the userid associated with each job.

A special option is available to prevent the use of the '$SJ(jobnumber)' to force a job into execution if it is needed. If the $SJ command is allowed, it will also override all of the above options that could prevent a job from moving into execution.

Here are some examples of how these enhancements to job selection can be used.

Assuming you have some number of jobs that can only run where a particular resource exists – a CICS region, a vendor program, or maybe where extra tape drives are attached, and assuming those resources may be available on different systems in the MAS configuration each time your jobs are submitted, you can still ensure your job is only initiated on the correct systems, by using the "/*ROUTE XEQ *scheduling environment name*" JECL statement if those resources are described by a WLM scheduling environment, and the WLM scheduling environment is activated only where the resource exists. The resource described by a WLM scheduling environment does not have to represent real resources. You could for example have a resource defined to WLM named "BATCHWINDOW" that operations or automation enables, or disables whenever they want, and jobs that are submitted with a "/*ROUTE XEQ BATCHWINDOW" statement would only be eligible for job selection when, and only on the systems where, the BATCHWINOW resource is enabled.

The "/*ROUTE XEQ HERE' statement allows jobs to execute only on the LPAR that they are submitted from, working like a dynamic SYSAFF= parameter. This option is maintained for compatibility reasons only, since the same thing can now be accomplished with standard JCL.

The "/*CNTL *resource_name*,EXC | SHR" statement can be used to create an arbitrary resource name and have any jobs you wish coordinate their availability for execution based on the need for exclusive or shared access of that arbitrary resource name. No previous resource name setup is required to use this feature; only agreement among the participating jobs on the resource name you wish to co-ordinate your job execution on.

One use of this feature is to serialize or single thread a group of jobs, where only one may run at a time. All of the jobs to be serialized would use a “/*CNTL” statement that uses the EXC or exclusive option, and the same resource name. Any resource name will do, it just has to be the same in all participating jobs. As soon as one of the jobs in the group is selected for execution all other jobs with the same “/*CNTL” statement will become ineligible for selection until the first job ends, and so on. Many times a DD statement with a dummy dataset name and a disposition of old is used to accomplish nearly the same thing, but that solution potentially ties up initiators that could be used to process other ready work and actually creates a throughput bottleneck. The use of the “/*CNTL” statements prevents the competing jobs from even being initiated, and so does not tie up, or waste initiators while each job waits for it’s turn to run. It also will not produce the annoying messages that jobs are waiting for exclusive use of the dummy dataset name. A more complex example using this feature is particularly valuable to systems programmers using SMP/e, where multiple jobs may run concurrently using the SHR option with a common name as long as each job is not actually performing updates, but a job that performs updates must run alone is coded to use the same resource name but with an EXC or EXCLUSIVE option specified, which ensures it will always run alone. In my shop we use the CSI name as the resource name, and list or check jobs are coded with a SHR option, while APPLY, or RESTORE jobs are coded with an EXC for the same resource name. At the request of ‘power’ users of this feature, a new option, PRG or PURGE, is now available that will prevent any other jobs regardless of whether or not they have a SHR or EXC specification from running until after the job with the PRG is selected and run. The PURGE option effectively says, “This job must run alone, and no other job with this resource name may start until after the job with the PRG specification has completed and it must run next.”

The “/*AFTER”, “/*BEFORE”, and “/*WITH” and “/*WITHOUT” control statements can be used to dictate the order of job selection for a group of jobs in relation to each other. There are some specific options, other than the most obvious ones, that can be specified for each system that determine rules that must be met before any of the jobs controlled by these statements is satisfied. The processing options that can be specified for these types of statements are covered in detail in the Installation Instructions, and the selection of the additional rules is set by your systems programmer.

A job class limit can be specified for each MAS member that limits the number of concurrently executing jobs in any execution class. Different limits can be set for each class, and each can be dynamically changed via operator command. The limits are effective with both traditional job classes and WLM managed job classes. This option can be used when putting a single lpar into maintenance mode, it is possible with just a couple of commands to stop all further job selection, except for the jobclass that maintenance jobs will run in, and then to restore the original environment that probably allowed any jobclass to execute. This option can be used when WLM management is in effect for a jobclass, to allow jobs in a single jobclass to run on only a single member of a MAS. This option can also be used to prevent WLM from starting too many jobs of a particular class, before it has a chance to determine the overall effect of the jobs on the system.

A mask value can be set that is used to select characters from the userid or jobname that is associated with each job, and the number of concurrently executing jobs may be limited for all jobs with a matching resultant value.

Masks are composed of up to 8 characters, with each character being either an asterisk, or a character “U”. The characters that form the userid are compared against the mask, and for each position in the mask with a ‘U’ the corresponding character from the userid is extracted. All of the extracted characters from the userid are then concatenated, to form an intermediate result. All jobs with the same intermediate result are considered to belong to the same group, and it is that group that the limit, if any, is imposed against.

As an example of limiting active jobs based on userids and a mask value, assume your installation has a standard for creating userids that says the userids are formatted with the following scheme;

Positions 1-2 represent location, LL
Positions 3-4 Are an individuals initials, II
Positions 5-6 Are the department, DD
Positions 7-8 Are job function, FF

A userid could then be said to be in the form of C’LLIIDDFF’.

Next if you wished to limit the number of jobs that were owned by users in each department, a mask of ****UU** could be set. With that mask, only positions 5 and 6 of the userid would be taken into account when determining if the maximum number of jobs that are allowed for that group. Likewise, a mask of **UU**** could be set and then the limit would be imposed for jobs that were owned by userids with the same first and last initials, or a mask of UU**UU** could be set to limit the number of active jobs for each location and department combination. Obviously then a mask of UUUUUUUU would limit the number of jobs for each userid. The masking feature is used exactly the same way with the JOBNAME as it is with the USERID.

Interaction with WLM Resources and Scheduling Environments

One of the primary job selection criterions available through the Shared Spool mods is the availability of a particular resource name being available on a particular system. In the past, we maintained these names in a table within the Shared Spool Mods code; and provided JES2 commands that would alter the state of the resources, either on or off. We now use WLM Scheduling Environment names instead. The scheduling environment names are the same ones displayed on the SDSF SE panel. The scheduling environments are on when all of the resource names that make up each environment are also on. WLM resource names are the same names displayed on the SDSF RES panel. The WLM scheduling environment names are what are matched on the “/*ROUTE XEQ scheduling environment name” JECL statements. This function is virtually identical to the new SCHENV parameter on the jobcard. In fact, we substitute the internal value of the SCHENV with what we find in the /*ROUTE XEQ card only if there is no current value set for SCHENV. These mods supported the function long before JES2 adopted the facility, which appears to have been modeled after the Shared Spool mods, and arose out of a long standing SHARE requirement.

While these mods continue to support the older style /*ROUTE XEQ statements to route jobs for execution based on resource locations, **we strongly suggest that the older /*ROUTE XEQ statements for job selection be replaced by the new IBM supported SCHENV= parameter on the jobcard.**

Compatibility and Support of the Shared Spool Mods

The Shared Spool mods, as they exist today, do not modify any JES2 source directly, other than the $USERCBS macro that is intended to be modified by the user, and only makes use of fully supported and documented exits and facilities; we see no reason why they may become unsupportable in the future.

Control Statements

You take advantage of the Shared Spool mods via JECL statements. There are currently eight supported statements;

"/*CNTL XEQ resource, { EXC | <u>SHR</u> | PRG}"
"/*ROUTE XEQ *scheduling environment name* | HERE "
"/*WITH jobname"
"/*WITHOUT jobname"
"/*AFTER jobname"
"/*BEFORE jobname"
"/*HOLDFOR hh:mm:ss"
"/*HOLDTILL hh:mm:ss"

/*CNTL –

Syntax:

/*CNTL XEQ *resourcename*,{<u>SHR</u>|EXC | PRG}

Resourcename: Resourcename is any arbitrary resource name. It may be up to 8 characters long with no embedded spaces. Resource names are NOT predefined, and may be any string of characters desired, they must of course match with the resource name specified in other jobs with which the co-ordination is desired.

SHR: Indicates the job requires shared control of the resource name. While held as shared, other jobs with the same resource name are eligible if they also require shared control, but jobs with EXC or exclusive control are not eligible for job selection. SHR is the default if neither SHR nor EXC nor PRG are specified.

EXC: Indicates the job requires exclusive control of the resource name. To be eligible for job selection no other job can currently hold the resource name with either shared or exclusive control. Once a job is selected for execution that holds the resource name as exclusive, no other job will be eligible for job selection that requires the same resource name as either shared or exclusive.

PRG: Indicates that no other jobs with the same resource name are eligible to start until after the job with the PRG specification has been selected and has completed.

Syntax Examples -

```
/*CNTL   MYNAME,SHR
/*CNTL   ANYNAME
/*CNTL   NONAME,EXC
/*CNTL   MYNAME,PRG
```

The **<u>/*CNTL</u>** statement is used to add an additional job selection criterion based on other jobs current use of the same resource name. The "/*CNTL XEQ" must be placed in columns 1 through 10. The resource name is arbitrary, and is made of up to eight (8) characters with no embedded blanks. The resource name follows the literal "/*CNTL" and must be preceded by at least one blank spaces. If the shared (SHR) or exclusive (EXC), or purge (PRG) keywords are used, they must immediately follow the resource name and be preceded by a comma. If neither the SHR nor EXC nor PRG keyword is specified SHR is assumed.

You may specify up to five /*CNTL statements; additional statements are discarded.

Examples:

Assume jobnames ABC, and CDE are currently in execution with the following /*CNTL statements –

```
//ABC  JOB (other job parms)
/*CNTL MYNAME,SHR

//CDE JOB (other job parms)
/*CNTL MYNAME,SHR
```

Then if job EFG is submitted with the following /*CNTL statement –

```
//EFG JOB (other job parms)
/*CNTL MYNAME,EXC
```

The job will not be selected for execution as long as either job ABC or CDE continues to run.

Next if job XYZ is submitted with the following /*CNTL statement –

```
        //XYZ JOB (acctng info)
        /*CNTL    MYNAME   🡐 SHR is the default
```

It would be immediately available for execution.

When no jobs remain in execution with a /*CNTL MYNAME job EFG with the EXC requirement would be available for job selection. Assume it has now been selected, and a new job enters the system with the following /*CNTL statement –

```
        //WXY  JOB  (acctng info)
        /*CNTL MYNAME
```

Job WXY will not be available for job execution until job XYZ that holds the resource name exclusively ends. If other jobs are submitted with resource names other than MYNAME, they will be treated separately and only other jobs with /*CNTL statements that reference the same resource name will affect their availability for job selection.

The PRG option, if used, will prevent any job with the same resource name specified on a /*CNTL statement from being selected for execution until after the job with the PRG option has been selected and completes execution. It effectively breaks in line at the front of all other jobs in the input queue, and once selected it also prevents any other jobs referencing the same CNTL name from being selected until it ends.

The /*CNTL statement only provides additional job selection criterion, and does not replace other JES2 requirements for job selection such as available initiators, appropriate job class and so on. The resource name is arbitrary, you make it up, there is no need for anyone to add the name to WLM or any other table before it is used. It is a "code and go" feature, just try it.

If you have a series of two or more jobs that use a DD statement with a dummy (or even real) dataset with a DISP=OLD exclusively for the purpose of ensuring that only of the jobs in the group may run at a time, please use consider using a /*CNTL resourcename,EXC statement instead of the DUMMY DD statement. Jobs that use the DUMMY dd statement, are still selected for execution, and tie up an initiator while waiting to run, jobs that use an exclusive /*CNTL statement are held up before they are selected for work and therefore do NOT tie up an initiator nor keep other jobs that could run from being initiated, not does it tie up the resources associated with an initiator.

For example - if you have 3 jobs, ABC, CDE, and EFG using dummy DD statements;

```
//ABC JOB (acctng info)
//STEP001  EXEC  PGM=OURPGM03
//DDDUM  DD      DSN=OUR.DUMMY.DSN,DISP=OLD


//CDE JOB (acctng info)
//STEP001  EXEC  PGM=OURPGM13
```

```
//DDDUM  DD      DSN=OUR.DUMMY.DSN,DISP=OLD


//EFG JOB (acctng info)
//STEP001  EXEC  PGM=OURPGM01
//DDDUM  DD      DSN=OUR.DUMMY.DSN,DISP=OLD
```

They could all easily be changed as follows, to eliminate the use of the dummy DD, and thereby eliminate unneeded contention for initiators, and stop wasting the resources held by the initiators that must wait for the enqueue on the DUMMY.DATASET.NAME to be freed.

```
//ABC JOB (acctng info)
/*CNTL  MYCNTLX,EXC              ← THIS REPLACES THE DUMMY DD
//STEP001  EXEC  PGM=OURPGM03

//CDE JOB (acctng info)
/*CNTL  MYCNTLX,EXC              ← USE THE SAME NAME ON ALL JOBS
//STEP001  EXEC  PGM=OURPGM13

//EFG JOB (acctng info)
/*CNTL  MYCNTLX,EXC              ← Efficiently uses initiator resources
//STEP001  EXEC  PGM=OURPGM01
```

/*ROUTE XEQ schenv

Syntax:

/*ROUTE XEQ *SchedulingEnvironmentName*

SchedulingEnvironmentName: This is the name of any WLM defined scheduling environment name.

```
Syntax Examples -   /*ROUTE XEQ TAPESYS
                    /*ROUTE XEQ AFTERSIX
```

The **/*ROUTE XEQ** is a standard JES2 statement used to route your job to a specific execution node. We have usurped the use of this standard JES2 statement and therefore must test to see if the statement was intended as a shared spool routing statement or a standard JES2 /*ROUTE XEQ statement. To determine if the statement is intended for standard JES facilities, when we read an XEQ statement, we first test to see if it is a valid JES2 routing destination and if it is we let JES2 handle it. If the name is not a valid name for execution routing we test to see if the name following the XEQ literal is a valid WLM scheduling environment name. If the name is a valid scheduling environment name, and if no schenv value is present on the JOB card, we use the XEQ name in the same way the job statement parameter SCHENV is used. If the name is neither a valid JES2 destination name nor a WLM scheduling environment name, we let JES2 handle it which will normally result in a JCL error.

The literal “/*ROUTE” must begin in column 1. The literal “XEQ” must follow “/*ROUTE”, and may be separated by one to twenty blank spaces. The resource name, if not a valid JES2 node name, must follow the “XEQ” literal by between 1 and 35 blank spaces. The resource name can be between 1 and 16 characters long, if the name is longer than 16 characters, only the first 16 are used.

There is one specific exception to all of the above: if the literal following XEQ is HERE, then the jobs system affinity is set to the system where you submitted the job. This can be quite a handy feature itself.

You may specify more than one /*ROUTE statement; but only the last will be used.

Examples:

//ABC JOB (job acctng),SCHENV=BEFOREALL
/*ROUTE XEQ AFTERALL   ← ignored by modes because a valid schenv is specified

//ABC JOB (job acctng),SCHENV=BEFOREALL
/*ROUTE XEQ N6   ← a valid JES2 routing node, both jes2 routing and schenv are valid for this job.

//ABC JOB (jobacctng)
/*ROUTE XEQ BEFOREALL   ← BEFOREALL is set up the same as SCHENV above

//TUV JOB (jobacctng)
/*ROUTE XEQ HERE   ← the jobs will only execute where submitted.

/*WITH jobname

Syntax:

/*WITH *jobname*

Jobname: Jobname is the name of another job that must be active in the system for this job to be considered for job selection.

Syntax Examples: /*WITH JOBNAME1
/*WITH PH2OP705

The ***/*WITH*** statement specifies that the job is only available for selection while the jobname named on the /*WITH statement is in execution. The condition is satisfied even if the jobname that must be executing is executing on a different system within the same MAS environment. If more than one /*WITH statement is read, only the last one is kept.

Example:

```
//FGH   JOB  (acctng info)
/*WITH    JKL
```

This job will only be selected for execution if jobname JKL is executing at the time of job selection. NOTE: this only works for jobs and is not effective when used with started tasks.

/*WITHOUT jobname

Syntax:

/*WITHOUT *jobname*

Jobname: Jobname is the name of another job that must not be active in the system for this job to be considered for job selection.

Syntax Examples: /*WITHOUT JOBNAME1
/*WITHOUT PH2OP705

The ***/*WITHOUT*** statement specifies that the job is only available for selection while the jobname named on the /*WITHOUT statement is not in execution. If more than one /*WITHOUT statement is read, only the last one is kept and used.

Example:

```
//FGH   JOB  (acctng info)
/*WITHOUT    JKL
```

This job will only be selected for execution if jobname JKL is NOT executing at the time of job selection. Note this works for jobs only, not started tasks.

/*AFTER -

Syntax:

/*AFTER *jobname*

Jobname: Jobname is the name of another job that if active in the system must end before this job is available for job selection.

Syntax Examples: /*AFTER JOBNAME1
/*AFTER PH2OP705

The ***/*AFTER*** jobname statement specifies that the job is only available for selection after the jobname in the /*AFTER statement has finished. When the systems programmer installed the Shared Spool Mods, he or she made a decision about the way /*AFTER and /*BEFORE statements would be handled. The first option, the TRADITIONAL option which is specified by setting the ‘BEAFTER’ SSM parm to ‘PREMOD’, says that if the job with the /*AFTER statement is selected by JES2 for execution, it will be allowed to run as long as the jobname referenced in the /*AFTER statement is not currently running, the assumption being that it may have already ended. This is the way the Shared Spool Mods have worked for many, many years.

An unfortunate consequence of the TRADITIONAL option is that when two jobs are submitted at almost the same time, “JOBA” and “JOBB”, where JOBB has a “/*AFTER JOBA” statement. The intent is clearly that JOBA should run first, and JOBB should run second, but JES2 can convert BOTH jobs at the same time, and if JOBB is converted more quickly than JOBA (perhaps because it has far fewer DD statements, or fewer procs to expand), JOBB may be available to run before JOBA ever makes it to the input queue. If this happens JOBB will go ahead and run, since it did not “see” JOBA either in the input queue or actually in execution - clearly not the intent.

An alternative to the traditional rules is put into effect by setting BEAFTER=DELAY. This option uses the TRADITIONAL rules but requires all submitted jobs to wait on the input queue for a few seconds before being allowed to be eligible for job selection. The actual wait time is variable and is setup by the installer, but can be changed via an operator command. The idea here is to prevent one job from unintentionally getting converted, or even running very quickly before all jobs in a group submitted at nearly the same time have made it to the input queue, so that they can have their control statements taken into account before allowing any one job in that group to be selected for execution.

In any case the situation rarely presents as a problem with appropriate job sequencing, but you should be aware of the possibility, and be careful about how jobs are submitted to the system, especially when submitting groups of jobs all at nearly the same time. Simply submitting the jobs in the order they are intended to run, and with a second or so between each individual job submission is normally sufficient to prevent the problem described above. Of course you may want to check with your systems programmer to see which option was chosen for your installation. If your systems programmer chose to use the PREMOD option, and you run into this particular problem, it can normally be corrected by adding a “/*HOLDFOR 00:00:04 “ statement to each of your jobs. This will cause the jobs to always wait 4 seconds before they are potentially selected for execution, and will allow even a very large job to finish input and conversion processing which will allow each job to ‘see’ the other on the input queue when they do become eligible for execution.

Finally, if more than one /*AFTER card is read; only the last one is kept.

Example:

```
//ABC JOB (acctng info)
/*AFTER    XYZ
```

Job ABC will not be available for execution while job XYZ is in execution anywhere within the MAS complex.

Also please note it is possible to form a lockout condition if JOBA is submitted with a /*AFTER JOBB, and JOBB is submitted with a /*AFTER JOBA. There is no checking is done for this type of deadly embrace, and will result in nether job ever being eligible for execution.

/*BEFORE jobname

Syntax:

/*BEFORE *jobname*

Jobname: Jobname is the name of another job in the input queue that will not be selected for execution until after this job is selected.

```
Syntax Examples:    /*BEFORE JOBNAME1
                    /*BEFORE PH2OP705
```

The **/*BEFORE** jobname statement causes the jobname specified in the statement to not be selected for execution until after this job has completed. When the Shared Spool Mods were first setup in your environment, the Systems Programmer decide upon a specific set of rules for both /*BEFORE and /*AFTER statement processing. The rules are referred to as TRADITIONAL, or DELAYED. Delayed is actually the same as traditional, except that all jobs are delayed a few seconds on the input queue before becoming eligible for execution. Please see the discussion of these rules in the explanation of the /*AFTER statement above. Specifically, if jobname ABC has a /*BEFORE XYZ statement, then if jobname XYZ is potentially selected for execution by JES2, the Shared Spool Mods will examine all jobs on the input queue and when jobname ABC is found to have a /*BEFORE for job XYZ, job XYZ will be rejected as a potential candidate for job execution. If the jobname specified in the BEFORE statement does not currently exist on the system, it is assumed to have already run, and the job becomes eligible for execution.

If more than one /*BEFORE statement is read; only the last one is kept.

Example:

```
//ABC   JOB  (acctng info)
/*BEFORE   XYZ
```

This before statement will cause JES2 to reject job XYZ as a potential candidate for job execution until after job ABC has completed execution. If job XYZ was already executing at the time job ABC is submitted, the /*BEFORE statement will not affect anything, unless there is another job XYZ waiting to execute. Also please note, it is possible to form a lockout condition where JOBA has a /*BEFORE

JOBB, and JOBB has a /*BEFORE JOBA statement. There is no checking done for this type of deadly embrace, and the result will be that neither job will ever become eligible for execution.

A few final notes for /*BEFORE and /*AFTER

A few final notes concerning the relationship between /*BEFORE, and /*AFTER. Many people try to use these statements, by stacking two or more jobs in the same PDS member and submitting them all at the same time with one SUBMIT command. This usually works as expected, but sometimes JES2 will NOT PROCESS the jobs in the order they appear in the submitted member. This can result in a job with a /*AFTER statement for a prior job you think JES2 has already seen and processed because of the sequence in the submitted member, being processed and initiated before JES2 ever finishes reading the job that is the object of the /*AFTER statement. This problem can be avoided by making sure that jobs with /*BEFORE and /*AFTER requirements are submitted separately from each other and in an appropriate sequence. The BEAFTER= option can also be set to DELAY to eliminate this type of problem, but it will result in a very small delay between job submission time, and the time a job becomes eligible for execution, for every job. If your systems programmer chose to use the PREMOD option, and you run into this particular problem, it can normally be corrected by adding a "/*HOLDFOR 00:00:04 " statement to each of your related jobs. This will cause the jobs to always wait 4 seconds before they are potentially selected for execution, and will allow even a very large job to finish input and conversion processing which will allow each job to 'see' the other on the input queue when they do become eligible for execution.

In relationship to any of the statements above, when routing a job to another node, the additional selection criterion defined by the Shared Spool Mods statements will follow the job to the new node, where it may or may not still be appropriate. Of course if the Shared Spool Mods are not installed at the receiving node, the additional job selection requirements are not honored.

/*HOLDFOR hh:mm:ss

Syntax:

```
/*HOLDFOR hh:mm:ss
```

HOLDFOR will hold a job in the input queue for the length of time specified by hh:mm:ss. HH specifies Hours, MM specifies minutes, and SS specifies seconds. A colon is required between hours and minutes, and between minutes and seconds. Up to 99 hours, 59 minutes, and 59 seconds can be specified.

Syntax Examples:	/*HOLDFOR 00:00:30	hold in input queue for 30 seconds.
	/*HOLDFOR 01:25:00	hold job in input queue for 85 minutes.

/*HOLDTIL hh:mm:ss

Syntax:

/*HOLDTIL hh:mm:ss

HOLDTIL will hold a job in the input queue until a time of day specified as hh:mm:ss. HH specifies Hours, MM specifies minutes, and SS specifies seconds. A colon is required between hours and minutes, and between minutes and seconds. If the time of day specified has already passed when the job enters the reader, it will be held until the time of day occurs on the following day. This allows you to specify a start time of 2:00am for example, and submit the job at 5:00pm before you leave work for the day. Up to 23 hours, 59 minutes, and 59 seconds can be specified. The time is always specified as a 24 hour clock, there is no am/pm indicator, and none is assumed.

Syntax Examples:	/*HOLDTIL 00:00:30	Hold until 30 seconds after midnight.
	/*HOLDFOR 18:00:00	Hold until 6pm.

The JES2 $DJ command

The JES2 $DJ command output has been extended to include information about /*CNTL statements. Up to five CNTL names are displayed qualified with an "E" for exclusive, or an "S" for shared. One /*WITH jobname, one /*WITHOUT, one /*BEFORE and one /*AFTER jobname, will each be displayed if those types of statements are present in the job. Examples of the extended displays are given below. Please note that the information is included in either the standard or long versions of the command.

Altered Display Commands –

```
-$DJ(25926)


$HASP890 JOB(T0SMX1)
$HASP890 JOB(T0SMX1)    STATUS=(AWAITING EXECUTION),CLASS=X,
$HASP890                PRIORITY=6,SYSAFF=(ANY),HOLD=(NONE),
$HASP890                DELAY RSN=HOLDTIL DELAY,AFTER=T0SMX0,
$HASP890                BEFORE=T0SMX2,WITH=T0SMA1,WITHOUT=T0SMX9,
$HASP890                HOLDFOR=00:01:00,HOLDTIL=21:50:00,
$HASP890                CNTL=(NOWAY-E,YESWAY-S,ANYWAY-S)
```

```
** NOTE the "DELAY RSN=" field, if the job has been selected by JES2 but was not
allowed to run yet because of shared spool mods control statements, it will have a
literal explaining the first hold reason found.  If the job has never been
selected by JES2, it will indicate that JES2 has not yet selected it for
execution.
```

JOBLOG messages

Informational messages, $HASP493 and $HASP494 are written to the log as jobs with /*CNTL, /*WITH, /*WITHOUT, /*HOLDTIL, /*HOLDFOR, /*BEFORE, or /*AFTER are read. Examples of the messages follow.

These messages were issued for jobname T0SM0TTY

```
$HASP944 T0SM0TTY * -- WITH   JOBNAME = T0SM0WTH   --
$HASP944 T0SM0TTY * -- AFTER  JOBNAME = T0SM0AF    --
$HASP944 T0SM0TTY * -- BEFORE JOBNAME = T0SM0BF    --
$HASP943 T0SM0TTY * -- CONTROL INFO = HOWDY,EXC    --
$HASP943 T0SM0TTY * -- CONTROL INFO = HOWDO,SHR    --
$HASP943 T0SM0TTY * -- CONTROL INFO = WILDO,SHR    --
$HASP943 T0SM0TTY * -- CONTROL INFO = HELLO,SHR    --
$HASP943 T0SMX1   * -- WITHOUT  JOBNAME = T0SMX9    --
$HASP943 T0SMX1   * --   HOLD FOR = 00:01:00        --
$HASP943 T0SMX1   * -- HOLD UNTIL = 21:50:00        --
```

SMF record logging

At installation time your systems programmer had the option of enabling SMF recording of certain Shared Spool Mods actions and options. The details of the options for SMF recording, and the actual records that are recorded, are contained in the installation documentation. This is a new option and the amount of recording is currently limited but will be enhanced in future releases.

Changes Planned for future versions of the Shared Spool Mods -

Automatic Resource routing - automatically route jobs to a specific MAS member(s) based on the program(s) or PROC names the job will execute.

Automatic Job Classing - automatically change the execution class of a job based on the programs(s), PROC names, submitting userids, or unit types specified in the JCL.

Automatic Job Accounting - Automatically generate, or change job accounting information based on the submitting userid, resources referenced in the JCL, the programs being executed, or the procs referenced in JCL.

Alternate Job Notification - will notify a user, or multiple users, only if the job ends with either a specific completion code or an ABEND code.

Step Completion Notification - will notify a user, or multiple users, if a particular step in a job ends with an unacceptable completion or ABEND code.

HSM dataset recall at job reader time - will, if requested, issue HSM recalls for migrated datasets at reader time, saving as much recall time as possible during job execution.

Expanded SMF recording - additional information will be recorded in SMF.

SMF Canned Reporting - Preset SMF reports based on the SMF records produced by the Shared Spool Mods.

OTHER IDEAS YOU HAVE - If you have an idea, or need something that the Shared Spool Mods can or should be able to do for you, PLEASE just drop me a line at SGMcCOLLEY@ALLTEL.NET and I will give your suggestion very serious consideration. I can not do everything, but I will do all that I can.

Special Considerations for WLM managed Initiators.

I either have, or will within a few days be submitting a separate file to Sam Golob who maintains the CBT TAPE website - THANK YOU VERY MUCH SAM! - . This 'other' file will deal specifically with what I have found out about WLM managed initiators, and how you can go about dealing with them from an operational standpoint. Just search for WLM managed INITs to find it on the CBT tape. All of the documentation that I have seen from IBM tells us very little about how JES2 and WLM interact, and how WLM managed initiators really work. The documentation that does exist left me with an impression, but that impression was dead wrong, I had to read the code before I knew what was really going on, and even then some of it is still hidden inside of WLM, but I am happy to share what I found, and what I found explained a lot to me about why some jobs were selected, and why others were not.

One thing in particular that I DID NOT LIKE, was that if your WLM service class is NOT at least partially based on JOB PRIORITY, then altering the PRIORITY of a job awaiting execution in a WLM managed class WILL NOT change the order it is selected for execution in. For example, if your service class is based solely on job class, and if your system becomes busy to the point that you have 8 jobs waiting for a WLM managed initiator before they will start running and if operations decides that the 5th and 6th jobs waiting in that input class are the most important, there in NOTHING THAT CAN BE DONE to alter the sequence in which the waiting jobs will be selected for execution, jobs 5 and 6 will just have to wait their turn. Of course I found a way around that, but it really is not part of the Shared Spool Mods.

** end of document **