IBM Education Assistance for z/OS V2R2

Item:
Job Execution Controls: Batch Parallelism, Dependent Jobs, Dependent Job Interfaces
Element/Component: JES2

Material current as of August 2015

Agenda

- Trademarks
- Presentation Objectives
- Overview
- Usage & Invocation
- Interactions & Dependencies
- Migration & Coexistence Considerations
- Presentation Summary
- Appendix

Trademarks

Presentation Objectives

- In this presentation, new Job Execution Controls will be introduced
- New feature in JES2 similar in function to JES3 dependent job control
  - New JOBGROUP and related JCL statement
    - Simultaneous execution using CONCURRENT statement
    - SCHEDULE statement for associated jobs with a group
  - Job group logging job
  - Commands changes
    - Updated commands that deal with job groups
    - New commands for job groups
  - SSI updates
    - Extended status changed to report on job groups
    - Job modify interactions with job groups

Overview – Details

- This presentation covers 3 items that are completely integrated in the code and this presentation:
  - Dependent Jobs
    - This is the framework that defines the new JOBGROUP concept and corresponding JCL that relate jobs into an execution zone (or group or network).
  - Batch Parallelism
    - Within the JOBGROUP construct there is something called a CONCURRENT dependency. This places a set of jobs into execution simultaneously. This statement and its supporting code implement the batch parallelism line item.
  - Dependent job interfaces
    - For both of these line items, there are SSI changes required to return information (SSI 80) and manipulate the job groups (SSI 85). These changes are part of this line item.

Overview

- Problem Statement / Need Addressed
  - Single processor speed growth will slow over time.
  - To support continued growth parallelism must increase.
  - Can be accomplished with application changes :
    - Re-code to increase parallelism
    - Replicating data bases
  - Alternative is to do it in JCL:
    - Break multi-step jobs into multiple single step jobs
    - Parallelize multiple single step job
    - Provide basic execution controls to sequence jobs correctly
    - Allow non-dependent steps to run in parallel
    - Provide for simultaneous execution of jobs
      - Support piping data between jobs
  - Not as good as rewriting an application, but can help
    - Does not help a single step job

Overview

- Solution – Design Goals
  - Should be easy to understand
  - Familiar syntax - JCL syntax rules
  - Desire to be able to see the whole picture
    - Controls defined separate from jobs
  - Have basic conditional logic similar to conditional step execution
    - Avoid COND= syntax but more like IF/THEN/ELSE
  - Application (JCL) writer centric
    - Everything submitted via internal reader (or NJE)
  - Should be usable by JCL analyzers that want to break up jobs
  - Out of the box solution that z/OS components can utilize
  - NOT intended to act as a batch job scheduler

Overview

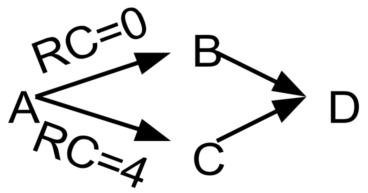
- Solution
  - No pre-canned scripts to run jobs
  - Separate entity – a job group – defined before jobs are submitted
    - Identifies all jobs in a group, dependencies, conditional processing, etc...
  - Everything about a job group is in one entity
  - Job structure assigned for logging, management, etc...
  - Jobs are associated to a job group via the SCHEDULE JCL statement (new in V2R2)
  - Can be submitted from various sources
  - JES2 commands can act on a job group
    - Display current status
    - Hold/release, cancel, purge, etc
  - No architectural limit on the number of jobs in a job group
    - Installation controllable limit of 2,000

Overview

- Benefit / Value
  - Native job execution control service within JES2
    - Always there and can be exploited by anyone
    - Syntax not dependent on product installed
  - Application centric design
    - Application writer can write and modify JCL as needed
    - No “sysprog” needed to implement parallelism
  - Installation control over key functions
    - Max number of jobs executing concurrently
  - Overall goal to allow for increased parallelism

Overview – Simple Job Group Example

A four job example using dependencies and conditionals



```
//MULTI    JOBGROUP                 JOBGROUP describes a group of dependent jobs
//A        GJOB                     GJOB describes a job in a group
//B        GJOB
//           AFTER NAME=A,          AFTER is positioned after a GJOB and describes its
//            WHEN=(RC=0)           dependency to another named job
//C        GJOB
//           AFTER NAME=A,
//            WHEN=(RC=4)           WHEN= describes a condition associated with the dependency
//D        GJOB
//           AFTER NAME=B
//           AFTER NAME=C
//MULTI    ENDGROUP                 ENDGROUP marks the end of the job group definition
```

Overview – JOBGROUP statement

```
//grpname        JOBGROUP  accounting information,
//                         programmer name,
//                         OWNER=userid,
//                         GROUP=racf group id,
//                         PASSWORD=password,
//                         SECLABEL=seclabel,
//                         TYPE=SCAN,
//                         HOLD=NO|YES,
//                         ERROR=(condition),
//                         ONERROR=(STOP|SUSPEND|FLUSH),
//                         SYSAFF=(affinity_list),
//                         SYSTEM=(system_list),
//                         SCHENV=scheduling_environment
```

- Jobs are associated with the job group using the following JCL statement
  //   SCHEDULE JOBGROUP=grpname
- The ERROR= condition is the same format as the condition on the IF JCL statement
- A JES2 job structure is created to manage the job group and to log important events for the job group and for the jobs that belong to it. This is called the “logging job”.
  - Commands like hold, cancel, display, etc... can act against the group.

Overview – JOBGROUP Logging Job

- The JES2 job identifier for a job group logging job starts with a G (eg G0001234).
- Operator commands that work on all jobs (JOBQ or JQ) do not apply to job group logging jobs.
- The job group logging job persists until all jobs in the group are purged.
- Purpose of the logging job is to maintain a job group log data set.
  - Data set maintained on SPOOL (SYSOUT)
  - No option to specify characteristics for the SYSOUT data set.
  - Available to SDSF (browse) and SAPI.
- The job group can be purged in its entirety (with all its constituent jobs).

Overview – JOBGROUP Dependencies

- Dependencies cannot be changed once established.
- You can display dependencies for an individual job in a job group or all dependencies in the entire job group.
  - Indicates status of dependency (pending or complete).
- The extended status SSI can return information about job groups, jobs within a job group, and dependencies within a job group.
  - Available to SDSF etc...
  - New output areas are associated with the job group logging job.
  - A dependency list can be returned for jobs within a job group.

Overview – JOBGROUP Statement keywords

▪ *Accounting information*	Use the accounting information parameter to enter an account number or other accounting information. Same usage as JOB statement.
▪ *Programmer's name*	Use the programmer's name parameter to identify the person or group responsible for a job group. Same usage as JOB statement.
▪ *grpname*	The name that will be associated with the job group. It will be the job name of the logging job that is associated with the group.
▪ OWNER=	The userid that is to be associated with the job group. This will propagate using the same rules as a batch job
▪ GROUP=	The RACF group that is to be associated with job group. This will propagate using the same rules as a batch job.

Overview – JOBGROUP Statement keywords

PASSWORD=	The password (if required) for the userid associated with the job group. This will propagate using the same rules as a batch job.
SECLABEL=	The security label that is to be associated with the job group. This will propagate using the same rules as a batch job.
TYPE=SCAN	The job group is validity checked but not processed. Any error will be recorded in the logging job. Since the internal structures for this job group are never created, any jobs subsequently submitted for this job group will fail.
HOLD=NO|YES	The job group can be submitted in a held or non-held state. If the job is submitted in the held state, none of the jobs associated with this job group will run until the job group is released.

Overview – JOBGROUP Statement keywords

- ERROR=(*condition*) Defines conditions, which if encountered by any job in the job group, will cause the group to be placed in an error state. The syntax is the same as for the WHEN= keyword, which will be described later. The impact of placing a job in an error state depends on the setting of the ONERROR= keyword.
- ONERROR=<u>STOP</u>|SUSPEND|FLUSH This is the action to take when a job group is determined to be in error. This applies when the condition defined on the ERROR= keyword is encountered or when a dependency is considered to have failed.

Overview – JOBGROUP Statement keywords

- There are 3 possible ONERROR actions

STOP	No new jobs in the job group are run. Currently executing jobs are allowed to complete. Jobs determined to be in error (based on the JOBGROUP ERROR= keyword or the condition in a dependency) can be resubmitted and the error state cleared if they run successfully.
SUSPEND	New jobs that have their dependencies satisfied are allowed to execute. Jobs determined to be in error (based on the JOBGROUP ERROR= keyword or the condition in a dependency) are considered to have not run. These jobs can be resubmitted and the error state cleared if they run successfully.
FLUSH	All jobs that have not executed yet will be canceled (flushed). No new jobs are started. Once there are no longer any jobs running, the job group is marked completed.

Overview – JOBGROUP Statement keywords

▪ SYSAFF=	Base system affinity for all jobs associated with this job group. Syntax is the same as SYSAFF= on the JOB card. This is ANDed with any affinity specification for each job in the group.
▪ SYSTEM=	Base list of systems where jobs associated with this job group can run. Syntax is the same as SYSTEM= on the job card. This is ANDed with any affinity specification for each job in the group.
▪ SCHENV=	Default scheduling environment for all jobs associated with this job group. Syntax is the same as SCHENV= on the job card. The systems where this scheduling environment is available is ANDed with the other affinity specifications for the job. Note this implies that a job in a job group can have 2 scheduling environments. The one for the job group and the one for the job in the group. These are ANDed together.

Overview – GJOB statement

```
//gjobname GJOB  FLUSHTYP=ALLFLUSH|ANYFLUSH
```

- All jobs in a job group must be defined with an appropriate GJOB or JOBSET statement. They do not need to be defined before their first reference.
- *gjobname*
  - The name of a job that will be included in the JOBGROUP. This name must match the name of a job that is submitted (via SCHEDULE) after the job group is defined. Each job name must be unique within the scope of the job group.
- FLUSHTYP=
  - This job will be flushed if ALL parent jobs are flushed (ALLFLUSH) or if ANY parent jobs are flushed (ANYFLUSH).
- A GJOB is followed by as many dependencies (BEFORE and AFTER) as required for the job.

Overview – JOBSET statement

```
//setname JOBSET FLUSHTYP=ALLFLUSH|ANYFLUSH
```

- A convenient method to define jobs with the same set of dependencies
- Example :

```
//SET1    JOBSET
//JOBA    SJOB
//JOBB    SJOB
//SET1    ENDSET
```

  - SJOB denotes a job within a set.
    - Cannot be referenced outside of a job set.
  - Can have as many SJOBs as needed in the set.
- BEFORE and AFTER can be added after JOBSET
  - Applies to all jobs in the job set
- BEFORE and AFTER outside the set may apply to a JOBSET
  - May only reference JOBSET name and not SJOB.
- Job sets must be ended with an ENDSET card
  - Label (*setname*) on ENDSET must match JOBSET label

Overview – JOBSET statement keywords

```
//setname JOBSET FLUSHTYP=ALLFLUSH|ANYFLUSH
```

- *setname*
  - The name that will be associated with the job set. It must be unique from any name on a GJOB statement and must be unique within the job group.
- FLUSHTYP=
  - For each job in the set, the job will be flushed if ALL parent jobs are flushed (ALLFLUSH) or if ANY parent jobs are flushed (ANYFLUSH).

Overview – Dependencies

- There are 3 forms of supported 'dependencies':
  - BEFORE
  - AFTER
  - CONCURRENT
- The syntax of the AFTER and BEFORE statements are the same.
- AFTER and BEFORE can be specified for GJOB or JOBSET.
- CONCURRENT may only be specified for GJOB.
  - Cannot be associated with a JOBSET
- You cannot have dependencies on jobs outside the JOBGROUP (or on jobs in another JOBGROUP).
  - Also, between any two jobs, you can only define one dependency.

Overview – Dependencies – Before and After statements

- The syntax of the AFTER and BEFORE statements are the same. An example of an AFTER statement

```
//          AFTER NAME=name|(name,name,...),
//                WHEN=(condition),
//                ACTION=SATISFY|FLUSH|FAIL,
//                OTHERWISE=FLUSH|FAIL|SATISFY
```

- NAME=
  - Defines jobs or job sets that the current job must run before or after.
    - Up to 10 names may be specified.
  - Each name can be
    - a job name (defined on a GJOB statement)
    - a job set name (defined on a JOBSET statement).
  - Note that each name must be unique among all the name values for the job that contains the BEFORE or AFTER statement.

Overview – Dependencies – Before and After statements

```
//          AFTER NAME=name|(name,name,...),
//                WHEN=(condition),
//                ACTION=SATISFY|FLUSH|FAIL,
//                OTHERWISE=FLUSH|FAIL|SATISFY
```

- WHEN=
  - This is an optional condition that refers to the ending status of the jobs specified on NAME=
  - Keywords supported:

• RC	indicates a job's return code
• ABEND	indicates an ABEND condition occurred
• ¬ABEND	indicates no ABEND condition occurred
• ABENDCC	indicates a specific system or user ABEND code
• RUN	indicates that the job was executed
• ¬RUN	indicates that the job was flushed from the job group

Overview – Dependencies – Before and After statements

```
//          AFTER NAME=name|(name,name,...),
//                WHEN=(condition),
//                ACTION=SATISFY|FLUSH|FAIL,
//                OTHERWISE=FLUSH|FAIL|SATISFY
```

- WHEN= operators:

		Precedence
NOT operator:		
`NOT or ¬ or !`	NOT	first
Comparison operators:		
`GT or >`	Greater than	second
`LT or <`	Less than	second
`NG or ¬> or !>`	Not greater than	second
`NL or ¬< or !<`	Not less than	second
`EQ or =`	Equal to	second
`NE or ¬= or !=`	Not equal to	second
`GE or >=`	Greater than or equal to	second
`LE or <=`	Less than or equal to	second
Logical operators:		
`AND or &`	AND	third
`OR or |`	OR	third

Overview – Dependencies – Before and After statements

```
//        AFTER NAME=name|(name,name,...),
//              WHEN=(condition),
//              ACTION=SATISFY|FLUSH|FAIL,
//              OTHERWISE=FLUSH|FAIL|SATISFY
```

- WHEN= examples:

```
WHEN=(RC=4 | RC=8)
WHEN=(!ABEND AND RC=8)
WHEN=(ABENDCC=S0C4 OR ABENDCC=U1024)
```

Overview – Dependencies – Before and After statements

```
//          AFTER NAME=name|(name,name,...),
//                WHEN=(condition),
//                ACTION=SATISFY|FLUSH|FAIL,
//                OTHERWISE=FLUSH|FAIL|SATISFY
```

- ACTION=
  - The action to take if the WHEN condition is true.
    - SATISFY The condition is to be considered satisfied. The default.
    - FLUSH The dependency is to be considered flushed. The dependent job may or may not be flushed, depending on the dependent job's FLUSHTYP=ALLFLUSH/ANYFLUSH value.
    - FAIL The failure of this dependency will mark the job group in error. The ONERROR= action from the JOBGROUP statement will be taken as a result of the failure.

Overview – Dependencies – Before and After statements

```
//          AFTER NAME=name|(name,name,...),
//                WHEN=(condition),
//                ACTION=SATISFY|FLUSH|FAIL,
//                OTHERWISE=FLUSH|FAIL|SATISFY
```

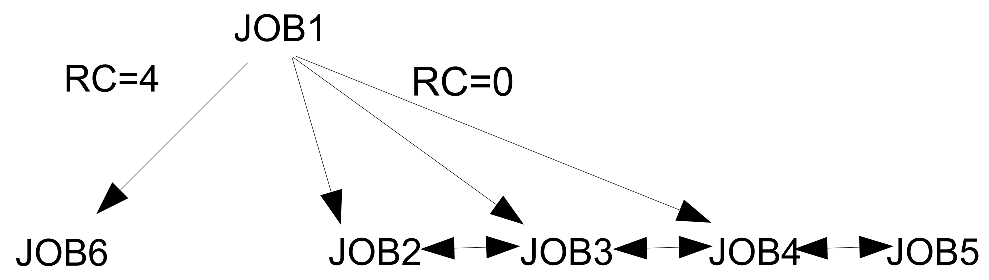
- OTHERWISE=
  - The action to take if the WHEN condition is false.
    - FLUSH The dependency is to be considered flushed. The dependent job may or may not be flushed, depending on the dependent job's FLUSHTYP=ALLFLUSH/ANYFLUSH value. The default.
    - FAIL The failure of this dependency will mark the job group in error. The ONERROR action from the JOBGROUP statement will be taken as a result of this failure.
    - SATISFY The condition is to be considered satisfied.

Overview – Dependencies – Concurrent statement

```
//gjobname GJOB
//         CONCURRENT NAME=name|(name,name,...)
```

- This states that the job specified on the GJOB statement and the job(s)/job set(s) on the CONCURRENT NAME= must execute at the same time (simultaneously) on the same JES2 MAS member.
- Note that jobs in a concurrent dependency can have other dependencies associated with them. However, all dependencies for all jobs that run concurrently must be met before any of the concurrent jobs will start.
- WLM has extended demand batch initiator to support this behavior.
- All jobs in a concurrent dependency must use the same WLM service class.
  - Should ensure this by setting the same characteristics
  - If not the same, JES2 code will pick one and force others to same value

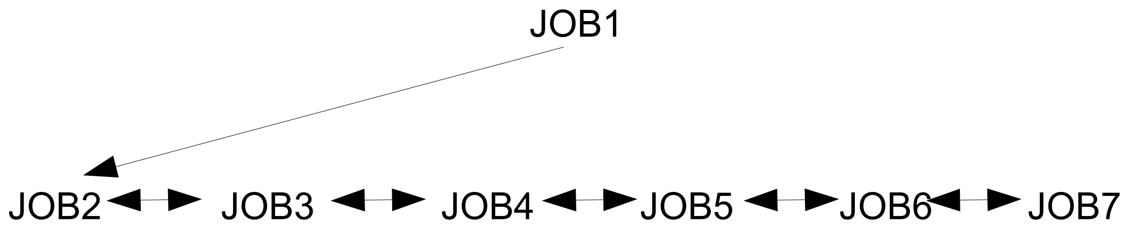
Statements Overview - GJOB-JOBSET-AFTER and CONCURRENT



```
//GRP1      JOBGROUP
//JOB1      GJOB
//JBSET1    JOBSET
//            AFTER NAME=JOB1,WHEN=(RC=0),ACTION=SATISFY,OTHERWISE=FLUSH
//JOB2      SJOB
//JOB3      SJOB
//JOB4      SJOB
//JBSET1    ENDSET
//JOB6      GJOB
//            AFTER NAME=JOB1,WHEN=(RC=4),ACTION=SATISFY,OTHERWISE=FLUSH
//JOB5      GJOB
//            CONCURRENT NAME=(JBSET1)
//GRP1      ENDGROUP
```

- Note: JOB2, JOB3, JOB4, and JOB5 will run simultaneously on same MAS member

Statements Overview - Concurrent statement and Concurrent-set



```
//GRP1          JOBGROUP
//JOB1           GJOB
//JOB2           GJOB
//                AFTER NAME=JOB1
//                CONCURRENT NAME=(JOB3,JOB4,JOB5)
//JOB6           GJOB
//                CONCURRENT NAME=(JOB3,JOB7)
//JOB3           GJOB
//JOB4           GJOB
//JOB5           GJOB
//JOB7           GJOB
//GRP1          ENDGROUP
```

- Two CONCURRENT statements above have an intersecting job – JOB3. The jobs combined In this way (JOB2, JOB3, JOB4, JOB5, JOB6, JOB7) are referred to as the concurrent set. Jobs in the concurrent set will run simultaneously on the same JES2 member after JOB1 runs.
- Once dependencies are resolved it is possible that an affinity or a resource constraint keeps the concurrent set from entering execution. $DJ may be used on any job within the set to determine the reason for the delay.

Overview – SCHEDULE JCL Statement

```
    //label    SCHEDULE JOBGROUP=grpname
```

- JOBGROUP= The name of a job group that this job will be associated with.
- Association with the groups occurs after successfully completing converter phase
  - Errors prior to this results in the job not being associated
  - A job that is successfully associated is considered “registered” to the job group.
- $DJ,JOBGROUP will show the name of the job group the job is registered to.

Overview – SCHEDULE JCL Statement Example

- Associating a job with a job group.

```
//RUN00001 JOBGROUP OWNER=IBMUSER,PASSWORD=IBMUSER
//*
//JOBA     GJOB
//           AFTER NAME=JOBB,WHEN=(RC=0)
//*
//JOBD     GJOB
//           AFTER NAME=JOBB,WHEN=(RC=4)
//*
//JOBB     GJOB
//RUN00001 ENDGROUP
//*
//JOBA     JOB TIME=NOLIMIT,REGION=0K,MSGCLASS=A,CLASS=A
//           SCHEDULE JOBGROUP=RUN00001
//STEP1    EXEC PGM=IEFBR14
//*
//JOBB     JOB TIME=NOLIMIT,REGION=0K,MSGCLASS=A,CLASS=A
//           SCHEDULE JOBGROUP=RUN00001
//STEP1    EXEC PGM=IEFBR14
//*
//JOBD     JOB TIME=NOLIMIT,REGION=0K,MSGCLASS=A,CLASS=A
//           SCHEDULE JOBGROUP=RUN00001
//STEP1    EXEC PGM=IEFBR14
```

Overview – Job Group Logging Job

- A job group logging job is created for each job group.
- The logging job has the same name and identifier as a job group.
- The logging job has two data sets:
  - JCL statements used to define the job group (JESJCLIN)
  - Job log data set (JESMSGLG)
- The logging job is a central place to collect messages related to important events in the life of the job group and its constituent jobs.
  - Jobs – executed/skipped, return codes, etc
- This job group log can be browsed from SDSF as any other job log.
  - Example of a job group log can be found later in this presentation.
- Life span of a job group logging job and data structures:
  - Remains in JES2 until all jobs in the group are purged
  - For TYPE=SCAN until the logging job itself is purged

Overview – Commands

- JES2 provides commands to manage job groups
  - Logging job is used by the commands to identify the job group
- The logging job shares the same “namespace” as regular batch jobs
  - Job identifier is prefixed by “G” (eg. “G0001234”)
  - Commands supported for job groups are similar to batch jobs
  - Specify names/identifiers like batch jobs
    - Specific name, such as 'MYGROUP'
    - Wildcards, such as 'MYGROUP*'
    - Job group range, such as 1-99
  - Job groups are not supported in JOBQ or ALL commands
    - A job group is never considered to be on the “job queue”
- Various views of job groups are provided … however
  - Job group definitions can be quite large and complex
  - Other products can be used to provide a more complete view

Overview – Commands

- Provided job group commands:

$AG	Release the job group. Jobs defined to the job group are eligible to run.
$CG	Cancel all the jobs in a job group and the job group
$DG	Display job group. Various views are provided. More later
$HG	Hold the job groups. Prevents any jobs defined to the job group from starting execution.
$PG	Purge all the jobs in the job group. When the last job is purged the job group will be scheduled to be purged.
$TG	Modify some of the key attributes defined on the JOBGROUP statement

Overview – Commands

- Keyword additions/updates to the $DJ command
  - AFTER – list jobs in the job group that are dependent on this job
    - The job in the $DJ command is the "parent" of the dependent jobs listed in the display
  - BEFORE – list of jobs in the job group that are a parent to this job
    - The job in the $DJ command is a "child" or "dependent job" of the parent job(s) listed in the display
  - JOBGROUP – name of the job group this job is registered to
  - DELAY – updates made to reflect delays due to job group settings (HOLD, SYSAFF), concurrent sets, etc.

Overview – Additional Properties of Job Groups

- Job structure of a job group logging job will be placed on the SETUP queue after input phase (assuming no errors):
  - Normal phases are:
    INPUT → SETUP → OUTPUT → HARDCOPY → PURGE
- Jobs in a job group will be placed on the SETUP queue until eligible to run (i.e. - unsatisfied dependencies exist).
- Failed jobs in a suspended job group can be rerun after a problem is fixed.
  - Based on ONERROR= not being FLUSH
  - Re-submit failed job to restart job group that is suspended.
- Must be at V2R2 checkpoint level to utilize job groups.
  - Cannot retro-activate until all logging jobs purge.

Overview – How are Job Groups Stored?

- Information about job groups are stored in the JES2 checkpoint
  - New section (CTENT) dedicated to job group information
  - Created when $ACTIVATE to LEVEL=Z22
- New data area called Zone Job Container (ZJC)
  - New CTENT implies new limit to manage
    - Initialization statement and command (GRPDEF)
  - New HASP050 message to report shortages
- Multiple ZJCs needed to represent a group
  - One per group (ZOD)
  - One per job in the group (ZJI)
  - One per dependency (ZDB)
- Created when job group submitted
- Freed when job group logging job is purged (after group completes)

Usage & Invocation – Concurrent Example

$$\begin{tikzcd}
A \arrow[d] & \\
C \arrow[r, leftrightarrow] & B
\end{tikzcd}$$

```
//G       JOBGROUP
//A       GJOB
//B       GJOB
//C       GJOB
//          AFTER NAME=A
//          CONCURRENT NAME=B
//G       ENDGROUP
//*         Schedule statements below
//*
//A       JOB TIME=NOLIMIT,REGION=0K,MSGCLASS=A,CLASS=B
//          SCHEDULE JOBGROUP=G
//STEP1   EXEC PGM=IEFBR14
//*
//B       JOB TIME=NOLIMIT,REGION=0K,MSGCLASS=A,CLASS=A
//          SCHEDULE JOBGROUP=G
//STEP1   EXEC PGM=IEFBR14
//*
//C       JOB TIME=NOLIMIT,REGION=0K,MSGCLASS=A,CLASS=A
//         SCHEDULE JOBGROUP=G
//STEP1   EXEC PGM=IEFBR14
```

Usage & Invocation – Concurrent Example – JESJCLIN

```
//G         JOBGROUP
//A         GJOB
//B         GJOB
//C         GJOB
//            AFTER NAME=A
//            CONCURRENT NAME=B
//G         ENDGROUP
```

`HASP1111 JOBGROUP has been activated`

Usage & Invocation – Concurrent Example - Job Group Log

```
                   J E S 2  J O B  L O G  --  S Y S T E M  N 1 M 1  --  N O D E  P O K

15.11.28 G0000096 ---- WEDNESDAY, 25 FEB 2015 ----
15.11.28 G0000096  IRR010I  USERID IBMUSER  IS ASSIGNED TO THIS JOB.
SY1      15.11.28 JOB00097  $HASP1300 A registered to job group G
SY1      15.11.28 JOB00097  $HASP1301 A in job group G queued for execution
SY1      15.11.28 JOB00097  $HASP373   A        STARTED - INIT 2    - CLASS B        - SYS N1M2
SY1      15.11.28 JOB00097  $HASP395   A        ENDED - RC=0000
SY1      15.11.28 JOB00098  $HASP1300 B registered to job group G
SY1      15.11.29 JOB00099  $HASP1300 C registered to job group G
SY1      15.11.29 JOB00099  $HASP1301 Concurrent set containing job C in job group G queued for execution
SY1      15.11.29 JOB00099  $HASP1201 Concurrent set containing job C in job group G entering execution
SY1      15.11.29 JOB00098  $HASP373   B        STARTED - WLM INIT  - SRVCLASS DISCRETN - SYS N1M1
SY1      15.11.29 JOB00099  $HASP373   C        STARTED - WLM INIT  - SRVCLASS DISCRETN - SYS N1M1
SY1      15.11.29 JOB00098  $HASP395   B        ENDED - RC=0000
SY1      15.11.29 JOB00099  $HASP395   C        ENDED - RC=0000
SY1      15.11.29 G0000096  $HASP1304 job group G is complete
```

- Notes:
  - Job A is using job class B which has mode=JES. Job A may run on any member within MAS – in this scenario N1M2.
  - Jobs B and C use job class A which has mode=WLM. Due to CONCURRENT keyword – B and C will run on same MAS member – in scenario case N1M1.
  - WLM initiator is started for each job within the concurrent-set.

Usage & Invocation – Type=SCAN Example with errors

```
//STEVE   JOBGROUP OWNER=IBMUSER,PASSWORD=IBMUSER,TYPE=SCAN
//JOBS1   JOBSET
//JOB!      SJOB
//JOB2      GJOB
//JOBS1   ENDSET
//JOB3    GJOB
//          AFTER NAME=JOBS1
//STEVE   ENDGROUP
```

- Errors above are:
  - Bad name on SJOB → JOB!
  - GJOB not allowed within a JOBSET
- TYPE=SCAN allows you to test your JCL without actually activating or creating a job group

Usage & Invocation – Type=SCAN Errors - JESJCLIN

```
//STEVE   JOBGROUP OWNER=IBMUSER,PASSWORD=,TYPE=SCAN
//JOBS1   JOBSET
//JOB!      SJOB
HASP1115 SJOB JOB!     name is not valid
//JOB2      GJOB
HASP1116 JOB2     not valid within JOBSET context
//JOBS1   ENDSET
HASP1118 JOBS1    not within valid JOBSET context
//JOB3    GJOB
//          AFTER NAME=JOBS1
//STEVE   ENDGROUP
HASP1111 JOBGROUP         contains errors
```

Usage & Invocation – Type=SCAN - Errors corrected

- Now, with previous errors corrected, JESJCLIN will look like this:

JESJCLIN

```
//STEVE    JOBGROUP OWNER=IBMUSER,PASSWORD=,TYPE=SCAN
//JOBS1    JOBSET
//JOB1       SJOB
//JOBS1    ENDSET
//JOB3     GJOB
//           AFTER NAME=JOBS1
//STEVE    ENDGROUP
HASP1111 JOBGROUP is valid but not activated due to TYPE=SCAN
```

Usage & Invocation – GRPDEF Init Statement and Commands

- Use GRPDEF to define and monitor key aspects of job groups.
  - Available as an initialization statement
  - Available in $D and $T commands
- $D GRPDEF provides information on job group object usage
  - ZJCNUM – total number of ZJC objects defined
  - ZJCFREE – number of ZJC objects available for use
  - ZJCWARN - % of ZJC objects used before warning message issued
    - Set through the initialization statement
- $HASP050 issued when the number of in-use ZJCs becomes greater than the ZJCWARN percentage
  - $HASP050 JES2 RESOURCE SHORTAGE OF ZJC - 80% UTILIZATION REACHED

Usage & Invocation – GRPDEF Init Statement and Command

- $D GRPDEF,ZJCUSE provides job group object type usage information

```
          $dgrpdef,zjcuse
          $HASP732 GRPDEF
$HASP732 GRPDEF  CURRENT ZJC UTILIZATION
$HASP732         TYPE         COUNT
$HASP732         -------- ----------
$HASP732         FREE             4
$HASP732         JOBGROUP         2
$HASP732         DEP JOB          9
$HASP732         DEPENDNT         5
```

  - FREE – number of ZJC objects available for use
  - JOBGROUP – number of ZJC objects used for job group info
  - DEP JOB – number of ZJC objects used for information about a dependent job in a job group
  - DEPENDNT – number of ZJC objects used for information about a dependency relationship between two jobs

Usage & Invocation – GRPDEF Init Statement and Command

- Use $T GRPDEF to manage job group definition parameters
  - ZJCNUM – Defines the number of ZJC objects allocated in checkpoint for defining parts of a job group
    - Job group information
    - Dependent job information
    - Information on dependencies between jobs
    - Range is 1 – 500000, default is 1000
  - CONCURRENT_MAX – defines the maximum number of dependent jobs that can be defined in a single concurrent job set
    - Range is 0 – 200, default is 0
  - JOBGROUP_JOB_MAX – defines the maximum number of dependent jobs that can be defined in a single job group
    - Range is 10 – 65,535, default is 2000

Usage & Invocation – Job Group Commands

- The default response for a job group command other than $TG shows:
  - JOB_GROUP_STATUS - indicating status of the job group
    - PENDING - no jobs have been registered with the job group
    - ACTIVE,INIT - at least one registered job exists
      - Not all jobs are registered
      - Some jobs may be active
    - ACTIVE - all jobs registered, jobs may be active.
      - Normal state
    - FLUSHING - non-recoverable error causes job group termination
      - All jobs that have not executed are being canceled (flushed)
      - Job group status moves to COMPLETE when last executing job completes

Usage & Invocation – Job Group Commands

- The default response for a job group command other than $TG will show:
  - JOB_GROUP_STATUS (continued)
    - SUSPENDING - at least one job INERROR
      - A subset of jobs cannot run
      - Jobs that can run will run
    - SUSPENDED - at least one job INERROR
      - A subset of jobs cannot run
      - Jobs that could run have run

Usage & Invocation – Job Group Commands

- The default command response for a job group command other than $TG will show:
  - JOB_GROUP_STATUS (continued)
    - COMPLETE - all jobs have completed and
      - All jobs that have not executed are being canceled (flushed)
      - Job group status moves to COMPLETE when last executing job completes
    - HELD - operator set or the initial state of the job group
      - No new jobs in the zone are started.
      - There may be jobs running that were started before the job group was held.

Usage & Invocation – Job Group Commands

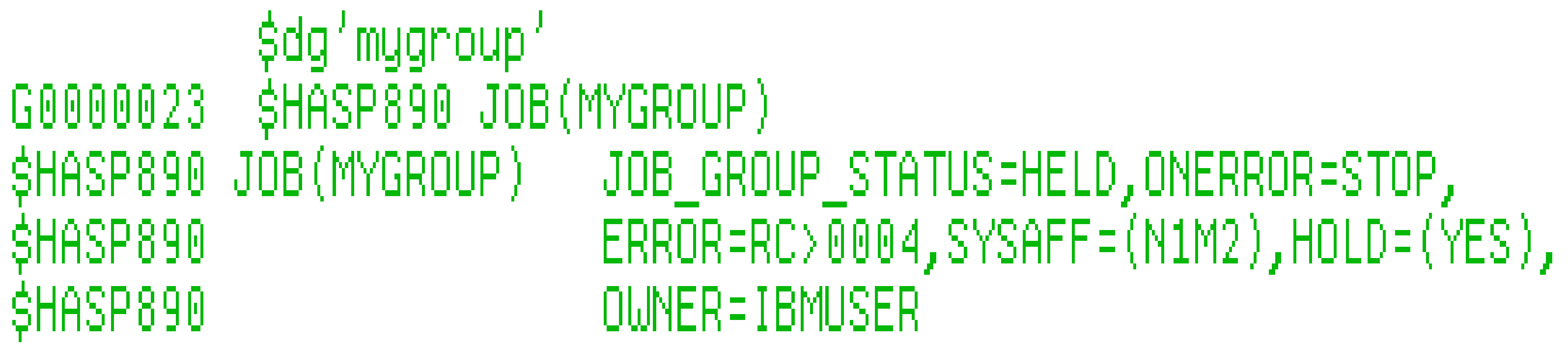
- The default command response for a job group command other than $TG will show:
  - ONERROR – default is STOP. Other values are SUSPEND and FLUSH
  - ERROR – any specified error condition is displayed
  - SYSAFF – system affinity for the job group
  - HOLD – indicates if the job group is held (YES) or not (NO)
  - OWNER – userid that owns this job group

Usage & Invocation – Job Group Commands

- The default command response for a $TG JOBGROUP command will show all fields associated with a job group.
  - These fields can be used as selection criteria
  - Fields that can be modified are:
    - SYSAFF – system affinity for the job group
    - SCHENV – scheduling environment for the job group

Usage & Invocation – Job Group Commands

- Once the JCL for a job group has been submitted the job group can be displayed using the $DG command
  - Sample output:



```
          $dg'mygroup'
G0000023  $HASP890 JOB(MYGROUP)
$HASP890 JOB(MYGROUP)   JOB_GROUP_STATUS=HELD,ONERROR=STOP,
$HASP890                ERROR=RC>0004,SYSAFF=(N1M2),HOLD=(YES),
$HASP890                OWNER=IBMUSER
```

Usage & Invocation – Job Group Commands

- The jobs defined in a job group can be displayed with the $DG,JOBS command

```
          $dg23,jobs
G0000023  $HASP890 JOB(MYGROUP)
$HASP890 JOB(MYGROUP)   JOB GROUP JOB LIST
$HASP890                JOB NAME  JOBID     JOB STAT  COMP STAT
$HASP890                --------  --------  --------  ---------
$HASP890                JOBA      JOB00024  HELD      ACTIVE
$HASP890                JOBC      JOB00026  PEND DEP  PENDING
$HASP890                JOBB      JOB00025  PEND DEP  PENDING
$HASP890                JOBD      JOB00027  PEND DEP  PENDING
```

- The JOBID column will be blank if the job has not been submitted. The JOB STAT column will indicate a job status of “NOT REG” showing the job is not registered.

Usage & Invocation – Job Group Commands

- Other values in the $DG,JOBs JOB STAT column:
  - PEND DEP – Pending dependencies keep this job from executing
  - HELD – the job is held (HOLD=(JOB))
  - ACTIVE – the job is active in execution
  - DELAYED – the job is eligible for execution. Has a resource delay.
    - Use $DJ,DELAY for further details on the delay affecting the job
  - NOT ELIG – deemed ineligible to execute (due to group error stat)
- COMP STAT column is the Completion Status of the job in terms of job group processing. Values are:
  - PENDING – job has unresolved/pending dependencies
  - ACTIVE – job is eligible for execution
  - COMPLETE – job's job group processing is complete
  - FLUSHED – job was canceled/flushed and is no longer eligible to execute
  - IN ERROR – job is in error and job group is awaiting its resubmission

Usage & Invocation – Job Group Commands

- The $DG,JOBFull version of the display job group command can be used to display additional values associated with the job definition in the job group.

```
          $dg24,jobs
G0000024  $HASP890 JOB(SHORTGRP)
$HASP890 JOB(SHORTGRP)   JOB GROUP JOB LIST
$HASP890                 JOB NAME  JOBID     JOB STAT  COMP STAT
$HASP890                 --------  --------  --------  ---------
$HASP890                 JOBA      NONE      NOT REG   PENDING
$HASP890                 JOBB      NONE      NOT REG   PENDING
          $dg24,jobf
G0000024  $HASP890 JOB(SHORTGRP)
$HASP890 JOB(SHORTGRP)   JOB_NAME=JOBA,JOBID=NONE,
$HASP890                 JOB_STAT=NOT_REG,COMP_STAT=PENDING,
$HASP890                 JOBSET_NAME=,FLUSH_ACTION=ALLFLUSH,
$HASP890                 JOB_NAME=JOBB,JOBID=NONE,
$HASP890                 JOB_STAT=NOT_REG,COMP_STAT=PENDING,
$HASP890                 JOBSET NAME=,FLUSH ACTION=ALLFLUSH
```

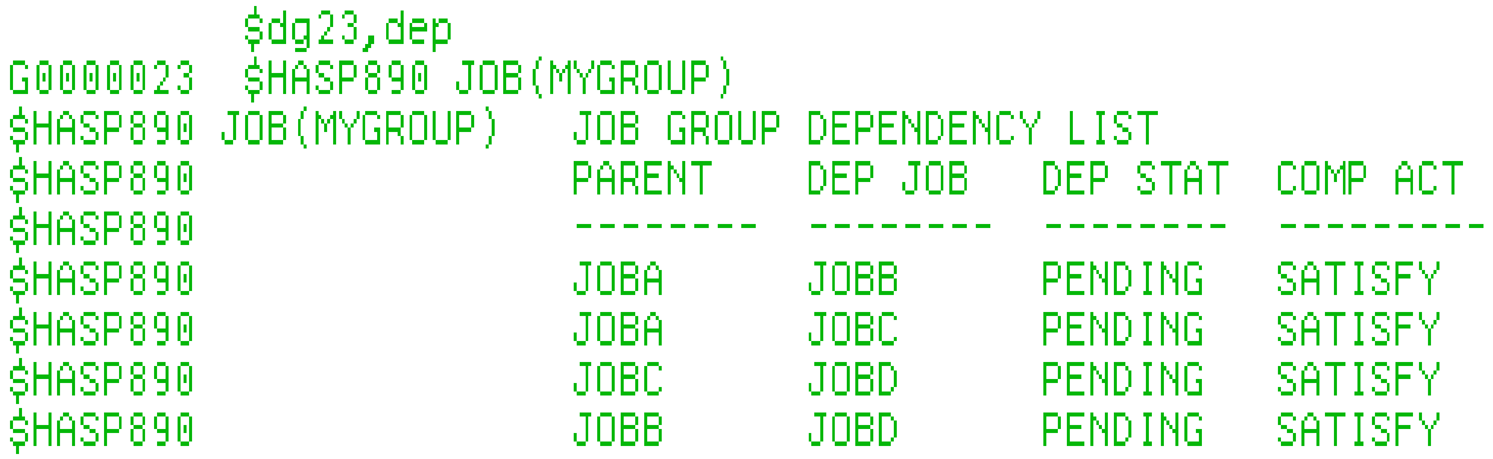
- JOB_NAME, JOBID, JOB_STAT and COMP_STAT are the same as the columns on the $DG,JOBS

Usage & Invocation – Job Group Commands

- Other job values displayed by $DG,JOBFull are:
  - JOBSET_NAME – Name of the jobset this job belongs to within the job group. Used during input processing to simplify job group definition.
  - FLUSH_ACTION – indicates the condition which would cause this job to be flushed. Values are:
    - ALLFLUSH – if all dependencies indicate a completion status of FLUSH then this job will be flushed.
    - ANYFLUSH – if any dependency indicates a completion status of FLUSH then this job will be flushed.

Usage & Invocation – Job Group Commands

- Job group dependencies can be viewed multiple ways
  - $DG,DEPlist will list all the dependencies defined in the job group



```
          $dg23,dep
G0000023  $HASP890 JOB(MYGROUP)
$HASP890 JOB(MYGROUP)   JOB GROUP DEPENDENCY LIST
$HASP890                PARENT    DEP JOB   DEP STAT  COMP ACT
$HASP890                --------- --------- --------- ---------
$HASP890                JOBA      JOBB      PENDING   SATISFY
$HASP890                JOBA      JOBC      PENDING   SATISFY
$HASP890                JOBC      JOBD      PENDING   SATISFY
$HASP890                JOBB      JOBD      PENDING   SATISFY
```

- PARENT – column indicating the parent job that must complete before the dependency of the job in the DEP JOB column can be evaluated.

Usage & Invocation – Job Group Commands

- DEP JOB – the dependent job whose evaluation depends on the completion of the parent job.
- DEP STAT – status of the dependency between the parent and dependent job. Values are:
  - PENDING – the evaluation of the dependency is pending. The parent job has not run, nor has it been flushed
  - COMPLETE – the evaluation of the dependency is complete. The parent job ran or was flushed and the completion action of the dependency has been assigned.

Usage & Invocation – Job Group Commands

- COMP ACT – Completion action. Assigned to the dependency. Used to determine how to process the dependent job when all its dependencies are complete and all dependency completion actions can be evaluated.

- Values are:
  - SATISFY – this dependency was satisfied by the parent job execution. If this completion action alone was evaluated the dependent job would be eligible for execution.
  - FLUSH – parent job results indicate the dependent job should be canceled/flushed and not executed.
  - FAIL – parent job results cause the dependency to be marked as failed and the dependent job will not run. The entire job group is marked in error and the ONERROR setting of the job group will determine the subsequent processing.

Usage & Invocation – Job Group Commands

- The $DG,DEPFull command can be used to display additional fields that do not display in the tabular output of the $DG,DEPlist command

```
          $dg23,depf
G0000023  $HASP890 JOB(MYGROUP)
$HASP890 JOB(MYGROUP)    PARENT_JOB=JOBA,DEP_JOB=JOBB,
$HASP890                 DEP_STAT=PENDING ,COMP_STAT=SATISFY,
$HASP890                 WHEN=RC=0000,ENDACTION=SATISFY,
$HASP890                 OTHERWISE=FLUSH,
$HASP890                 PARENT_JOB=JOBA,DEP_JOB=JOBC,
$HASP890                 DEP_STAT=PENDING ,COMP_STAT=SATISFY,
$HASP890                 WHEN=RC=0004,ENDACTION=SATISFY,
$HASP890                 OTHERWISE=FLUSH,
$HASP890                 PARENT_JOB=JOBC,DEP_JOB=JOBD,
$HASP890                 DEP_STAT=PENDING ,COMP_STAT=SATISFY,
$HASP890                 ENDACTION=SATISFY,OTHERWISE=FLUSH,
$HASP890                 PARENT_JOB=JOBB,DEP_JOB=JOBD,
$HASP890                 DEP_STAT=PENDING ,COMP_STAT=SATISFY,
$HASP890                 ENDACTION=SATISFY,OTHERWISE=FLUSH
```

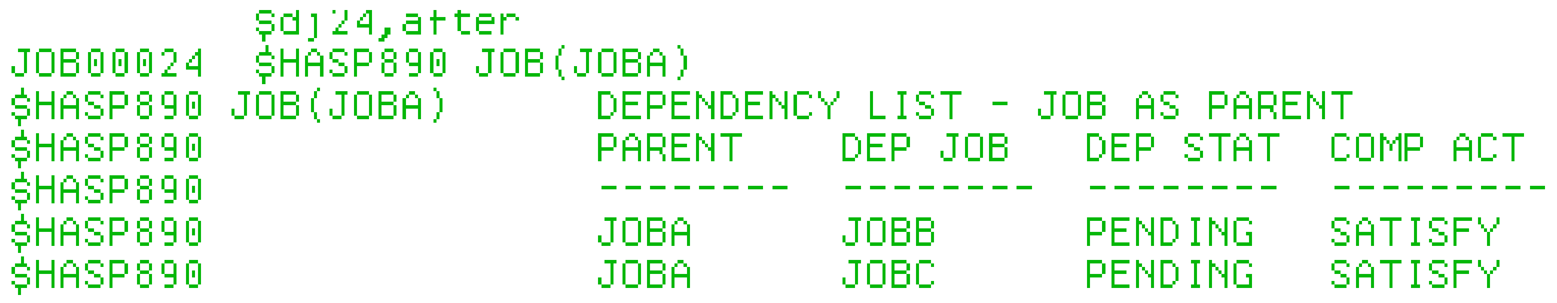
- PARENT_JOB, DEP_JOB, DEP_STAT and COMP_STAT match the columns in the $DG,DEPlist output

Usage & Invocation – Job Group Commands

- WHEN – like IF/THEN/ELSE in JCL , it is a conditional expression that, if supplied, is evaluated using the parent job's return or ABEND code once this dependency is marked COMPLETE. If the evaluation is TRUE the ENDACTION is taken, else the OTHERWISE action will be performed.
- ENDACTION – value to set dependency completion status if a supplied WHEN expression is evaluated to TRUE. Values are SATISFY, FLUSH or FAIL as in the Completion status.
- OTHERWISE – value to set dependency completion status if a supplied WHEN expression is evaluated to FALSE. Values are SATISFY, FLUSH or FAIL as in the Completion status.

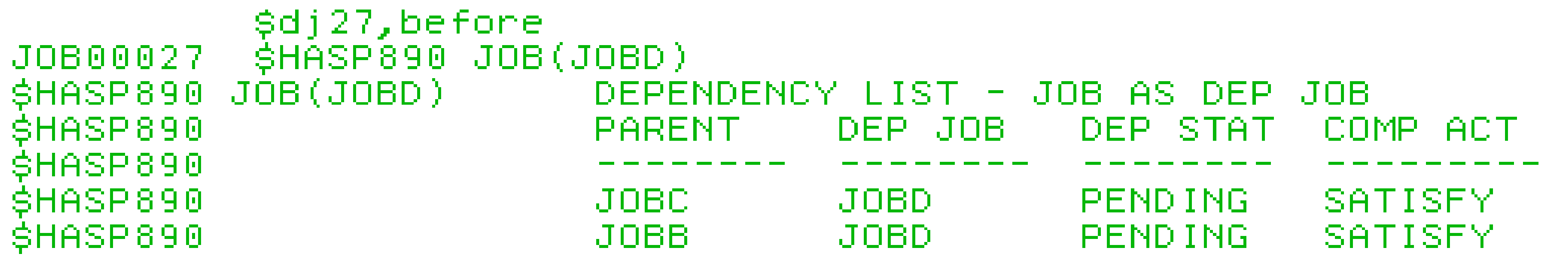
Usage & Invocation – JOB List Commands

- Dependencies for a given job can be viewed using the $DJ command.
- $DJ,AFTER will show the dependencies where the supplied job is the parent of the dependency



```
          $dj24,after
JOB00024  $HASP890 JOB(JOBA)
$HASP890 JOB(JOBA)       DEPENDENCY LIST - JOB AS PARENT
$HASP890                 PARENT    DEP JOB   DEP STAT  COMP ACT
$HASP890                 --------  --------  --------  ---------
$HASP890                 JOBA      JOBB      PENDING   SATISFY
$HASP890                 JOBA      JOBC      PENDING   SATISFY
```

- $DJ,BEFORE will show the parent jobs of the supplied dependent job



```
          $dj27,before
JOB00027  $HASP890 JOB(JOBD)
$HASP890 JOB(JOBD)       DEPENDENCY LIST - JOB AS DEP JOB
$HASP890                 PARENT    DEP JOB   DEP STAT  COMP ACT
$HASP890                 --------  --------  --------  ---------
$HASP890                 JOBC      JOBD      PENDING   SATISFY
$HASP890                 JOBB      JOBD      PENDING   SATISFY
```

Usage & Invocation – Job Group Commands

- Concurrent jobs in the job group can be viewed using $DG,CONjobs
- For this job group

```
      A=F=G
     /   \
    B     C
   /       \
  D         E
```

- The concurrent set is

```
          $dg'bruce',con
G0000014  $HASP890 JOB(BRUCE)
$HASP890 JOB(BRUCE)      JOB GROUP CONCURRENT JOB LIST
$HASP890                 JOB NAME   CONC JOB
$HASP890                 --------   --------
$HASP890                 JOBA       JOBF
$HASP890                 JOBA       JOBG
$HASP890                 JOBG       JOBA
$HASP890                 JOBG       JOBF
$HASP890                 JOBF       JOBG
$HASP890                 JOBF       JOBA
```

Usage & Invocation – Job Group Commands

- Jobs in error in the job group can be viewed using $DG,INERROR

```
          $dg23,inerror
G0000023  $HASP890 JOB(MYGROUP)
$HASP890 JOB(MYGROUP)   JOB GROUP JOB IN ERROR LIST
$HASP890                LIST OF JOBS TO RESUBMIT
$HASP890                JOB NAME  JOBID     JOB STAT  COMP STAT
$HASP890                --------  --------  --------  ----------
$HASP890                JOBA      JOB00024  Q=HARDCPY IN ERROR
```

- The job's entry in the $DG,JOBS display will show the same information

Usage & Invocation – Job Group Commands

- A more complete picture of the job group status can be obtained using the $DG,SUMMARY command. Viewing from SYSLOG is helpful:

```
$DG23,SUMMARY
$HASP890 JOB(MYGROUP) 890
$HASP890 JOB(MYGROUP)   JOB_GROUP_STATUS=SUSPENDED,ONERROR=STOP,
$HASP890                ERROR=RC>0004
$HASP890                ***************************************
$HASP890                JOB GROUP JOB IN ERROR LIST
$HASP890                LIST OF JOBS TO RESUBMIT
$HASP890                JOB NAME  JOBID     JOB STAT  COMP STAT
$HASP890                --------  --------  --------  ---------
$HASP890                JOBA      JOB00024  Q=HARDCPY IN ERROR
$HASP890                ***************************************
$HASP890                JOB GROUP JOB LIST
$HASP890                JOB NAME  JOBID     JOB STAT  COMP STAT
$HASP890                --------  --------  --------  ---------
$HASP890                JOBA      JOB00024  Q=HARDCPY IN ERROR
$HASP890                JOBC      JOB00026  PEND DEP  PENDING
$HASP890                JOBB      JOB00025  PEND DEP  PENDING
$HASP890                JOBD      JOB00027  PEND DEP  PENDING
$HASP890                ***************************************
$HASP890                JOB GROUP CONCURRENT JOB LIST
$HASP890                JOB NAME   CONC JOB
$HASP890                --------   --------
$HASP890                ***************************************
$HASP890                JOB GROUP DEPENDENCY LIST
$HASP890                PARENT    DEP JOB   DEP STAT  COMP ACT
$HASP890                --------  --------  --------  ---------
$HASP890                JOBA      JOBB      PENDING   SATISFY
$HASP890                JOBA      JOBC      PENDING   SATISFY
$HASP890                JOBC      JOBD      PENDING   SATISFY
$HASP890                JOBB      JOBD      PENDING   SATISFY
$HASP890                ***************************************
```

Usage & Invocation – Job Group Commands

- Use $CG to cancel all the jobs in a job group and the job group logging job. $CG,P will cancel and then purge the jobs and job group.
- The command issues messages indicating what jobs have been scheduled to cancel/cancel and purge

```
$CG46,P
$HASP1304 job group CANGRP is complete
$HASP890 JOB(CANGRP) 121
$HASP890 JOB(CANGRP)
$HASP890 JOBGROUP cancel - cancel issued for job JOBA
$HASP890 JOBGROUP cancel - cancel issued for job JOBG
$HASP890 JOBGROUP cancel - cancel issued for job JOBF
$HASP890 JOBGROUP cancel - cancel issued for job JOBC
$HASP890 JOBGROUP cancel - cancel issued for job JOBB
$HASP890 JOBGROUP cancel - cancel issued for job JOBD
$HASP890 JOBGROUP cancel - cancel issued for job JOBE
$HASP890 JOBGROUP cancel - cancel issued for jobgroup CANGRP
$HASP890 JOB(CANGRP) 122
$HASP890 JOB(CANGRP)    JOB_GROUP_STATUS=COMPLETE,ONERROR=STOP,
$HASP890                ERROR=RC>0004,SYSAFF=(ANY),HOLD=(NO),
$HASP890                PURGE=YES,CANCEL=YES,OWNER=IBMUSER
$HASP250 JOBA PURGED -- (JOB KEY WAS CE9525C6)
$HASP250 JOBG PURGED -- (JOB KEY WAS CE9525CC)
$HASP250 JOBF PURGED -- (JOB KEY WAS CE9525CB)
$HASP250 JOBC PURGED -- (JOB KEY WAS CE9525C8)
$HASP250 JOBB PURGED -- (JOB KEY WAS CE9525C7)
$HASP250 JOBD PURGED -- (JOB KEY WAS CE9525C9)
$HASP250 JOBE PURGED -- (JOB KEY WAS CE9525CA)
$HASP250 CANGRP PURGED -- (JOB KEY WAS CE9525C5)
```

Usage & Invocation – Job Group Commands

- Use $PG to purge all the jobs in a job group. When the last job purges out the purge will be scheduled for the job group logging job.
- The command issues messages indicating what jobs have been scheduled to purge

```
$PG37
$HASP890 JOB(CANGRP)
$HASP890 JOB(CANGRP)
$HASP890 JOBGROUP purge - purge issued for job JOBA
$HASP890 JOBGROUP purge - purge issued for job JOBG
$HASP890 JOBGROUP purge - purge issued for job JOBF
$HASP890 JOBGROUP purge - purge issued for job JOBC
$HASP890 JOBGROUP purge - purge issued for job JOBB
$HASP890 JOBGROUP purge - purge issued for job JOBD
$HASP890 JOBGROUP purge - purge issued for job JOBE
.
.
.
$HASP250 JOBG PURGED -- (JOB KEY WAS CE95250D)
$HASP250 JOBF PURGED -- (JOB KEY WAS CE95250C)
$HASP250 JOBA PURGED -- (JOB KEY WAS CE952507)
$HASP250 JOBC PURGED -- (JOB KEY WAS CE952509)
$HASP250 JOBB PURGED -- (JOB KEY WAS CE952508)
$HASP250 JOBD PURGED -- (JOB KEY WAS CE95250A)
$HASP250 JOBE PURGED -- (JOB KEY WAS CE95250B)
$HASP250 CANGRP PURGED -- (JOB KEY WAS CE952506)
```

Usage & Invocation – Job Group Commands

- Use $AG to release the hold on a job group. A held job group will keep its jobs from being selected for execution, regardless of the individual job hold values. Releasing the job group hold does not release any individual job hold values.
- Use $HG to hold a job group and keep any of new jobs from being selected for execution. The job group hold does not cause a hold to be set on the individual jobs in the job group.

Usage & Invocation – Job List Commands

- $CJ command execution differences for Dependent Jobs
  - A pre-execution/executing dependent job will be marked INERROR. Dependent jobs of this “parent” will be suspended. A corrected version of the canceled job can be resubmitted and processing will continue.
  - Canceling the last dependent job that is not complete can cause a job group state change
    - SUSPENDING → SUSPENDED if there are INERROR jobs
    - ACTIVE → COMPLETE if all jobs are completed
  - Canceling any job in an active concurrent set will cause all jobs in the concurrent set to be canceled.

Usage & Invocation – Job List Commands

- $CJ,P or $PJ command execution differences for Dependent Jobs
  - Purging the last dependent job will cause the job group to be scheduled for purge.
  - Purging any job in an active concurrent set will cause all jobs in the concurrent set to be canceled.

Usage & Invocation – SSI updates – Extended Status (SSI 80)

- The Extended Status SSI (SSI 80) was enhanced to return additional job group information :
  - New data will be returned in a job's standard output (STATJQ) :
    - The job group the job is in (if any)
    - The status of the job within the job group (if any)
    - The HOLDUNTL= and STARTBY= values (if any)
    - The WITH= job name (if any)
  - The ability to add a list of dependency objects (STATDBs) to a job's standard output (STATJQ).
    - Describes jobs where the given job is the dependent job in a parent/dependent relationship.
  - The ability to filter on a job's associated JOBGROUP name.

Usage & Invocation – SSI updates – Extended Status (SSI 80)

- The Extended Status SSI (SSI 80) was enhanced to return additional job group information (continued) :
  - The ability to differentiate 'JOBGROUP logging jobs' from regular jobs.
    - Accomplished via a new job type (batch,STC,TSU and now 'JOBGROUP logging job'), and associated SSI 'job type' filter.
    - Allows STATJQs for all job groups to be returned, etc...
  - When returning a 'JOBGROUP logging job', the ability to return a representation of it's entire dependency group.
    - If requested, an additional section is added to the STATJQ:
      - A list of all jobs (STATJQs) in the group (STATJQs)
      - A list of all dependencies (STATDBs) in the group
      - Status of the group, etc...

Usage & Invocation – SSI updates – Job Modify (SSI 85)

- The Job Modify SSI (SSI 85) was enhanced to handle job groups and jobs associated with a job group :
  - The ability to differentiate 'JOBGROUP logging jobs' from regular jobs.
    - Accomplished via a new job type (batch,STC,TSU and now 'JOBGROUP logging job'), and associated SSI 'job type' filter.
    - Allows job groups to be acted upon.
  - The ability to filter on a job's associated JOBGROUP name.

Usage & Invocation – SSI updates – Job Modify (SSI 85)

- The Job Modify SSI (SSI 85) was enhanced to handle job groups:
  - A job modify on a 'JOBGROUP logging job' will act similar to the job group ($xG) commands ($PG, $CG, etc...).
  - Note that the following behavior related to job groups :
    - Canceling/purging a job group cancels/purges all the jobs in the group.
    - Cannot purge a job group that is not COMPLETE.
    - Cannot start job ($SJ) a job group.
    - Cannot restart ($EJ) a job group.
    - Cannot spin a job group.
    - Cannot change the execution node of a job in a job group or a job group.
    - Can only change the SYSAFF and SCHENV for a job group.

Interactions & Dependencies

- Must $ACTIVATE to V2R2 checkpoint level.

Migration & Coexistence Considerations

- Support for dependent job control requires that all members in MAS are at V2R2 level.

Presentation Summary

- In this session we introduced the new Job Execution Controls
- New feature in JES2 similar in function to JES3 dependent job control
  - New JOBGROUP and related JCL statement
    - Simultaneous execution using CONCURRENT statement
    - SCHEDULE statement for associated jobs with a group
  - Job group logging job
  - Commands changes
    - Updated commands that deal with job groups
    - New commands for job groups
  - SSI updates
    - Extended status changed to report on job groups
    - Job modify interactions with job groups

Presentation Summary

- In this session we discussed new JCL that supports JES2 Job Execution Controls:
  - JOBGROUP
  - ENDGROUP
  - GJOB
  - JOBSET
  - SJOB
  - ENDSET
  - BEFORE
  - AFTER
  - CONCURRENT
  - SCHEDULE
- Also discussed the following commands
  - $AG
  - $CG
  - $DG
  - $HG
  - $PG
  - $TG
  - $CJ
  - $DJ

Appendix

- Publications
  - *z/OS V2R2.0 JES Application Programming* – SA32-0987
  - *z/OS V2R2.0 JES2 Commands* – SA32-0990
  - *z/OS V2R2.0 JES2 Initialization and Tuning Guide* – SA32-0991
  - *z/OS V2R2.0 JES2 Initialization and Tuning Reference* – SA32-0992
  - *z/OS V2R2.0 JES2 Installation Exits* – SA32-0995
  - *z/OS V2R2.0 JES2 Macros* – SA32-0996
  - *z/OS V2R2.0 JES2 Messages* – SA32-0989
  - *z/OS V2R2.0 MVS JCL Reference* - SA23-1385
  - *z/OS V2R2.0 MVS Using the Subsystem Interface* – SA38-0679