

# JOB CONTROL LENGUAJE

## JOB CONTROL LANGUAGE

- Job Control Language (JCL) le indica al sistema que programa ejecutar y proporcionar una descripción de las entradas y salidas del programas
- Que puedo hacer con JCL
  - someter un trabajo al sistema operativo
  - solicitar recursos necesarios para correr un trabajo
  - controlar el sistema para procesar el trabajo
- Que se necesita para escribir JCL
  - acceso al mainframe
  - un ID válido TSO

## QUE CONTIENE EL JOB CONTROL LANGUAGE ?

- El programa o procedimiento a ser ejecutado
- Datos de entrada
- Datos de salida
- Reportes de salida
- También proporciona información acerca de a quien pertenece el trabajo y a que cuenta cargar el trabajo

## QUE ES UN TRABAJO (JOB)

- Algo que se desea completar con la ayuda de la computadora
  - p.e. copiar un data set, ejecutar un programa o procesar varios “job steps”
- Necesario proporcionar la información que requiere el trabajo e indicar a la computadora que hacer con esta información.
- Un trabajo consiste de enunciados que controlan la ejecución de un programa o procedimiento, solicita recursos y define entradas y/o salidas.

## SINTAXIS BÁSICA DE COMANDOS DE JCL (I)

//NOMBRE OPERACIÓN OPERANDO,OPERANDO,OPERANDO, COMENTARIOS



CAMPO DE  
NOMBRE



CAMPO DE  
OPERACION



CAMPO DE  
OPERANDO



CAMPO DE  
COMENTARIOS

- **campo nombre**
  - identifica al enunciado de tal forma que otros enunciados o el sistema pueda hacer referencia a él.
  - debe ir después de la segunda diagonal
  - puede variar de 1 a 8 caracteres en longitud y solo puede contener cualquier carácter alfanumérico o @ \$ #
- **campo operación**
  - especifica el tipo de enunciado: JOB, EXEC, DD o un comando de operador

## SINTAXIS BÁSICA DE COMANDOS DE JCL (II)

//NOMBRE OPERACIÓN OPERANDO,OPERANDO,OPERANDO, COMENTARIOS



CAMPO DE  
NOMBRE



CAMPO DE  
OPERACION



CAMPO DE  
OPERANDO



CAMPO DE  
COMENTARIOS

- **campo operando**
  - contiene parámetros separados por comas
  - parámetros están compuestos de keywords y variable
- **campo comentarios**
  - opcional
  - pueden extenderse a través de la columna 80 y pueden ser incluidos si existe un campo de operando



## REGLAS GENERALES

- Debe empezar con // en columnas 1 y 2– excepto por el enunciado /\*
- Es sensible a mayúsculas – no se permiten minúsculas
- El campo de nombre es opcional– debe empezar en la columna 3 si es usado
  - debe codificar uno o más blancos si se omite
- Campo OPERATION debe comenzar antes o en la columna 16
- Campo de operandos debe terminar antes columna 72
- Los operandos son separados por comas.
- Todos los campos, excepto los operandos, deben estar separados por un espacio en blanco.

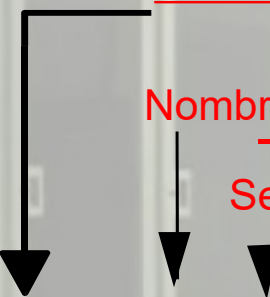
## Sintaxis básica JCL

JCL debe ser en mayúsculas

Barras en columnas 1 y 2

Nombre (1-8 caracteres) siguen las barras

Separadores de espacios


  
 //JOBNAME JOB

//STEPNAME EXEC

//DDNAME DD

/\* comment -upper or lower case

/\* ....end of JCL stream



## CONTINUACION DE ENUNCIADOS

- Conceptos introducidos como consecuencia limitación número caracteres que pueden contenerse en las tarjetas perforadas de 80 columnas.
- Continuación sintaxis JCL involucra una coma al final del último parámetro completo.
- La siguiente línea JCL incluye un // seguido de al menos un espacio y después parámetros adicionales.
- Parámetros JCL en una línea de continuación debe empezar en o antes de la columna 16 y no deben extenderse más allá de la columna 72

## CONTINUACION DE ENUNCIADOS (EJEMPLO)

- Enunciado JCL original

```
//JOB CARD JOB 1,REGION=8M,NOTIFY=ZPROF
```

- Tendría el mismo resultado que:

```
//JOB CARD JOB 1,
```

```
//    REGION=8M,
```

```
//    NOTIFY=ZPROF
```

## SENTENCIA JOB

- Informa al sistema operativo del inicio de un trabajo.
- Proporciona la información de accounting necesaria y establece los parámetros de ejecución.
- Cada trabajo debe comenzar con un enunciado JOB— //nombre-trabajo  
JOB
- El nombre-trabajo es un nombre descriptivo asignado al trabajo por el usuario que se despliega en la salida del trabajo
  - cualquier nombre de 1 a 8 caracteres alfanumérico y nacionales (\$ @ #)
  - primer caracter debe ser alfabetico o nacional

## SENTENCIA DEL JCL

- Existen tres enunciados básicos en JCL
  - JOB: identifica el principio del trabajo
  - EXEC: identifica que trabajo se debe hacer
  - DD (Data Definition): identifica que recursos son necesarios y donde encontrarlos
- Parámetros enunciados
- JOB, EXEC y DD cuentan con muchos parámetros que permiten que el usuario especifique instrucciones e información

## PARAMETROS SENTENCIA JOB (I)

- REGION=
  - solicita recursos especificos de memoria a ser asignados al trabajo
- NOTIFY=
  - envia notificación de que el trabajo se completo a un determinado usuario
- USER=
  - especifica que el trabajo se ejecuta bajo la autoridad del ID de usuario especificado
- TYPRUN=
  - retarda o detiene la ejecución, para ser liberado después
- CLASS=
  - dirige un enunciado statement para ser ejecutado en una cola de entrada en particular

## PARAMETROS SENTENCIA JOB (II)

- |   |  |
|---|--|
| • | MSGCLASS=<br>– dirige la salida del trabajo a una cola de salida en particular |
| • | MSGLEVEL=<br>– controla el numero de mensajes del sistema a ser recibidos      |
| • | EJEMPLO:<br>– //MYJOB JOB 1,NOTIFY=&SYSUID,REGION=6M                           |



## PARAMETROS SENTENCIA JOB (III)

//jobname JOB USER=userid, TIME=m, MSGCLASS=class, NOTIFY=userid

- USER=userid
  - identifica, al sistema, el usuario ejecutando el trabajo
- TIME=m
  - total de minutos maquina permitido a un trabajo ejecutar
- MSGCLASS=class
  - clase de salida para el log del trabajo
- NOTIFY=userid
  - usuario que recibirá un mensaje TSO cuando el trabajo termine

## PARAMETROS SENTENCIA EXEC (I)

- Identifica el programa de aplicación o catalogado o procedimiento de flujo que este trabajo va a ejecutar y le indica al sistema como procesar el trabajo.
- Sintaxis

`//stepname EXEC procedure,REGION=####K`

`//stepname EXEC PGM=program,REGION=####K`

## PARAMETROS SENTENCIA EXEC (II)

- PARM=
  - parámetros a pasar al programa que se va a ejecutar
- COND=
  - operadores lógicos para controlar la ejecución de otros EXEC en el trabajo
  - existen enunciados IF, THEN, ELSE, sin embargo varios de este tipo se pueden encontrar
- TIME=
  - impone un límite de tiempo
- Ejemplo
  - //MYSTEP EXEC PGM=SORT

## OPCIONES SENTENCIA EXEC

- Sintaxis
  - //stepname EXEC procedure,REGION=####K
  - //stepname EXEC PGM=program,REGION=####K
- Donde
  - stepname: palabra opcional de 1 a 8 caracteres usada para identificar el trabajo
  - EXEC: indica que se desea invocar un programa o procedimiento catalogado
  - procedure: nombre el procedimiento catalogado a ser ejecutado
  - program: nombre del programa a ser ejecutado
  - REGION=####K: monto del almacenamiento a asignar al trabajo

## PROGRAMA A EJECUTAR

- `//EXEC PGM=pgmname`
- Es una versión compilada y ligada de un conjunto de enunciados de lenguaje que estan listos para llevar a cabo una tarea determinada
- Tambien conocido como un executable load module.

## SENTENCIA DD – DATA DEFINITION

- Un enunciado DD debe ser incluido después del enunciado EXEC para cada data set usado en el paso.
- El enunciado proporciona el nombre del data set, unidad de E/S, probablemente un volumen específico a usar, y la disposición del data set.
- El sistema asegura que los dispositivos de E/S puedan ser asignados al trabajo antes que la ejecución inicie.
- El enunciado también puede proporcionar al sistema información varia acerca del data set
  - su organización, longitud registro, bloqueo, etc.



## PARAMETROS PARA SENTENCIA DD – DATA DEFINITION (I)

- Sintaxis
  - //ddname DD operando,operando,etc.
- ddname
  - un nombre de 1 a 8 caracteres proporcionado al enunciado DD
- DD
  - identificador del DD
- operando
  - parámetros usados para definir el dataset de entrada o de salida

## PARAMETROS PARA SENTENCIA DD – DATA DEFINITION (II)

- Enunciado DD cuenta con más parámetros que los enunciados JOB y EXEC
- Entre los más comunes encontramos
  - DSN=
    - el nombre del data set
    - puede incluir creación de data sets temporales o nuevos, así como una referencia al nombre del data set
  - DISP=
    - disposición de data sets
    - si el data set existe o necesita ser creado o si puede ser compartido por más de un trabajo
  - SPACE=
    - cantidad de espacio en disco requerido por el nuevo data set

## PARAMETROS PARA SENTENCIA DD – DATA DEFINITION (III)

- DCB=
  - data set control block,
  - tiene varios subparámetros
- LRECL= longitud de registro lógico, número de bytes/característica en cada registro
- RECFM= formato del registro, fijo, lógico, variable, etc
- BLOCKSIZE= se almacenan registros en un bloque de este tamaño
- DSORG= organización del data set (secuencial, particionado, etc)
- \*,DLM=
  - todo lo que sigue son datos de entrada (incluyendo //) hasta que dos caracteres alfanuméricos o caracteres especiales especificados se encuentren en la columna 1

## PARAMETROS PARA SENTENCIA DD – DATA DEFINITION (IV)

- SYSOUT=
  - define la ubicación de impresión (y la cola de salida o el data set)
- VOL=SER=
  - nombre del volumen, disco o cinta
- UNIT=
  - disco sistema, tipo especial dispositivo, o esotérico (nombre local)
- DEST=
  - rutea la salida a un destino remoto
- LABEL=
  - se espera la etiqueta de una cinta
  - una cinta puede almacenar múltiples data sets
  - cada data set en la cinta es una posición en el archivo
  - el primer data set en la cinta es el archivo 1

## PARAMETROS PARA SENTENCIA DD – DATA DEFINITION (V)

- DUMMY =
  - resulta en una entrada nulo o tirar datos escritos a este ddname
- \* (asterisco)
  - siguen datos de entrada o enunciados de control
  - es un método para pasar datos a un programa desde el stream JCL
  - los caracteres /\* marcan el final de los datos

## EJEMPLOS DE JCL

```
//MYJOB      JOB 1
//MYSORT     EXEC PGM=SORT
//SORTIN     DD DISP=SHR,DSN=IBMUSER.AREA.CODES
//SORTOUT    DD SYSOUT=*
//SYSOUT     DD SYSOUT=* //SYSIN
//DD         *
SORT FIELDS (1,3,CH,A)=
/*
```



## EJEMPLOS CREACION DE JCL

- JOB statement
  - Crear un miembro usando editor ISPF
  - Crear las sentencias JOB , EXEC y DD's
  - Crear para cada sentencia los parámetros necesarios para la ejecución.

```

EDIT      MIRIAM.PRIVATE.JCLLIB(JOB1) - 01.05          Columns 00001 00072
Command ==> _____ Scroll ==> HALF
***** ***** Top of Data *****
000001 //MIRIAM2 JOB 19,MIRIAM,NOTIFY=8SYSUID,MSGCLASS=T,
000002 // MSGLEVEL=(1,1),CLASS=A
000003 //STEP1 EXEC PGM=IEFBR14
000004 //*-----*
000005 //* THIS IS AN EXAMPLE OF A NEW DATA SET ALLOCATION
000006 //*-----*
000007 //NEWDD DD      DSN=MIRIAM.IEFBR14.TEST.NEWDD,
000008 //              DISP=(NEW,CATLG,DELETE),UNIT=SYSDA,
000009 //              SPACE=(CYL,(10,10,45)),LRECL=80,BLKSIZE=3120
***** ***** Bottom of Data *****
  
```

## SENTENCIA EXEC EN EL EJEMPLO

- EXEC statement
  - Region size

```

EDIT      MIRIAM.PRIVATE.JCLLIB(JOB1) - 01.05          Columns 00001 00072
Command ---> _____ Scroll ---> HALF
***** Top of Data *****
000001 //MIRIAM2 JOB 19,MIRIAM,NOTIFY=&SYSUID,MSGCLASS=T,
000002 // MSGLEVEL=(1,1),CLASS=A
000003 //STEP1 EXEC PGM=IEFBR14
000004 //-----*
000005 //* THIS IS AN EXAMPLE OF A NEW DATA SET ALLOCATION
000006 //*-----*
000007 //NEWDD DD      DSN=MIRIAM.IEFBR14.TEST.NEWDD,
000008 //              DISP=(NEW,CATLG,DELETE),UNIT=SYSDA,
000009 //              SPACE=(CYL,(10,10,45)),LRECL=80,BLKSIZE=3120
***** Bottom of Data *****

```

## SENTENCIA DD EN EL EJEMPLO

- DD statement
- DD name (referenciado en el programa)

DSN= (nombre data set, tal y como esta catalogado)

```
EDIT          MIRIAM.PRIVATE.JCLLIB(JOB1) - 01.05          Columns 00001 00072
Command ---> _____ Scroll ---> HALF
***** ***** Top of Data *****
000001 //MIRIAM2 JOB 19,MIRIAM,NOTIFY-8SYSUID,MSGCLASS-T,
000002 // MSGLEVEL=(1,1),CLASS=A
000003 //STEP1 EXEC PGM-IEFBR14
000004 //*-----*
000005 //* THIS IS AN EXAMPLE OF A NEW DATA SET ALLOCATION
000006 //*-----*
000007 //NEWDD DD      DSN=MIRIAM.IEFBR14.TEST.NEWDD,
000008 //              DISP=(NEW,CATLG,DELETE),UNIT=SYSDA,
000009 //              SPACE=(CYL,(10,10,45)),LRECL=80,BLKSIZE=3120
***** ***** Bottom of Data *****
```



## SENTENCIA DD EN EL EJEMPLO

- La función DISP es una de los más importantes enunciados de DD
- Entre otras cosas el parámetro DISP advierte al sistema acerca del data set
- El parámetro completo cuenta con los siguientes campos
  - DISP=(status, normal end, abnormal end)
  - DISP=(status, normal end)
  - DISP=status
  - donde status puede ser NEW, OLD, SHR o MOD

```
EDIT          MIRIAM.PRIVATE.JCLLIB(JOB1) - 01.05          Columns 00001 00072
Command ---> _____ Scroll ---> HALF
***** ***** Top of Data *****
000001 //MIRIAM2 JOB 19,MIRIAM,NOTIFY-8SYSUID,MSGCLASS-T,
000002 // MSGLEVEL=(1,1),CLASS=A
000003 //STEP1 EXEC PGM-IEFBR14
000004 //*-----*
000005 //* THIS IS AN EXAMPLE OF A NEW DATA SET ALLOCATION
000006 //*-----*
000007 //NEWDD DD      DSN=MIRIAM.IEFBR14.TEST.NEWDD,
000008 //              DISP=(NEW,CATLG,DELETE),UNIT=SYSDA,
000009 //              SPACE=(CYL,(10,10,45)),LRECL=80,BLKSIZE=3120
***** ***** Bottom of Data *****
```

## SENTENCIA DD EN EL EJEMPLO

- Si el parámetro DISP tiene un valor de NEW se debe proporcionar más información incluyendo –  
Un nombre del data set –
- El tipo de dispositivo para el data set –
- Un “volser” si se trata de un disco o una cinta etiquetada – Si se usa un disco, la cantidad de espacio para ser asignado –
- Si se trata de un data set, el tamaño del directorio debe ser especificado. –
- Opcionalmente se pueden especificar parámetros DCB • alternativamente el programa que escribirá el data set puede proporcionar dichos parámetros

```
EDIT          MIRIAM.PRIVATE.JCLLIB(JOB1) - 01.05          Columns 00001 00072
Command ---> _____ Scroll ---> HALF
***** ***** Top of Data *****
000001 //MIRIAM2 JOB 19,MIRIAM,NOTIFY=8SYSUID,MSGCLASS=T,
000002 // MSGLEVEL=(1,1),CLASS=A
000003 //STEP1 EXEC PGM=IEFBR14
000004 //*-----*
000005 //* THIS IS AN EXAMPLE OF A NEW DATA SET ALLOCATION
000006 //*-----*
000007 //NEWDD DD      DSN=MIRIAM.IEFBR14.TEST.NEWDD,
000008 //              DISP=(NEW,CATLG,DELETE),UNIT=SYSDA,
000009 //              SPACE=(CYL,(10,10,45)),LRECL=80,BLKSIZE=3120
***** ***** Bottom of Data *****
```

## SENTENCIA DD EN EL EJEMPLO

- NEW
  - indica que un nuevo data set va a ser creado
  - el trabajo tiene acceso exclusivo al data set
- OLD
  - indica que el data set ya existe y que el trabajo tiene acceso exclusivo mientras esta corriendo.
- SHR
  - indica que el data set ya existe y que varios trabajos puede tener acceso a él mientras están corriendo
  - todos los trabajos deben especificar SHR
- MOD
  - data set ya existe y el trabajo actual tiene acceso exclusivo
  - si se abre el data set para salida, la salida será añadida al final del data set

```
EDIT      MIRIAM.PRIVATE.JCLLIB(JOB1) - 01.05      Columns 00001 00072
Command ---> _____ Scroll ---> HALF
***** ***** Top of Data *****
000001 //MIRIAM2 JOB 19,MIRIAM,NOTIFY-8SYSUID,MSGCLASS-T,
000002 // MSGLEVEL=(1,1),CLASS=A
000003 //STEP1 EXEC PGM-IEFBR14
000004 //*-----*
000005 //* THIS IS AN EXAMPLE OF A NEW DATA SET ALLOCATION
000006 //*-----*
000007 //NEWDD DD      DSN=MIRIAM.IEFBR14.TEST.NEWDD,
000008 //              DISP=(NEW,CATLG,DELETE),UNIT=SYSDA,
000009 //              SPACE=(CYL,(10,10,45)),LRECL=80,BLKSIZE=3120
***** ***** Bottom of Data *****
```



## SENTENCIA DD EN EL EJEMPLO

- Si el parámetro DISP tiene un valor de NEW se debe proporcionar más información incluyendo
  - Un nombre del data set
  - El tipo de dispositivo para el data set
  - Un “volser” si se trata de un disco o una cinta etiquetada
  - Si se usa un disco, la cantidad de espacio para ser asignado
  - Si se trata de un data set, el tamaño del directorio debe ser especificado.
  - Opcionalmente se pueden especificar parámetros DCB
- alternatively el programa que escribirá el data set puede proporcionar dichos parámetros

```
EDIT          MIRIAM.PRIVATE.JCLLIB(JOB1) - 01.05          Columns 00001 00072
Command ---> _____ Scroll ---> HALF
***** ***** Top of Data *****
000001 //MIRIAM2 JOB 19,MIRIAM,NOTIFY-8SYSUID,MSGCLASS-T,
000002 // MSGLEVEL=(1,1),CLASS=A
000003 //STEP1 EXEC PGM-IEFBR14
000004 //*-----*
000005 //* THIS IS AN EXAMPLE OF A NEW DATA SET ALLOCATION
000006 //*-----*
000007 //NEWDD DD      DSN=MIRIAM.IEFBR14.TEST.NEWDD,
000008 //              DISP=(NEW,CATLG,DELETE),UNIT=SYSDA,
000009 //              SPACE=(CYL,(10,10,45)),LRECL=80,BLKSIZE=3120
***** ***** Bottom of Data *****
```

## SENTENCIA DD EN EL EJEMPLO

- Indica que hacer cuando el data set (la disposición) si el trabajo termina normal o anormalmente.
- Opciones para los dos parámetros
  - DELETE: borra y descataloga el data set
  - KEEP: guarda pero no cataloga el data set
  - CATLG: guarda y cataloga el data set
  - UNCATLG: guarda el data set pero lo descatalogaba
  - PASS: permite que un “job step” posterior especifique la disposición final.
- La disposición por defecto (para un final normal y anormal) son dejar el data set como estaba antes de que el trabajo empezara.

```
EDIT      MIRIAM.PRIVATE.JCLLIB(JOB1) - 01.05      Columns 00001 00072
Command ---> _____ Scroll ---> HALF
***** ***** Top of Data *****
000001 //MIRIAM2 JOB 19,MIRIAM,NOTIFY-8SYSUID,MSGCLASS-T,
000002 // MSGLEVEL=(1,1),CLASS=A
000003 //STEP1 EXEC PGM-IEFBR14
000004 //*-----*
000005 //* THIS IS AN EXAMPLE OF A NEW DATA SET ALLOCATION
000006 //*-----*
000007 //NEWDD DD      DSN=MIRIAM.IEFBR14.TEST.NEWDD,
000008 //              DISP=(NEW,CATLG,DELETE),UNIT=SYSDA,
000009 //              SPACE=(CYL,(10,10,45)),LRECL=80,BLKSIZE=3120
***** ***** Bottom of Data *****
```



## SENTENCIA DD EN EL EJEMPLO

- Parametro SPACE
- Requerido para asignar data sets en un DASD.
- Identifica el espacio requerido para el data set.
- Antes que data set será creado en el disco, el sistema debe saber cuanto espacio requiere el data set y como se mide este espacio.
- En su formato básico se cuenta con dos parámetros:
  - unidad de medida
  - cantidad de espacio

```
EDIT      MIRIAM.PRIVATE.JCLLIB(JOB1) - 01.05      Columns 00001 00072
Command ---> _____ Scroll ---> HALF
***** Top of Data *****
000001 //MIRIAM2 JOB 19,MIRIAM,NOTIFY=8SYSUID,MSGCLASS=T,
000002 // MSGLEVEL=(1,1),CLASS=A
000003 //STEP1 EXEC PGM=IEFBR14
000004 //*-----*
000005 //* THIS IS AN EXAMPLE OF A NEW DATA SET ALLOCATION
000006 //*-----*
000007 //NEWDD DD      DSN=MIRIAM.IEFBR14.TEST.NEWDD,
000008 //              DISP=(NEW,CATLG,DELETE),UNIT=SYSDA,
000009 //              SPACE=(CYL,(10,10,45)),LRECL=80,BLKSIZE=3120
***** Bottom of Data *****
```

## SENTENCIA DD EN EL EJEMPLO

- Parametro SPACE
- • Unidad de medida –tracks, cilindros o el tamaño de bloque promedio
- • Cantidad de espacio: tres subparámetros –1) tamaño de la primera extensión – 2) tamaño de cada extensión secundaria -3 (opcional) si existe indica que un data set particionado esta siendo creado

```
EDIT          MIRIAM.PRIVATE.JCLLIB(JOB1) - 01.05          Columns 00001 00072
Command ---> _____ Scroll ---> HALF
***** ***** Top of Data *****
000001 //MIRIAM2 JOB 19,MIRIAM,NOTIFY-8SYSUID,MSGCLASS-T,
000002 // MSGLEVEL=(1,1),CLASS=A
000003 //STEP1 EXEC PGM-IEFBR14
000004 //*-----*
000005 //* THIS IS AN EXAMPLE OF A NEW DATA SET ALLOCATION
000006 //*-----*
000007 //NEWDD DD      DSN=MIRIAM.IEFBR14.TEST.NEWDD,
000008 //              DISP=(NEW,CATLG,DELETE),UNIT=SYSDA,
000009 //              SPACE=(CYL,(10,10,45)),LRECL=80,BLKSIZE=3120
***** ***** Bottom of Data *****
```

## SENTENCIA DD EN EL EJEMPLO

- Otros Parametros
- Volser
  - formato para un dispositivo de almacenamiento en DD es VO L=SER=xxxxxx, donde xxxxxx es el volser
- Tipo dispositivo (device type)
  - existen varias formas de hacer esto
  - UNIT=xxxx es la más común
  - xxxx puede ser un dispositivo IBM (como un 3390) o una dirección específica de dispositivo (p.e. 300), o un número esotérico definido por la instalación (como un SYSDA)
- Nombre miembro (member name)
  - una librería (o un PDS) puede ser tratado como un data set por varias aplicaciones y utilerías

```
EDIT      MIRIAM.PRIVATE.JCLLIB(JOB1) - 01.05      Columns 00001 00072
Command ---> _____ Scroll ---> HALF
***** ***** Top of Data *****
000001 //MIRIAM2 JOB 19,MIRIAM,NOTIFY-8SYSUID,MSGCLASS-T,
000002 // MSGLEVEL=(1,1),CLASS=A
000003 //STEP1 EXEC PGM-IEFBR14
000004 //*-----*
000005 //* THIS IS AN EXAMPLE OF A NEW DATA SET ALLOCATION
000006 //*-----*
000007 //NEWDD DD      DSN=MIRIAM.IEFBR14.TEST.NEWDD,
000008 //              DISP=(NEW,CATLG,DELETE),UNIT=SYSDA,
000009 //              SPACE=(CYL,(10,10,45)),LRECL=80,BLKSIZE=3120
***** ***** Bottom of Data *****
```



## SENTENCIA DD EN EL EJEMPLO

### Sentencia SPACE

- Existen varios formatos y variantes, por ejemplo
  - SPACE=(TRK,10) 10 tracks, no extensiones secundarias
  - SPACE=(TRK,(10,5)) 10 tracks primarios, 5 tracks por cada extensión secundaria
  - SPACE=(CYL,5) posible usar CYL (cilindros) en lugar de tracks
  - SPACE=(TRK,(10,5,8)) PDS con 8 bloques de directorios
  - SPACE=(1000,(50000,10000))

50,000 registros primarios con 1,000 bytes cada uno

```
EDIT          MIRIAM.PRIVATE.JCLLIB(JOB1) - 01.05          Columns 00001 00072
Command ---> _____ Scroll ---> HALF
***** ***** Top of Data *****
000001 //MIRIAM2 JOB 19,MIRIAM,NOTIFY-8SYSUID,MSGCLASS-T,
000002 // MSGLEVEL=(1,1),CLASS=A
000003 //STEP1 EXEC PGM-IEFBR14
000004 //*-----*
000005 //* THIS IS AN EXAMPLE OF A NEW DATA SET ALLOCATION
000006 //*-----*
000007 //NEWDD DD      DSN=MIRIAM.IEFBR14.TEST.NEWDD,
000008 //              DISP=(NEW,CATLG,DELETE),UNIT=SYSDA,
000009 //              SPACE=(CYL,(10,10,45)),LRECL=80,BLKSIZE=3120
***** ***** Bottom of Data *****
```



## SENTENCIA DD EN EL EJEMPLO

- Unidad de medida
  - tracks, cilindros o el tamaño de bloque promedio
- Cantidad de espacio: tres subparámetros
  - tamaño de la primera extensión
  - tamaño de cada extensión secundaria (opcional)
  - si existe, indica que un data set particionado (librería) esta siendo creado

```
EDIT          MIRIAM.PRIVATE.JCLLIB(JOB1) - 01.05          Columns 00001 00072
Command ---> _____ Scroll ---> HALF
***** ***** Top of Data *****
000001 //MIRIAM2 JOB 19,MIRIAM,NOTIFY-8SYSUID,MSGCLASS-T,
000002 // MSGLEVEL=(1,1),CLASS=A
000003 //STEP1 EXEC PGM-IEFBR14
000004 //*-----*
000005 //* THIS IS AN EXAMPLE OF A NEW DATA SET ALLOCATION
000006 //*-----*
000007 //NEWDD DD      DSN=MIRIAM.IEFBR14.TEST.NEWDD,
000008 //              DISP=(NEW,CATLG,DELETE),UNIT=SYSDA,
000009 //              SPACE=(CYL,(10,10,45)),LRECL=80,BLKSIZE=3120
***** ***** Bottom of Data *****
```

## SENTENCIA DD EN EL EJEMPLO –DDNAMES RESERVADOS

- Es posible seleccionar casi cualquier nombre para un DD – recomendable usar un nombre significativo dentro límite de 8 caracteres •
- Existen algunos nombres de DD reservados que un programador no puede usar
  - //JOBLIB DD ... –
  - //STEPLIB DD ... –
  - //JOB CAT DD ... –
  - //STEP CAT DD
  - //STEP CAT DD ... –
  - //SYSABEND DD ... –
  - //SYSUDUMP DD ... –
  - //SYSMDUMP DD ... –
  - //CEEDUMP DD

## SENTENCIA DD EN EL EJEMPLO

- Un nombre ddname puede contener múltiples enunciados DD.
  - esto se conoce como concatenación
- El siguiente JCL indica que los data sets son concatenados

```
//DATAIN DD DISP=OLD,DSN=MY.INPUT1  
//      DD DISP=OLD,DSN=MY.INPUT2  
//      DD DISP=SHR,DSN=YOUR.DATA
```

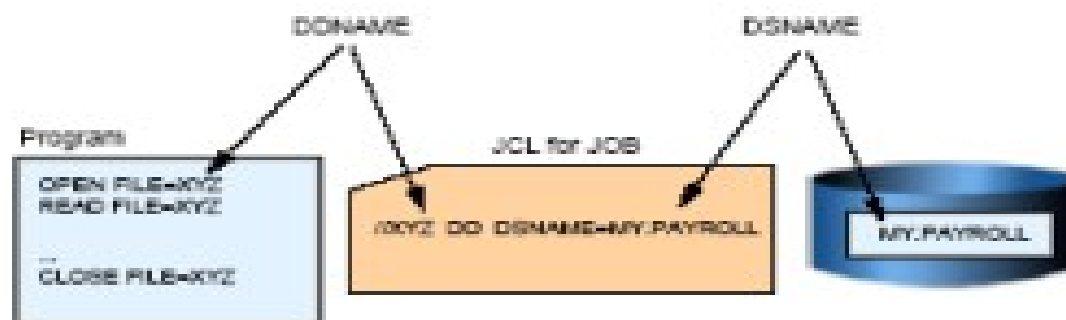
cuando el programa de aplicación lee el final de MY.INPUT1, el sistema abre automáticamente MY.INPUT2 y empieza a leerlo.

- Concatenación solo aplica a data sets de entrada.
- Los data sets son automáticamente procesados en secuencia.

## SENTENCIA DD EN EL EJEMPLO

### Nombres simbolicos

- Z/OS usa nombres archivos simbólicos.
- Aplica una redirección entre un nombre relacionado con un data set y el data set actual usado durante la ejecución del programa.



- Cuando el programa se escribe, nombre XYZ es seleccionado para referenciar al data set.
- El programa puede ser compilado y almacenado como ejecutable.
- Cuando alguien quiere correr el ejecutable, un JCL debe ser usado para relacionar el nombre XYZ al data set.
- El nombre simbólico usado en el programa es un DDNAME y el nombre real del data set es un DSNAME.

## LOS PARAMETROS EN EL EJEMPLO

nombre del trabajo

nombre del paso

```
//MYJOB JOB 1
//MYSORT EXEC PGM=SORT
//SORTIN DD DISP=SHR,DSN=IBMUSER.AREA.CODES
//SORTOUT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSIN DD *
  SORT FIELDS(1,3,CH,A)
/*
```

nombre entrada programa

nombre salida programa

especifica si la entrada serán datos o enunciados de control

donde enviar mensajes salida del sistema



## PROCEDURES

- Algunos programas y tareas requieren de una gran cantidad de JCL que un usuario puede introducir con facilidad
  - estas funciones pueden residir en librerías de procedimientos
- Un miembro de la librería de procedimiento contiene parte del JCL para una determinada tarea, usualmente la parte fija, no alterable del JCL
  - el usuario del procedimiento proporciona la parte variable del JCL
- Un procedimiento JCL es como un macro.
- También conocido como procedimiento catalogado (cataloged procedure).



# PROCEDURES

```
//MYPROC      PROC
//MYSORT      EXEC PGM=SORT
//SORTIN      DD DISP=SHR,DSN=&SORTDSN
//SORTOUT     DD SYSOUT=*
//SYSOUT      DD SYSOUT=*
//            PEND
```

- Enunciados PROC y PEND son únicos a los procedimientos
  - identifican principio y fin del procedimiento
- PROC es precedido por una etiqueta o nombre
  - en el ejemplo es MYPROC
- Las variables de sustitución de JCL son la razón por la cual JCL PROCs son usados
  - en ejemplo &SORTDSN es la única variable

## PROCEDURES - EJECUTANDO

```
//MYJOB      JOB 1
//*-----*
//MYPROC     PROC
//MYSORT     EXEC PGM=SORT
//SORTIN     DD DISP=SHR,DSN=&SORTDSN
//SORTOUT    DD SYSOUT=*
//SYSOUT     DD SYSOUT=*
//           PEND
//*-----*
//STEP1      EXEC MYPROC,SORTDSN=PRF.AREA.CODES
//SYSIN      DD *
             SORT FIELDS=(1,3,CH,A)
```

- Cuando MYJOB es enviado a ejecución se llevan a cabo la sustitución.
  - se debe proporcionar valor de &SORTDSN
  - la variable &SORTDSN y su valor debe ser colocados en una línea diferente, continuación del enunciado EXEC
  - La secuencia //SYSIN DD \* seguida del enunciado SORT serán añadidos al final del JCL substituido

## PROCEDURES - EJECUTANDO

- Cuando una sentencia entera JCL PROC necesita ser reemplazada, una secuencia de reemplazo JCL PROC puede ser usada.
  - El enunciado cuenta con la siguiente forma •
- Ejemplo //stepname.ddname DD ... reemplazo del enunciado SORTOUT DD en MYPROC – SORTOUT es dirigido a un data set secuencial recién creado

## PROCEDURES – EJECUTANDO –EJEMPLO REEMPLAZO

```
//MYJOB      JOB 1
//*-----*
//MYPROC     PROC
//MYSORT     EXEC PGM=SORT
//SORTIN     DD DISP=SHR,DSN=&SORTDSN
//SORTOUT    DD SYSOUT=*
//SYSOUT     DD SYSOUT=*
//          PEND
//*-----*
//STEP1      EXEC MYPROC,SORTDSN=ZPROF.AREA.CODES
//MYSORT.SORTOUT DD DSN=ZPROF.MYSORT.OUTPUT,
//          DISP=(NEW,CATLG),SPACE=(CYL,(1,1)),
//          UNIT=SYSDA,VOL=SER=SHARED,
//          DCB=(LRECL=20,BLKSIZE=0,RECFM=FB,DSORG=PS)
//SYSIN      DD *
            SORT FIELDS=(1,3,CH,A)
```

# CL - PROCESAMIENTO CONDICIONAL – (CONDITIONAL PROCESSING)

**El sistema JES utiliza dos enfoques para realizar el procesamiento condicional en un JCL.**

**Cuando se completa un Paso (Step) se establece un código de retorno basado en el estado de ejecución.**

**El código de retorno puede ser un número entre 0 (ejecución exitosa) y 4095 (distinto de cero muestra una condición de error). Los valores convencionales más comunes son:**

**0 = Normal - todo OK**

**4 = Advertencia: errores o problemas menores.**

**8 = Error: errores o problemas significativos.**

**12 = Error grave: errores o problemas importantes, no se debe confiar en los resultados.**

**16 = Error de terminal - Problemas muy graves, no utilice los resultados.**

**La ejecución de un paso de trabajo se puede controlar en función del código de retorno de los pasos**

**anteriores mediante el parámetro COND y la construcción IF-THEN-ELSE**



## Parámetro COND

Un parámetro **COND** se puede codificar en la instrucción JOB o EXEC de JCL.

Es una prueba del código de retorno de los pasos de trabajo anteriores.

Si la prueba se evalúa como verdadera, se omite la ejecución del paso de trabajo actual. Omitir es solo la omisión del paso del trabajo y no una terminación anormal.

Puede haber como máximo ocho condiciones combinadas en una sola prueba.

## Parámetro COND

### Sintaxis

La siguiente es la sintaxis básica de un parámetro JCL COND:

```
COND=(rc,logical-operator)
or
COND=(rc,logical-operator,stepname)
or
COND=EVEN
or
COND=ONLY
```

Aquí está la descripción de los parámetros utilizados:

- **rc** : Este es el código de retorno
- **operador lógico** : puede ser GT (mayor que), GE (mayor o igual que), EQ (igual que), LT (menor que), LE (menor o igual que) o NE (no igual a).
- **stepname** : este es el paso de trabajo cuyo código de retorno se usa en la prueba.

## COND dentro de la Sentencia JOB

Cuando COND se codifica en la instrucción JOB, la condición se prueba para cada paso del trabajo.

Cuando la condición es verdadera en cualquier paso de trabajo en particular, se omite junto con los pasos de trabajo que le siguen.

Ejemplo de uso de COND en un JOB:

```
//CNDSAMP JOB CLASS=6,NOTIFY=&SYSUID,COND=(5,LE)
```

```
/*
```

```
//STEP10 EXEC PGM=FIRSTP
```

```
/* STEP10 executes without any test being performed.
```

```
//STEP20 EXEC PGM=SECONDP
```

```
/* STEP20 is bypassed, if RC of STEP10 is 5 or above.
```

```
/* Say STEP10 ends with RC4 and hence test is false.
```

```
/* So STEP20 executes and lets say it ends with RC16.
```

```
//STEP30 EXEC PGM=SORT
```

```
/* STEP30 is bypassed since 5 <= 16.
```

## COND dentro de la instrucción EXEC

Cuando COND se codifica en la declaración EXEC de un paso de trabajo y se determina que es verdadero, solo se omite ese paso de trabajo y la ejecución continúa desde el siguiente paso de trabajo.

```
//CNDSAMP JOB CLASS=6,NOTIFY=&SYSUID
//*
//STP01 EXEC PGM=SORT
//* Assuming STP01 ends with RC0.

//STP02 EXEC PGM=MYCOBB,COND=(0,EQ,STP01)
//* In STP02, condition evaluates to TRUE and step bypassed.

//STP03 EXEC PGM=IEBGENER,COND=((10,LT,STP01),(10,GT,STP02))
//* In STP03, first condition fails and hence STP03 executes.
//* Since STP02 is bypassed, the condition (10,GT,STP02) in
//* STP03 is not tested.
```



## COND=EVEN

Cuando se codifica COND=EVEN, se ejecuta el paso de trabajo actual, incluso si alguno de los pasos anteriores termina anormalmente. Si se codifica cualquier otra condición RC junto con COND=EVEN, el paso del trabajo se ejecuta si ninguna de las condiciones RC es verdadera.

```
//CNDSAMP JOB CLASS=6,NOTIFY=&SYSUID
/*
//STP01 EXEC PGM=SORT
/* Assuming STP01 ends with RC0.

//STP02 EXEC PGM=MYCOBB,COND=(0,EQ,STP01)
/* In STP02, condition evaluates to TRUE and step bypassed.

//STP03 EXEC PGM=IEBGENER,COND=((10,LT,STP01),EVEN)
/* In STP03, condition (10,LT,STP01) evaluates to true,
/* hence the step is bypassed.
```



## COND=ONLY

Cuando se codifica COND=ONLY, se ejecuta el paso de trabajo actual, solo cuando cualquiera de los pasos anteriores finaliza de manera anormal. Si se codifica cualquier otra condición de RC junto con COND=ONLY, entonces el paso de trabajo se ejecuta si ninguna de las condiciones de RC es verdadera y cualquiera de los pasos de trabajo anteriores falla de manera anormal.

```
//CNDSAMP JOB CLASS=6,NOTIFY=&SYSUID
/*
//STP01 EXEC PGM=SORT
/* Assuming STP01 ends with RC0.

//STP02 EXEC PGM=MYCOBB,COND=(4,EQ,STP01)
/* In STP02, condition evaluates to FALSE, step is executed
/* and assume the step abends.

//STP03 EXEC PGM=IEBGENER,COND=((0,EQ,STP01),ONLY)
/* In STP03, though the STP02 abends, the condition
/* (0,EQ,STP01) is met. Hence STP03 is bypassed.
```

## Construcción IF-THEN-ELSE

Otro enfoque para controlar el procesamiento del trabajo es mediante el uso de construcciones IF-THEN-ELSE. Esto brinda más flexibilidad y una forma fácil de usar de procesamiento condicional.

### Sintaxis

```
//name IF condition THEN  
list of statements /* action to be taken when condition is true  
//name ELSE  
list of statements /* action to be taken when condition is false  
//name ENDIF
```

EN-ELSE:

A continuación se encuentra la descripción de los términos utilizados en la construcción IF-THEN-ELSE anterior:

- **nombre** : esto es opcional y un nombre puede tener de 1 a 8 caracteres alfanuméricos que comiencen con el alfabeto, #, \$ o @.
- **Condición** : Una condición tendrá un formato: **VALOR DEL OPERADOR DE LA PALABRA CLAVE** , donde **las PALABRAS CLAVE** pueden ser RC (Código de retorno), ABENDCC (Código de finalización del sistema o del usuario), ABEND, EJECUTAR (ejecución iniciada por el paso). Un **OPERADOR** puede ser un operador lógico (AND (&), OR (|)) o un operador relacional (<, <=, >, >=, <>).

Ejemplo IF THEN ELSE: El siguiente es un ejemplo simple que muestra el uso de IF-THEN-ELSE:

```
//CNDSAMP JOB CLASS=6,NOTIFY=&SYSUID
//*
//PRC1 PROC
//PST1 EXEC PGM=SORT
//PST2 EXEC PGM=IEBGENER
// PEND
//STP01 EXEC PGM=SORT
//IF1 IF STP01.RC = 0 THEN
//STP02 EXEC PGM=MYCOBB1,PARM=123
// ENDIF
//IF2 IF STP01.RUN THEN
//STP03a EXEC PGM=IEBGENER
//STP03b EXEC PGM=SORT
// ENDIF
//IF3 IF STP03b.!ABEND THEN
//STP04 EXEC PGM=MYCOBB1,PARM=456
// ELSE
// ENDIF
//IF4 IF (STP01.RC = 0 & STP02.RC <= 4) THEN
//STP05 EXEC PROC=PRC1
// ENDIF
//IF5 IF STP05.PRC1.PST1.ABEND THEN
//STP06 EXEC PGM=MYABD
// ELSE
//STP07 EXEC PGM=SORT
// ENDIF
```

Intentemos analizar el programa anterior para comprenderlo con un poco más de detalle:

- El código de retorno de STP01 se prueba en IF1. Si es 0, entonces se ejecuta STP02. De lo contrario, el procesamiento pasa a la siguiente instrucción IF (IF2).
- En IF2, si STP01 ha comenzado la ejecución, entonces se ejecutan STP03a y STP03b.
- En IF3, si STP03b no ABEND, entonces se ejecuta STP04. En ELSE, no hay sentencias. Se llama declaración NULL ELSE.
- En IF4, si STP01.RC = 0 y STP02.RC <=4 son VERDADEROS, entonces se ejecuta STP05.
- En IF5, si el paso de proceso PST1 en PROC PRC1 en el paso de trabajo STP05 ABEND, entonces se ejecuta STP06. De lo contrario, se ejecuta STP07.
- Si IF4 se evalúa como falso, entonces STP05 no se ejecuta. En ese caso, no se prueban IF5 y no se ejecutan los pasos STP06, STP07.

IF-THEN-ELSE no se ejecutará en el caso de una finalización anormal del trabajo, como la cancelación del trabajo por parte del usuario, el vencimiento del tiempo de trabajo o la referencia inversa de un conjunto de datos a un paso que se omite.



## Establecer puntos de control

Puede establecer un conjunto de datos de punto de control dentro de su programa JCL usando **SYSCKEOV**, que es una declaración DD.

Un **CHKPT** es el parámetro codificado para conjuntos de datos QSAM de varios volúmenes en una declaración DD. Cuando un CHKPT se codifica como CHKPT=EOV, se escribe un punto de control en el conjunto de datos especificado en la instrucción SYSCKEOV al final de cada volumen del conjunto de datos de varios volúmenes de entrada/salida.

```
//CHKSAMP JOB CLASS=6,NOTIFY=&SYSUID
//*
//STP01      EXEC PGM=MYCOBB
//SYSCKEOV   DD DSN=SAMPLE.CHK,DISP=MOD
//IN1        DD DSN=SAMPLE.IN,DISP=SHR
//OUT1       DD DSN=SAMPLE.OUT,DISP=(,CATLG,CATLG)
//           CHKPT=EOV,LRECL=80,RECFM=FB
```

En el ejemplo anterior, se escribe un punto de control en el conjunto de datos SAMPLE.CHK al final de cada volumen del conjunto de datos de salida SAMPLE.OUT.

## Reiniciar procesamiento

Puede reiniciar el procesamiento de ether de forma automática con el **parámetro RD** o manual con el **parámetro RESTART**

• El **parámetro RD** está codificado en la declaración JOB o EXEC y ayuda en el reinicio automático de JOB/STEP y puede contener uno de los cuatro valores: R, RNC, NR o NC.

- **RD=R** permite reinicios automáticos y considera el punto de control codificado en el parámetro CHKPT de la instrucción DD.
- **RD=RNC** permite reinicios automáticos, pero anula (ignora) el parámetro CHKPT.
- **RD=NR** especifica que el trabajo/paso no se puede reiniciar automáticamente. Pero cuando se reinicia manualmente usando el parámetro RESTART, se considerará el parámetro CHKPT (si lo hay).
- **RD=NC** no permite el reinicio automático ni el procesamiento de puntos de control.

Si existe un requisito para realizar un reinicio automático solo para códigos anormales específicos, entonces se puede especificar en el miembro **SCHEDxx** de la biblioteca parmlib del sistema IBM.



**El parámetro RESTART** está codificado en la instrucción JOB o EXEC y ayuda en el reinicio manual del TRABAJO/PASO después de la falla del trabajo. RESTART puede ir acompañado de un checkid, que es el punto de control escrito en el conjunto de datos codificado en la instrucción SYSCKEOV DD. Cuando se codifica un checkid, la instrucción SYSCHK DD debe codificarse para hacer referencia al conjunto de datos del punto de control después de la declaración JOBLIB (si existe), o después de la declaración JOB.

```
//CHKSAMP JOB CLASS=6,NOTIFY=&SYSUID,RESTART=(STP01,chk5)
//*
//SYSCHK DD DSN=SAMPLE.CHK,DISP=OLD
//STP01 EXEC PGM=MYCOBB
//*SYSCKEOV DD DSNAME=SAMPLE.CHK,DISP=MOD
//IN1 DD DSN=SAMPLE.IN,DISP=SHR
//OUT1 DD DSN=SAMPLE.OUT,DISP=(,CATLG,CATLG)
// CHKPT=EOV,LRECL=80,RECFM=FB
```

En el ejemplo anterior, chk5 es el checkid, es decir, STP01 se reinicia en el punto de control5. Tenga en cuenta que se agrega una declaración SYSCHK y se comenta la declaración SYSCKEOV en el programa anterior explicado en la sección Configuración del punto de control