***This module is designed for three contact hours of class time***

Module Objectives:

* Be able to discuss the role of JCL for scheduling and batch processing
* Be able to run programs that utilize sequential files and JCL on the System z

Background:Chapters 4,5 & 6 of zconcepts.pdf in the Reference Module

**How To—Run a Simple Cobol Program with JCL**

**A Simple Cobol Program in a JCL Environment -- Overview**

The program listing below is a simple Cobol program that reads a record, moves part of the information to an output record and then writes the output record. In the skeleton code, the records written will be in a file that you can view—in the **acctid.work.out** pds. The name of the file will be COBOUT.

This simple example can also serve as a beginning introduction to JCL (**Note—all JCL must be upper case**) and how it works. We will discuss this in class but note that importance of certain JCL statements—JOB, EXEC, and DD. For example, the EXEC statement invokes either a program or a procedure. In this case, the EXEC invokes the Procedure (**IGYWCLG**)which compiles links and executes the Cobol program. This means that procedure IGYWCLG calls the Cobol compiler to compile the program and then take the compiler output and link it with all the supporting functions and programs it needs to execute (the linkage editor) and then run the package. The CLG of IGYW**CLG** means Compile, Link and Go.

The DD (Data Definition) statements are very important as they always reference data. For example, consider the statement

//COBOL.SYSIN DD \*

The SYSIN means system input and a DD statement with an \* means the input immediately follows the DD statement. In this case, it means that the Cobol program follows immediately as the Cobol source code will be the input. Consider that you could reference the Cobol program from a PDS, but in this case the Cobol program was placed here to make it easy to follow.

Consider the statements

 SELECT CARD-IN ASSIGN TO UT-S-READER.

//\*

//GO.READER DD \*

ANDERSON ESSABIA ANNIECE XXXXXXXXX ISYS4283 002 EANDERS PWD001

...

 ...

/\*

//

The Cobol SELECT statement is the definition of the input file and this illustrates how the Cobol program knows where to locate the input data. In this case, the DD \* means the data follows immediately. Note the name **Reader** is defined by the programmer and has no inherent meaning.

Can you guess the meaning of // and the //\* ?Consider the statements

SELECT DISK-FILE1 ASSIGN TO UT-S-DISKFL1

...

 ...

//GO.DISKFL1 DD DSN=ACCTID.WORK.OUT(COBOUT),DISP=SHR,

// VOL=SER=DB1469,SPACE=(CYL,(1,1,1)),UNIT=SYSDA,

// DCB=(RECFM=FB,LRECL=80,BLKSIZE=6400,DSORG=PO)

The SELECT statement defines the desired output. Again, it points to a DD statement. In this case, the Cobol program creates and writes records to the COBOUT member of a PDS (any PDS will work but ACCTID.WORK.OUT is recommend above for output). The following terms are used in this definition.

DISP(MOD, KEEP) DISP 🡪 Disposition. Relates what is to be done with the PDS. In this case, it is modify and keep. There are many possibilities but can you guess what NEW, CATLG and OLD, DELETE do?

VOL=SER=DB1469 VOL 🡪 Volume, SER 🡪 SERIAL. This is analogous to a letter for a disk drive on your PC. TEMP91 is the name of the disk.

SPACE=(CYL,(1,1,1)) Defines space for the file in terms of cylinders—you could also use tracks

UNIT=SYSDA SYSDA 🡪 Systems Direct Access and means a disk

DCB=(RECFM=FB… DCB 🡪 Data Control Block, LRECL 🡪 Logical Record Length

 BLKSIZE🡪Block Size and should be multiple of the record length

 DSORG🡪Dataset Organization – Partitioned Organization

Note that the real world environment is much more complicated in that jobs can be scheduled to run and the normal mode would be that a stack of jobs must be run sequentially (with the output of one job feeding the input to the next job). Data is being pulled from many different files and databases, repackaged, sorted, summarized, etc. as it moves through the process. This discussion is hopefully for you to gain an appreciation of JCL and the need for it.

Even though real world JCL can be considerable more complex, this example can be reduced even more. The following one line of JCL statement will work in place of the three lines of JCL above by accepting defaults for many of the parameters.

**//GO.DISKFL1 DD DSN=ACCTID.WORK.OUT(COBOUT),DISP=SHR**

**The complete Job Stream that runs the Cobol Program is listed below.**

*//COBJOB JOB (COBOLPROG),'ACCTID',NOTIFY=ACCTID,*

// CLASS=A,MSGLEVEL=(1,1),TIME=1,MSGCLASS=A

//STEP1 EXEC PROC=IGYWCLG

//COBOL.SYSIN DD \*

 IDENTIFICATION DIVISION.

Use your account id

 PROGRAM-ID. COBTEST.

 ENVIRONMENT DIVISION.

 INPUT-OUTPUT SECTION.

 FILE-CONTROL.

 SELECT CARD-IN ASSIGN TO UT-S-READER.

 SELECT DISK-FILE1 ASSIGN TO UT-S-DISKFL1.

 DATA DIVISION.

 FILE SECTION.

 FD DISK-FILE1 LABEL RECORDS ARE STANDARD

 RECORDING F

 BLOCK CONTAINS 0 RECORDS.

 01 DSK-RCD1.

 02 FILLER PIC X(80).

 FD CARD-IN LABEL RECORDS OMITTED

 RECORDING F.

 01 CRD-REC.

 02 CRD-NAME PIC X(30).

 02 FILLER PIC X(23).

 02 CRD-U-ID PIC X(7).

 02 FILLER PIC X.

 02 CRD-PASSW PIC X(6).

 02 FILLER PIC X(13).

 WORKING-STORAGE SECTION.

 01 EOF PIC X VALUE 'N'.

 01 HOLD-INST.

 02 CRD-CRSE PIC X(30).

 02 FILLER PIC X(23).

 02 CRD-INST-ID PIC X(7).

 02 FILLER PIC X.

 02 CRD-PWD PIC X(6).

 02 FILLER PIC X(13).

 01 OUTPUT-INST.

 02 CRD-OUT-ID PIC X(7).

 02 FILLER PIC X(2) VALUE SPACES.

 02 CRD-OUT-PWD PIC X(6).

 02 FILLER PIC X(65) VALUE SPACES.

 PROCEDURE DIVISION.

 OPEN INPUT CARD-IN

 OUTPUT DISK-FILE1

 PERFORM 2000-RD-CRD-RT

 PERFORM 1000-PROCESS-IT

 UNTIL EOF = 'Y'

 CLOSE CARD-IN

 DISK-FILE1

 STOP RUN.

 2000-RD-CRD-RT.

 READ CARD-IN INTO HOLD-INST

 AT END

 MOVE 'Y' TO EOF.

 1000-PROCESS-IT.

 MOVE CRD-INST-ID TO CRD-OUT-ID

 MOVE CRD-PWD TO CRD-OUT-PWD

 WRITE DSK-RCD1 FROM OUTPUT-INST

You will need to create this PDS—**without the COBOUT member - it will be created by the program.**

 PERFORM 2000-RD-CRD-RT.

//\*

//GO.DISKFL1 DD DSN=ACCTID.WORK.OUT(COBOUT),DISP=SHR

//GO.READER DD \*

ANDERSON ESSABIA ANNIECE XXXXXXXXX ISYS4283 002 EANDERS PWD001

ANIPA EKO A XXXXXXXXX ISYS4283 002 EAANIPA PWD002

BLOMBERG BETTY ANN XXXXXXXXX ISYS4283 002 BBLOMBE PWD003

CORDIERO SHELDON JOHN XXXXXXXXX ISYS4283 002 SCORDEI PWD004

NESTRUD CHRISTOPHER CHARLES XXXXXXXXX ISYS4283 002 CCN PWD005

RODRIGUEZ SARAH MARTHA XXXXXXXXX ISYS4283 002 SRODRIG PWD006

SIMMONS KIMBERLY M XXXXXXXXX ISYS4283 002 KMS06 PWD007

WIDEMAN JOHN RANDELL XXXXXXXXX ISYS4283 002 JWIDEMA PWD008

/\*

//

**Running the COBOL program on the Mainframe – (ISPF environment)**

1. Create the PDS ACCTID.WORK.OUT – This PDS is where the output from the program will be written.
2. Upload the text file named SIMPLE (JCL, Cobol, data) from into the ACCTID.WORK.COMPILES PDS.
3. Edit the program (select 2 from the ISPF Primary Option Menu) and make the changes as needed (change ACCTID to your account id, etc.).
4. Save the program by typing save on the command line and pressing Enter.
5. Run the program by typing Submit on the command line and pressing Enter.

You will see:

IKJ56250I JOB COBJOB(JOB00572) SUBMITTED

 \*\*\*

Press Enter.

You will see the following because of the NOTIFY option of the job statement in the program:

16.18.06 JOB00572 $HASP165 COBJOB ENDED AT N1 **MAXCC=0** CN(INTERNAL)

 \*\*\*

MAXCC=0 means that the maximum condition code is 0.

A condition code greater than 0 means that the program didn’t run properly.

Press Enter to return to the program in edit mode.

To determine the status (ST), use the System Display and Search Facility (SDSF). On the command line, type:

TSO SDSF ST

Press Enter

At the command line, replace acctid with your account ID and type:

OWNER acctid

Press Enter. You will see only your jobs:



In front of the LAST job, type a question mark as shown above and press enter. You will see a screen similar to the one below:



Next, we want to examine the condition codes for the compile, linkedit, and go steps of the JCL. If everything ran correctly, the condition codes will be 0 (zero).

To do so, place an s in front of the file JESYSMSG JES2 and press enter. You will see:



Type FIND COND (or F COND) and press enter. This will take you to the first cond code. Press the F5 key twice to repeat the Find command to find the other condition codes. They should all be 0 (zero).

Press F3 to return to the SDSF DATA SET DISPLAY screen.

To determine if the COBOL executed without errors, we want to look at the contents of the file with the ProcStep COBOL. Type an S in front of this file and press Enter. Go to the bottom of the file (BOT). If the COBOL executed without errors, you will see Return code 0 in the last line. Press F3 to exit this file and return to the SDSF JOB DATA SET DISPLAY.

Press F3 again to return to the SDSF STATUS DISPLAY screen. This is a workspace that you need to manage. To delete (purge) old jobs, place a P in the far left as shown below and then press Enter.

\*\*\* DO NOT delete the rows with Job IDs that begin with TSU. \*\*\*

In the Confirm Action to Delete Data box that appears, press 1 and then the Enter key to process the purge. You may need to press Enter again to see that the file has been purged. If you have more than one file marked with a P, you will have to press enter once for each file.



Type END and press Enter to exit SDSF Status (or press F3).

Type END and press Enter to exit SDSF (or press F3).

To see the output (if it ran correctly!), go to the PDS ACCTID.WORK.OUT. You should see a member named COBOUT. When you edit it, you should see the following:



**Exercises**

Create the following PDSs

ACCTID.WORK.COMPILES

ACCTID.WORK.JCL

ACCTID.WORK.COBOL

ACCTID.WORK.OUT

ACCTID.WORK.DATA

**Exercise 1**

* Copy the SIMPLE.txt file into your ACCTID.WORK.COMPILES
* Open the SIMPLE jobstream (file) and Submit the job
* Using TSO and SDSF, verify that the job ran correctly
* If not, correct and resubmit
* Review your output in your ACCTID.WORK.OUT PDS

**Exercise 2**

Purposely, put a syntax error in the Cobol program—for example, take the V out of one of the MOVE statements.

* Submit the SIMPLE jobstream(file)
* Using TSO and SDSF, verify there is an error an find it
* Correct the error and resubmit the job
* Verify that it now runs correctly
* Review your output in your ACCTID.WORK.OUT PDS

**Exercise 3/ TSO**

* Unwrap the JCL—name it SIMPLE-- and put it into your ACCTID.WORK.JCL PDS

The JCL will need modification to reference the input data and the Cobol program—see below. Use a new name for the output such as COBOUT1

//COBJOB JOB (COBOLPROG),'UOASXXX',NOTIFY=UOASXXX,

// CLASS=A,MSGLEVEL=(1,1),TIME=1,MSGCLASS=A

//STEP1 EXEC PROC=IGYWCLG

//COBOL.SYSIN DD DSN=UOASXXX.WORK.COBOL(SIMPLE),DISP=SHR

//GO.DISKFL1 DD DSN=UOASXXX.WORK.OUT(COBOUT1),DISP=SHR

//GO.READER DD DSN=UOASXXX.WORK.DATA(SMPLDATA),DISP=(OLD,KEEP)

* Place only the COBOL—name it SIMPLE in your ACCTID.WORK.COBOL PDS
* Place the input data into ACCTID.WORK.DATA PDS
* Open the SIMPLE JCL file in your ACCTID.WORK.JCL PDS and submit the job
* Review the output