

JCL1

Orchestrating the Enterprise

- [JCL1 - MAKING THINGS HAPPEN](#)
- [1 LOAD IT UP](#)
- [2 SUBMIT IT](#)
- [3 FILTER AND FIND](#)
- [4 YOU GOT A ZERO. PERFECT!](#)
- [5 JUMP RIGHT TO IT](#)
- [6 MY FIRST COPY JOB](#)
- [7 FIRST ERROR](#)
- [8 KICKING OFF SOME COBOL](#)
- [9 RUN, CODE, RUN](#)
- [10 THEY CAN'T ALL BE ZEROES](#)
- [11 COMPARE THE CODE](#)
- [12 ALL ABOARD](#)
- [13 A FRIENDLY DISPOSITION](#)
- [14 WHAT'S YOUR STATUS?](#)
- [15 YOU'RE NO DUMMY](#)
- [16 RIGHT ON TIME](#)
- [17 WHO KNEW?](#)

JCL1 - MAKING THINGS HAPPEN

The Challenge

You've seen some **JCL** already, but we haven't really gone in depth.

In these challenges, you'll learn a little more about what JCL is used for, why it is important in a Z environment, and how you can take those skills even further.

JCL is an essential part of making things happen in z/OS and getting comfortable with the concepts and syntax will let you power through many challenges you might face as you explore.

Before You Begin

You should have completed the **FILES1** challenge, all about Data Sets and Members. If you have those concepts understood, you're all set to continue with the steps in this challenge.

Investment

Steps	Duration
17	60 minutes

JCL1240630-2221

1 LOAD IT UP

```
≡ ZXP.PUBLIC.JCL(JCLSETUP).jcl
1  //JCLSETUP JOB
2  //      EXEC PGM=IEFBR14
3  //LOAD   DD DSN=&SYSUID..LOAD,DISP=(,CATLG),DATACLAS=SLOAD
4  //JCL    DD DSN=&SYSUID..JCL,DISP=(,CATLG),DATACLAS=SPDS
5  //SOURCE DD DSN=&SYSUID..SOURCE,DISP=(,CATLG),DATACLAS=SPDS
6  //OUTPUT DD DSN=&SYSUID..OUTPUT,DISP=(,CATLG),DATACLAS=SPDS
7
```

Look in **ZXP.PUBLIC.JCL** for a member named **JCLSETUP**.

This is a fairly simple job that will allocate a few new data sets that you need for this, and other challenges.

(Line #4 in the screenshot creates your own JCL data set.)

You can see in Line #3, it mentions **&SYSUID..LOAD**

The Ampersand (**&**) with **SYSUID** after it is what is known as a “Symbolic”, and when the system sees that, it will automatically replace **&SYSUID.** with your Z-userid.

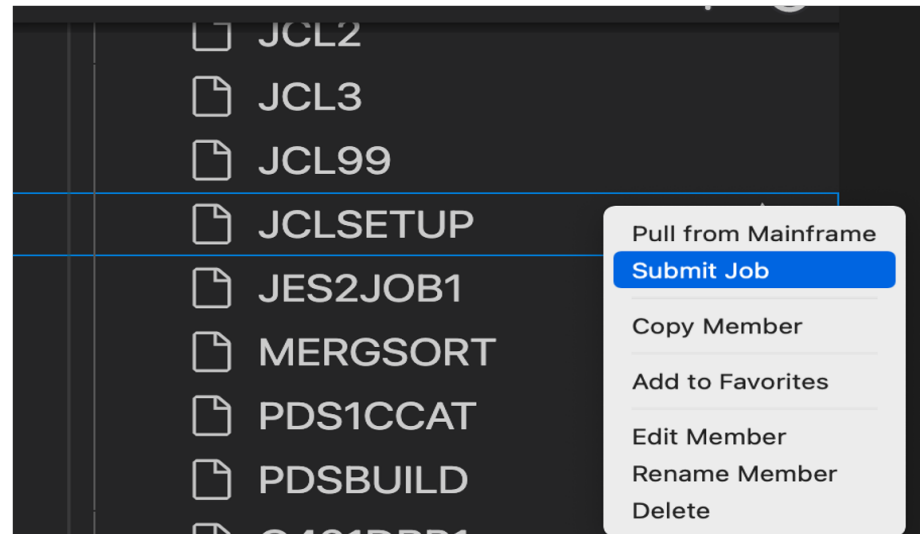
Note the “.” at the end - this marks the end of the symbolic name.

You don’t need to make any changes to this job for it to work.

This means everyone can use the same job, and it will automatically replace **&SYSUID.** with *their* z-userid. How convenient!

JCL1240630-2221

2 SUBMIT IT



Right-click on that job and select **Submit Job**.

A note on the word “Job”: JCL is used to describe to the system exactly what you want it to do.

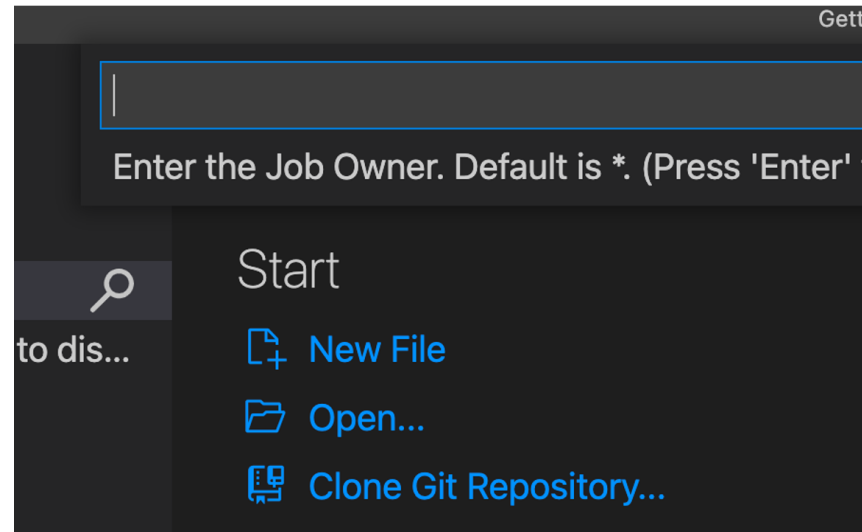
The task which we hand over to the system is known as a “Job”, and the component of z/OS that accepts, processes, and manages the output of these jobs is known as the **Job Entry Subsystem (JES)**.

So, for this challenge, you sent a job to **JES** for it to process the task that you just looked at.

Note: this job is intended to create datasets for you, and it assumes you do not already have those datasets; *if you run it more than once, you are likely to see errors about **DUPLICATE** dataset names.*

JCL1240630-2221

3 FILTER AND FIND



You should already have a profile under **JOB** on the left side of VSCode.

Click on the magnifying glass () to the right of it.

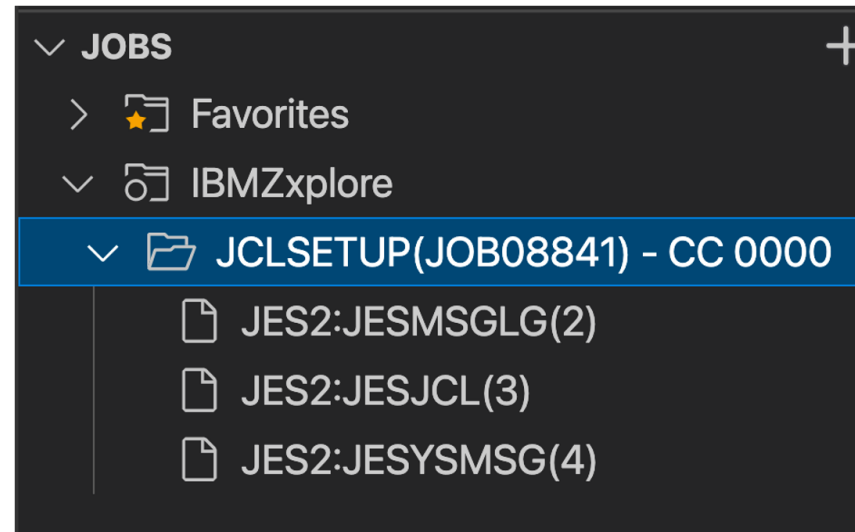
- Enter your userid for the Job Owner
- Enter an asterisk (*) for the Job Prefix
- Hit Enter (blank, no data) for Job Id Search.

You can redo this filter by clicking on the magnifying glass again, and selecting Owner/Prefix or Job Id. You should be able to find the job you just submitted in here. Look for **JCLSETUP**.

The next step will dig into that in more detail.

JCL1240630-2221

4 YOU GOT A ZERO. PERFECT!



Open up the triangle “twistie” next to the **JCLSETUP** job you just submitted. There will probably be other jobs in there as well, but you are specifically looking for **JCLSETUP**. If you submitted it more than once, find the one with **CC 0000** to the right.

You will also see this number in the **JESMSGGLG** member once you open the twistie. A condition code (**CC**) of zero means everything ran as expected, with no errors, so that’s good!

If you got any other number for the completion code, then there is usually something worth investigating - and probably fixing.

JCL1240630-2221

5 JUMP RIGHT TO IT

```
JOB08841  -STEPNAME PROCSTEP    RC    EXCP
JOB08841  -                      00      1
JOB08841  -JCLSETUP ENDED.  NAME-
JOB08841  $HASP395 JCLSETUP ENDED - RC=0000
ES2 JOB STATISTICS -----
2022 JOB EXECUTION DATE
        6 CARDS READ
```

You may have noticed that after submitting `JCL` in `VSCode`, a little message pops up in the bottom right corner of the `VSCode` window.

Rather than digging through your `JOBS` output, you can usually just click on that message, and it will take you right to the output.

A job will start out in `ACTIVE` while it is being executed.

You can refresh the status of a job by closing and then re-opening the twistie to the left of the job name.

JCL1240630-2221

WHY ARE JES AND JCL IMPORTANT? WHY CAN'T I JUST RUN PROGRAMS?

When you submit **JCL**, it goes to the Job Entry Subsystem (shortened to **JES**).

JES looks through the **JCL** you have submitted and gathers all of the resources needed to accomplish the task. On a heavily-loaded system, it may be necessary to prioritize some jobs lower or higher than others so that important work gets done faster.

Think of **JCL** as the order that a waiter writes up, and **JES** as the kitchen staff that looks at the order and decides how they're going to handle it.

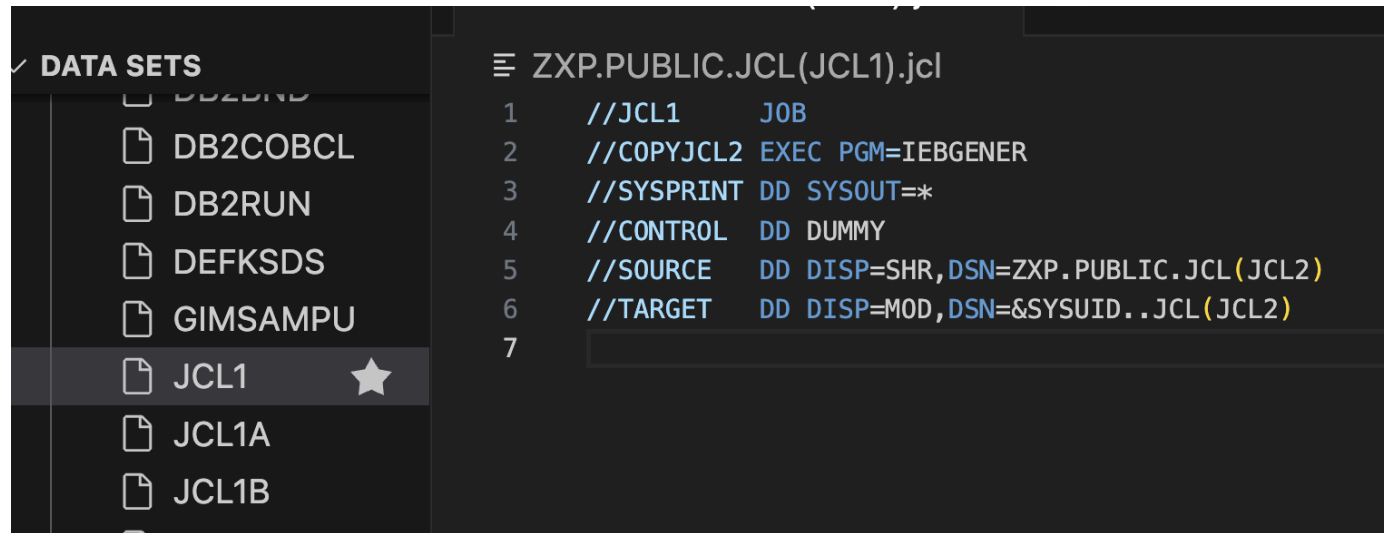
The **L** in **JCL** stands for Language, but it really isn't a programming language as much as it is a way for us to effectively describe tasks to the system.

Everything else that shows up in the job output (the "joblog") is information about how the job ran. As you can see, there's a lot of information in here.

JCL1240630-2221

6 MY FIRST COPY JOB

Now you have some more datasets to work with, and with this being a **JCL** challenge, you need to get some of your own **JCL** to work on.



The screenshot shows the IBM Z Explorer interface. On the left, under the 'DATA SETS' section, a list of datasets is displayed: DB2COBCL, DB2RUN, DEFKSDS, GIMSAMPU, JCL1 (highlighted with a star), JCL1A, and JCL1B. On the right, the JCL code for 'ZXP.PUBLIC.JCL(JCL1).jcl' is shown, consisting of seven lines of code:

```
1 //JCL1 JOB
2 //COPYJCL2 EXEC PGM=IEBGENER
3 //SYSPRINT DD SYSOUT=*
4 //CONTROL DD DUMMY
5 //SOURCE DD DISP=SHR,DSN=ZXP.PUBLIC.JCL(JCL2)
6 //TARGET DD DISP=MOD,DSN=&SYSUID..JCL(JCL2)
7
```

Copy the **JCL1** member from **ZXP.PUBLIC.JCL** to your own **JCL** data set, and then open your copy. You *need* to have this in your own JCL dataset as you will be modifying it over the next few steps.

You may need to close and re-open your **DATA SETS** triangle to refresh the view, so your **JCL1** shows up.

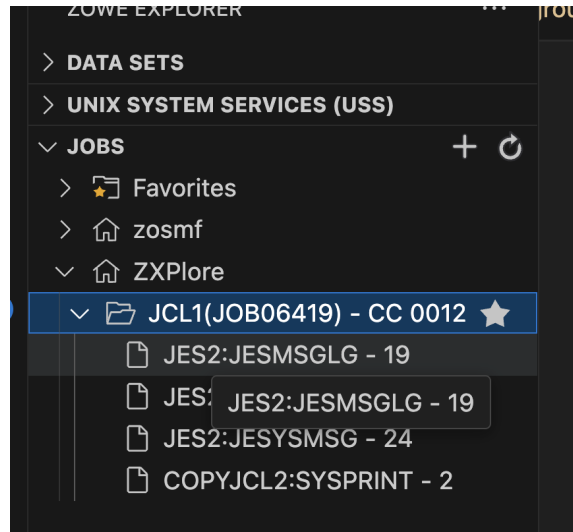
This job is provided so you can copy the **JCL2** member from **ZXP.PUBLIC.JCL** into your **JCL** dataset, but using the **IEBGENER** utility program instead of **VSCODE**.

Submit JCL1 in the same way you submitted the **JCLSETUP**, and look at the joblog.

JCL1240630-2221

7 FIRST ERROR

When you look at the **JOBS** section and see your **JCL1** job, you should see straight away that the completion code is **0012**; assume that means something went wrong.



Open the **JES2:JESMSGLG** section of the job and check for an indication of what caused the failure.

```

1      1                      J E S 2  J O B  L O G  --  S Y S T E M
2      ✓ 0
3      08.13.04 JOB06419 ---- TUESDAY, 07 NOV 2023 ----
4      08.13.04 JOB06419 IRR010I USERID Z##### IS ASSIGNED TO
5      08.13.05 JOB06419 ICH70001I Z##### LAST ACCESS AT 08:02:
6      08.13.05 JOB06419 $HASP373 JCL1          STARTED - INIT 1  -
7      08.13.05 JOB06419 IEC130I SYSIN  DD STATEMENT MISSING
8      08.13.05 JOB06419 -
9      08.13.05 JOB06419 -STEPNAME PROCSTEP      RC      EXCP      CONN
10     08.13.05 JOB06419 -COPYJCL2                12       10       0
11     08.13.05 JOB06419 -JCL1          ENDED.    NAME-
12     08.13.05 JOB06419 $HASP395 JCL1          ENDED - RC=0012
13     0----- JES2 JOB STATISTICS -----
14     - 07 NOV 2023 JOB EXECUTION DATE
15     -              6 CARDS READ
16     -              53 SYSOUT PRINT RECORDS
17     -              0 SYSOUT PUNCH RECORDS
18     -              3 SYSOUT SPOOL KBYTES
19     -              0.00 MINUTES EXECUTION TIME
20

```

In this case, you can see that the problem is due to a missing **DD** statement for **SYSIN**

In **JCL**, statements that define where data is coming from, or going to, are known as **Data Definition** statements, or simply, “**DD** statements”.

In this job there are only two steps - both execute the **IEBGENER** program; if you search online, you would eventually find the following information about the setup for **IEBGENER**:

<https://www.ibm.com/docs/en/zos/3.1.0?topic=c-job-control-statements-4>

File/DD	Purpose
SYSPRINT	Defines a sequential data set for messages. The data set can be written to a system output device, a tape volume, or a DASD volume
SYSUT1	Defines the input data set. The “source”. It can reference a existing sequential data set, member of a partitioned data set or PDSE or a z/OS UNIX file. A sequential data set can be basic format, large format, extended format, compressed format, spooled input, tape, card reader, TSO terminal or DUMMY. Note

File/DD	Purpose
	that the source must exist - if it does not, IEBGENER will end with a 013 error.
SYSUT2	Defines the output data set. The “target”. It can define a sequential data set, a member of a partitioned data set or PDSE, a partitioned data set or PDSE or a z/OS UNIX file. A sequential data set can be basic format, large format, extended format, compressed format, spooled output, tape, card punch, printer, TSO terminal or DUMMY
SYSIN	Defines the control data set, or specifies DUMMY when the output is sequential and no editing is specified. The control data set normally resides in the input stream; however, it can be defined as a member in a partitioned data set or PDSE.

Hopefully, you can see that the files assigned for **IEBGENER** in the **JCL1** job, using the DD statements, do not match the files **required** by the program.

Make appropriate changes to the DD statements in your JCL1 member (**remember to save!**) and submit again.

If you made the correct changes, the job should complete with CC=0000.

If not, check the joblog again for messages that indicate the cause of the new errors; make the necessary adjustments and try again ...

With this type of program, working out what went wrong with things like incorrect or missing DD statements is easy to identify and fix – it’s in the book!

Once you have a successful completion of the JCL1 job, you should have a new **JCL2** member in your **JCL** dataset.

(You will also have a **JCL3** member that is created by the second step of the **JCL1** job)

JCL1240630-2221

8 KICKING OFF SOME COBOL

```
1  //JCL2    JOB 1
2  //*****
3  //COBRUN   EXEC IGYWCL
4  //COBOL.SYSIN DD DSN=ZXP.PUBLIC.SOURCE(CBL0001),DISP=SHR
5  //LKED.SYSLMOD DD DSN=&SYSUID..LOAD(CBL0001),DISP=SHR
6  //*****
7  // IF RC = 0 THEN
8  //*****
9  //RUN      EXEC PGM=CBL0001
10 //STEPLIB  DD DSN=&SYSUID..LOAD,DISP=SHR
11 //FNAMES   DD DSN=ZXP.PUBLIC.INPUT(FNAMES),DISP=SHR
12 //LNAMES   DD DSN=ZXP.PUBLIC.INPUT(LNAMES),DISP=SHR
13 //COMBINE  DD DSN=&SYSUID..OUTPUT(NAMES),DISP=SHR
14 //SYSOUT   DD SYSOUT=*,OUTLIM=15000
15 //CEEDUMP  DD DUMMY
16 //SYSUDUMP DD DUMMY
```

Edit your personal copy of the **JCL2** job definition.

You may need to close and re-open your **DATA SETS** triangle to refresh the view, so your **JCL2** shows up.

This JCL is used to compile and run some **COBOL** code. After compiling, it will put the resulting program in your **LOAD** data set.

Note: the programs in the LOAD dataset are binary - you won't be able to view anything in here with VSCode.

Look for the line that begins with **//COBRUN** - this is the start of a job “step” that will run the COBOL compiler.

On the next line, you can see the input data set (the source) on Line 18 (**//COBOL.SYSIN**), and where it will put the output on the following line (**//LKED.SYSLMOD**).

The lines following the start of the **//RUN** step have a similar format:

```
//ddname DD DSN=dataset,DISP=access
```

- “ddname” is also known as the filename - the name used by programs to access data in datasets

- “dataset” is the actual location of the data - this can change, but the program doesn’t need to be aware
- “access” (or “disposition”) states how the program can use the dataset

Reading further, if the jobstep finishes with a Completion Code of 0 (because there were no problems) from the compile step, it will then run the **CBL0001** program.

All making sense so far? You will be using JCL to compile **COBOL** source code, and then run the resulting program.

WHAT DOES COMPILE MEAN? WHAT IS COBOL?

COBOL is a programming language used in many financial, healthcare, and government institutions. Its high degree of mathematical precision and straightforward coding methods make it a natural fit when programs need to be fast, accurate, and easy to understand.

Code that gets written by humans needs to be turned into Machine Code for it to run as a program. Compiling is a step that performs this transformation. Our **JCL** has two main steps, compiling the source code into machine code, and then running the program.

This particular program also requires two input files, and writes to an output file, so we will specify those files (data sets) in the **JCL** as well.

JCL1240630-2221

9 RUN, CODE, RUN

After successfully compiling the COBOL code, JES will run the program (the `//RUN EXEC PGM=CBL0001` command) and tell it where to find the input data sets, as well as where the output will be stored in your OUTPUT dataset –

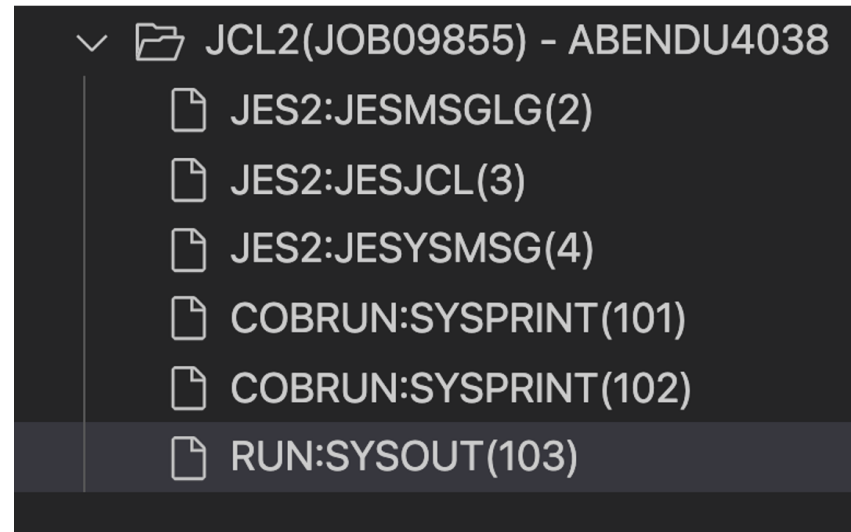
```
//COMBINE DD DSN=&SYSUID..OUTPUT(NAMES),DISP=SHR
```

The name of the DD statement is what comes directly after the double slashes, so “FNames” and “LNames”, for example.

Any lines starting with `//*` are comments and are ignored by JES. Commented lines are helpful for providing informational information, or holding lines of code we might use later, but don’t need right now.

JCL1240630-2221

10 THEY CAN'T ALL BE ZEROES



Submit **JCL2** from your JCL data set and then look at the output, using what you learned from the earlier steps in this challenge.

NOTE: You *will* get an **ABEND** (short for Abnormal End), so something isn't quite right yet.

But don't worry - with your new skills, you will get to the bottom of this!

In the previous steps for the **JCL1** job, you could use the documentation for **IEBGENER** to determine what DD statements were *required* for the program to work; in this case, there is no documentation - only the COBOL code itself.

In the next step, you will look at the COBOL code and see how the actual code matches up with the JCL code being used to compile and execute it.

And again, don't worry! You do not need to become a **COBOL** expert to resolve this - remember this is a **JCL** challenge, not a **COBOL** challenge.

JCL1240630-2221

For **z/OS** programs to work with datasets in a job, they need 4 things to be in place:

1. the internal definition of the file that represents the data structure and content that the program will create, read, modify, or delete – this is defined inside the program
2. the current dataset that a particular execution of the program can use – each time the program is executed it can use different datasets, as long as they have the correct format that the program expected; this is the name used in the **DSN=** parameter of a DD statement
3. a correct choice of disposition for the dataset to indicate whether it should be created, deleted, passed on - this is the ****DISP=**** parameter of a DD statement
4. a DD statement that links the program's internal file to the particular dataset – this must match the name of the program's file definition, and specify the required dataset, and the disposition

JCL1240630-2221

11 COMPARE THE CODE

```
*-----  
IDENTIFICATION DIVISION.  
*-----  
PROGRAM-ID.    NAMES  
AUTHOR.        Otto B. Named  
*-----  
ENVIRONMENT DIVISION.  
*-----  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    SELECT FIRST-NAME ASSIGN TO FNAMES.  
    SELECT LAST-NAME  ASSIGN TO LNAMES.  
    SELECT FIRST-LAST ASSIGN TO COMBINED.
```

The JCL statement beginning `//COBOL.SYSIN` points to the COBOL source code dataset it will compile, so start there. Open up that program source code in `VSCode`, and start by looking at the `FILE-CONTROL` area.

This is where you get the names used by the program which you need to match in the JCL. For example, `FIRST-NAME` is a record reference in the `COBOL` code for a file, and that is assigned (or linked to) the `FNAMES` `DD` statement in the JCL.

Open up the `JCL`, `COBOL` code, and the joblog – look at the output, and you should be able to see what

*very simple **single letter** change*

needs to be made to the `JCL2` job in order for everything to link together between the `COBOL` program, the datasets, and the JCL.

When you have fixed the problem in `JCL2`, it should run with a `CC=0000`, and you will *find the correct output in the correct member* of your `OUTPUT` data set.

12 ALL ABOARD

```
//*  
//PEEKSKL EXEC PGM=IEBGENER  
//SYSPRINT DD DUMMY  
//SYSIN DD DUMMY  
//SYSUT1 DD *  
Peekskill - 41mi  
//SYSUT2 DD DSN=&SYSUID..JCL3OUT,DISP=(MOD,PASS,DELETE),  
//          SPACE=(TRK,(1,1)),UNIT=SYSDA,  
//          DCB=(DSORG=PS,RECFM=FB,LRECL=80)  
//*  
//CORTLNDT EXEC PGM=IEBGENER  
//SYSPRINT DD DUMMY  
//SYSIN DD DUMMY  
//SYSUT1 DD *  
Cortlandt - 38mi
```

Open the

JCL3 member of your JCL dataset and take a look at what's inside. You'll see this JCL contains a number of steps, with each one using the **IEBGENER` utility program to direct one or more records into a sequential data set.**

This is a pretty simple extension of the **JCL1** job you worked through earlier – one big difference here being how the “source” data is defined.

In this job, the data for all **SYSUT1** data definitions is defined as “in-stream” - it is right there in the job after the DD statement. This is specified by the **DD *** option instead of a **DSN=** parameter. The **IEBGENER** program will copy input from **SYSUT1** until the next JCL statement is reached – beginning with either **//** or **/***

You will see that there is a header line, some lines with station information for the peak train stops between Poughkeepsie, NY and Grand Central Terminal in NYC, followed by some text about operating hours.

Looks pretty straightforward ... what is the tricky part going to be?

13 A FRIENDLY DISPOSITION

```
...T,DISP=(MOD,PASS,DELETE),  
UNIT=SYSDA,  
...=FB,LRECL=80)
```

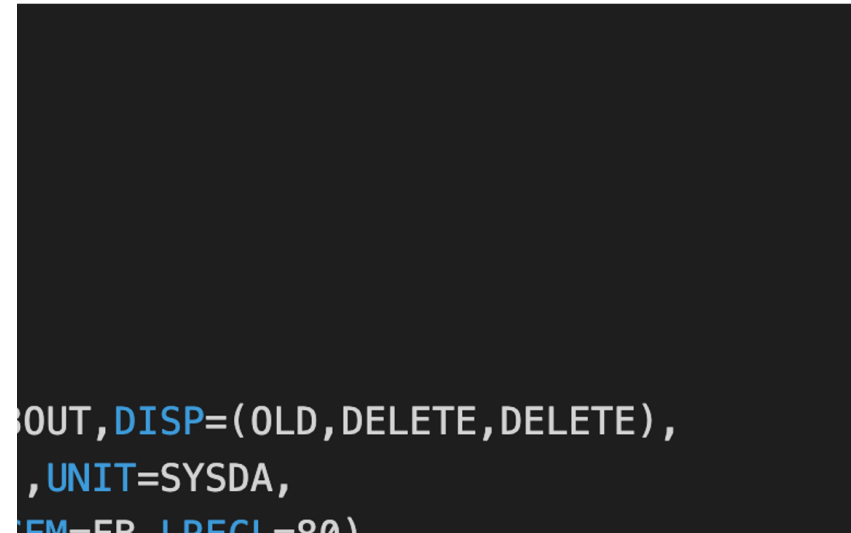
In every piece of JCL you have used so far, you will have encountered some sort of **DISP=** (disposition) parameter.

DISP parameters are used to describe how JCL should use or create a data set, and what to do with it after the job, or job step, completes.

A standard **DISP** parameter has three parts. The first parameter is the status, which can be any of the following:

Parameter	Meaning
NEW	Create a new data set
SHR	Re-use an existing data set, and let other people use it if they'd like
OLD	Re-use an existing data set, but don't let others use it while we're using it
MOD	For sequential data sets only. Re-use an existing data set, but only append new records to the bottom of it. If no data set exists, create a new one.

14 WHAT'S YOUR STATUS?



Field 2 of the `DISP` parameter describes what should happen to the dataset in the case of a normal jobstep completion, and the third field is what should happen to the dataset in the case of a jobstep failure.

There are a number of values that can be used here, but for this challenge, you only need to know about the following:

Field2	Meaning
DELETE	Erase it from storage completely
CATLG	Record the data set so you can use it after the job finishes
PASS	After this step completes, hold on to it so jobsteps that come after this (in the same job) can use it

15 YOU'RE NO DUMMY

```
//JCL3      JOB
//*
//* IEBGENER is a system utility program to copy data
//* where the default input filename is SYSUT1
//* and the default output filename is SYSUT2
//*
//HEADER EXEC PGM=IEBGENER
//SYSPRINT DD DUMMY
//SYSIN     DD DUMMY
//SYSUT1    DD *

*****
METRO NORTH POUGHKEEPSIE -> NYC M-F SCHEDULE
PEAK HOUR OPERATION
*****
```

You should noticed lots of mentions of **DUMMY** in the DD statements.

Don't worry, this JCL isn't calling anyone names; it's just a way of saying "This file allocation is required, but this time it isn't going to be used for anything, so it doesn't matter".

As you saw earlier, the **IEBGENER** program running in each step requires:

- **SYSIN** an input file (DD) statement for control commands
- **SYSPRINT** an output DD statement for the program to report success, failure, progress
- **SYSUT1** a "source" DD statement where data is to be copied from
- **SYSUT2** a "target" DD statement where the data from SYSUT1 is to be copied to

But in this job, the output from the program is not needed, and there are no control statements required, so DUMMY is a way of saying "It doesn't matter, don't waste your time setting this up" – just take the default action and copy the dataset associated with **SYSUT1** into the dataset associated with **SYSUT2**.

JCL1240630-2221

16 RIGHT ON TIME

Submit your copy of the **JCL3** job and look at the output.

NOTE: you should expect this job to complete with code 0000 – that means nothing is broken - it does not mean that the output is correct!

There are two edits you need to make in order for this job to run 100% correctly:

- a full listing of **all 10 station stops**, from Poughkeepsie to Grand Central Terminal
- the information at the top and bottom of the data set.
- You should have **no repeat stops**. If Beacon or Cortlandt is listed in there twice, something still needs fixing.

NOTE: You will need to delete the **JCL3OUT** output data set each time before re-submitting **JCL3**

Optionally, check back through the JCL for the **JCLSETUP** and **PDSBUILD** jobs to see how you might automatically delete **JCL3OUT** by adding an extra step at the start of **JCL3**.

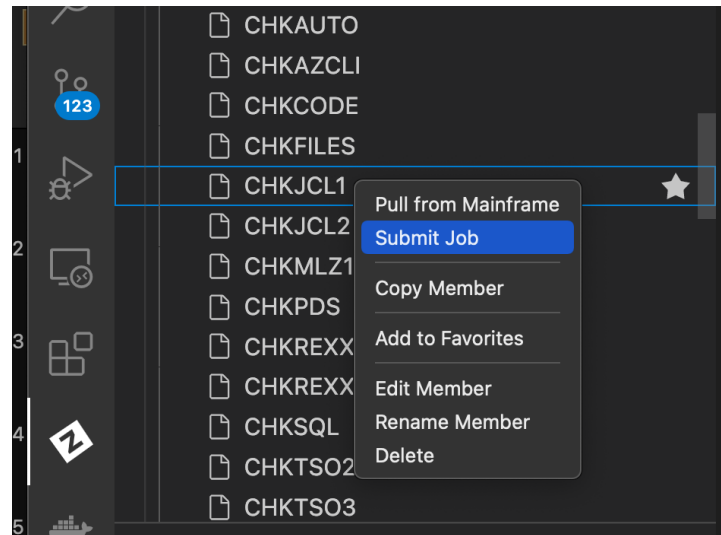
Submit **JCL3** again and check to see if you have the correct output.

When completed, you should have the full output in your **JCL3OUT** sequential data set, totaling 23 lines (records) – the same as the following view:

JCL1240630-2221


```
1 *****
2 METRO NORTH Poughkeepsie -> NYC M-F SCHEDULE
3 PEAK HOUR OPERATION
4 *****
5 Poughkeepsie - 74mi
6 New Hamburg - 65mi
7 Beacon - 59mi
8 Cold Spring - 52mi
9 Garrison - 50mi
10 Peekskill - 41mi
11 Cortlandt - 38mi
12 Croton-Harmon - 33mi
13 Harlem - 125th Street - 4mi
14 Grand Central Terminal - 0mi
15 *****
16 Peak fares are charged during business rush hours on any
17 weekday train scheduled to arrive in NYC terminals between
18 6 a.m. and 10 a.m. or depart NYC terminals between 4 p.m.
19 and 8 p.m. On Metro-North trains, peak fares also apply to
20 travel on any weekday train that leaves Grand Central Terminal
21 between 6 a.m. and 9 a.m.
22 Off-peak fares are charged all other times on weekdays, all
23 day Saturday and Sunday, and on holidays.
```

17 WHO KNEW?



Now submit the job **CHKJCL1** from **ZXP.PUBLIC.JCL** to validate the correct output from **JCL2** and **JCL3**, and hope to see completion code (**CC**) of 0000.

If **CHKJCL1** returns **CC=0127**, go back and double-check and adjust your work for JCL2 and JCL3, and submit those jobs again if needed.

Recheck for the correct output by submitting **CHKJCL1** again until you get CC=0000.

AGAIN: You will need to make sure the **JCL3OUT** output dataset is deleted before re-submitting **JCL3**

You've accomplished a lot, and hopefully understand the requirement for following the details ... Well done!

JCL1240630-2221

Nice job - let's recap	Next up ...
<p>JCL can seem a little tricky, and maybe even a little unnecessary at first. We're not used to using code to start programs, we usually just double-click on them and they run! However, once you start getting into the types of applications that keep Z systems busy 24/7, you start to build an appreciation for the precision and power that the structure gives you. Suffice to say, JCL is a necessary skill for any true Z professional.</p>	<p>Check out DfsMS Utilities for other common “utilities” used in JCL for managing datasets, and other storage systems.</p> <p>Discover the Unix environment inside zOS – Unix System Services (USS)</p>

JCL1240630-2221