



# JCL

## Chapter a3

### Coding EXEC statements

## **Job Control Language**

**Chapter a1. Introduction to JCL**

**Chapter a2. Coding JOB statements**

**Chapter a3. Coding EXEC statements**

**Chapter a4. Coding DD statements**

**Chapter a5. Analyzing job output**

**Chapter a6. Conditional processing**

## **Job Control Language**

**Chapter b1. Using special DD statements**

**Chapter b2. Introducing procedures**

**Chapter b3. Modifying EXEC parameters**

**Chapter b4. Modifying DD parameters**

**Chapter b5. Determining the effective JCL**

**Chapter b6. Symbolic parameters**

## **Job Control Language**

**Chapter c1. Nested procedures**

**Chapter c2. Cataloging procedures**

**Chapter c3. Using utility programs**

**Chapter c4. Sample utility application**

# **Chapter a3**

## **Coding EXEC statements**

## Coding EXEC statements

### Course objectives.

#### Be able to:

- **Code an EXEC statement to specify a program to be executed.**
- **Correct coding errors in an EXEC statement.**
- **Identify which JCL statement has caused a "PROGRAM NOT FOUND" error message.**
- **Identify the system library from which programs are retrieved at execution time.**
- **Identify the DD statement names used to specify a private library from which programs are retrieved at execution time.**
- **Select the place in the job stream where STEPLIB and JOBLIB DD statements should be located.**
- **Code a JOBLIB DD statement.**

## The EXEC statement

### The EXEC statement.

To execute each program in a job, you need to code one EXEC statement. Each job step begins with an EXEC statement that identifies a program name.

The EXEC statement is used to invoke the program that you want to execute as part of a job. In addition to this, you can also use the EXEC statement to invoke a cataloged procedure.

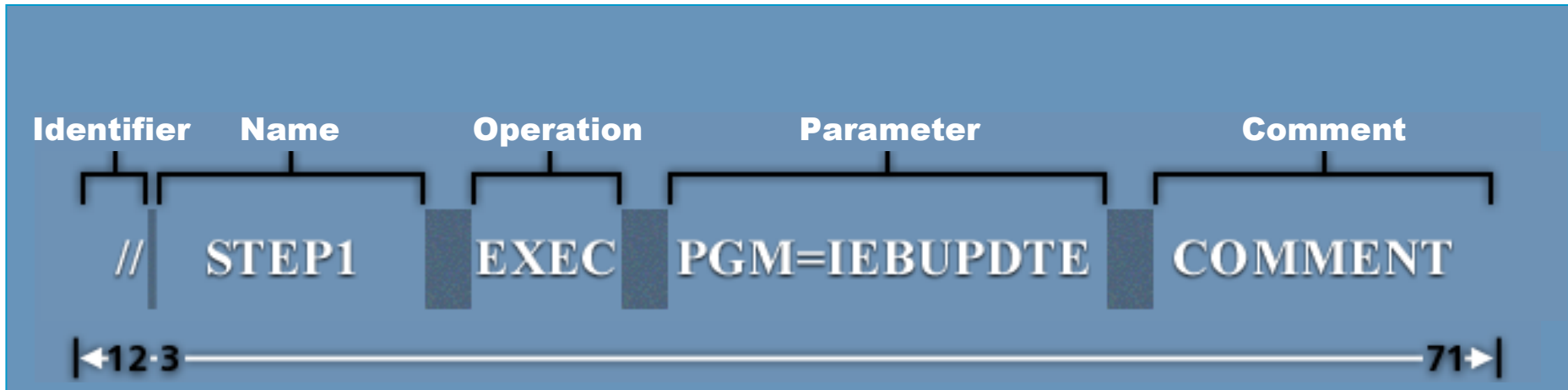
### What is a cataloged procedure?

A cataloged procedure refers to a set of JCL statements that are stored in a library and retrieved by its name.

A procedure may contain one or more EXEC statements.

## The EXEC statement

### The EXEC statement.



Like the JOB statement, the EXEC statement too has five fields. The EXEC statement format includes the following:

- Identifier field (//): It occupies position 1 and 2.
- Name field: It names the step starting in position 3.
- EXEC operator field: It states the JCL statement type.
- Parameter field: It is used to state the parameters used on an EXEC statement.
- Comment field: This field is optional.



## The EXEC statement

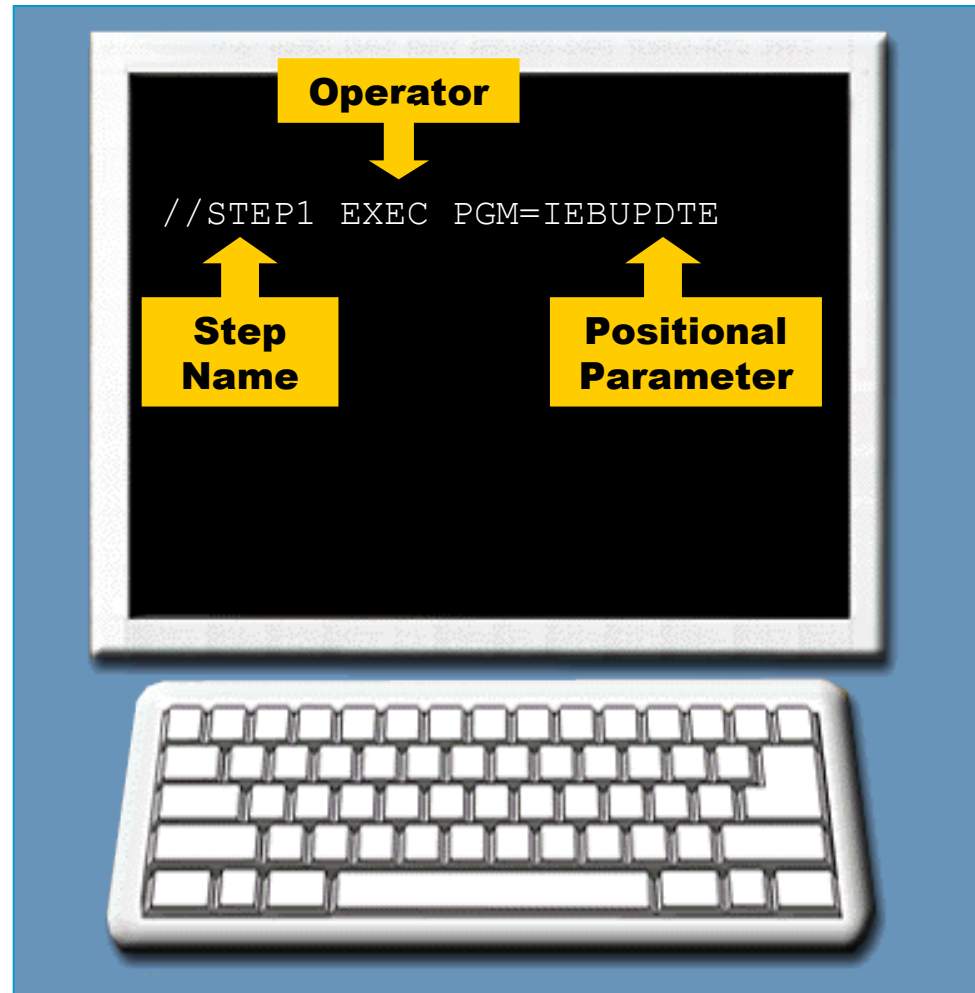
### The EXEC statement.

Shown here is an example where the step name is STEP1. The operator is EXEC and the positional parameter is defined by PGM=IEBUPDTE.

### What is IEBUPDTE?

IEBUPDTE is a system utility program that the system invokes during the execution of STEP1.

The step name STEP1 identifies the EXEC statement so that the subsequent JCL statements can refer to it.



## The EXEC statement

### The step name.

Following are the coding rules for the step name:

- The step name must begin in position 3.
- The step name must be 1 to 8 characters in length.
- The first character in the step name should be either alphabetic or a national symbol. It cannot be a number.
- Rest of the characters in the step name can either be alphanumeric or they can be national symbols.
- Special characters and spaces cannot be used in a step name.

#### Valid Step Names

```
//STEP1 EXEC
```

```
//EXAMPLE4 EXEC
```

```
//RUN#2 EXEC
```

#### Invalid Step Names

```
//STEP1+ EXEC (Includes a special character)
```

```
//EXAMPLE14 EXEC (More than eight characters)
```

```
// RUN#2 EXEC (Does not begin in position 3)
```

## The EXEC statement

### The step name.

Shown here are two step names  
STEP#FOUR and LA\$JOE.

### Are both the names valid step names?

The step name LA\$JOE is acceptable because it fits all the requirements defined in the rules for coding a step name (the \$ is one of the national symbols).

But, STEP#FOUR is not a valid step name because it contains more than eight characters.



The EXEC statement.

Are we on track?

Which of the following step names are valid?

- A. \$STEP#5
- B. RUN TWO
- C. \*STEP4
- D. EXAMPLE#12

**The EXEC statement.**

**Glossary.**

**Cataloged procedure**

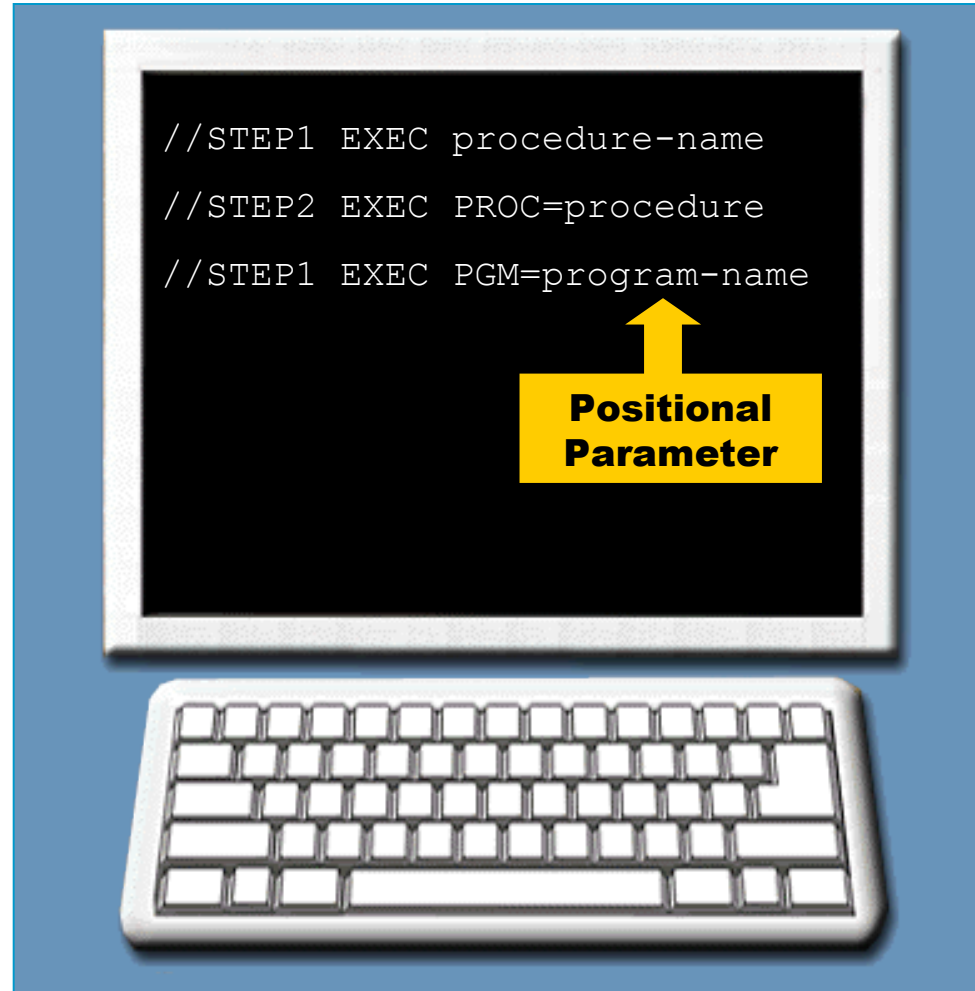
**Prepared sets of JCL statements cataloged in a procedure library.**

## The EXEC parameter field.

### The positional parameter.

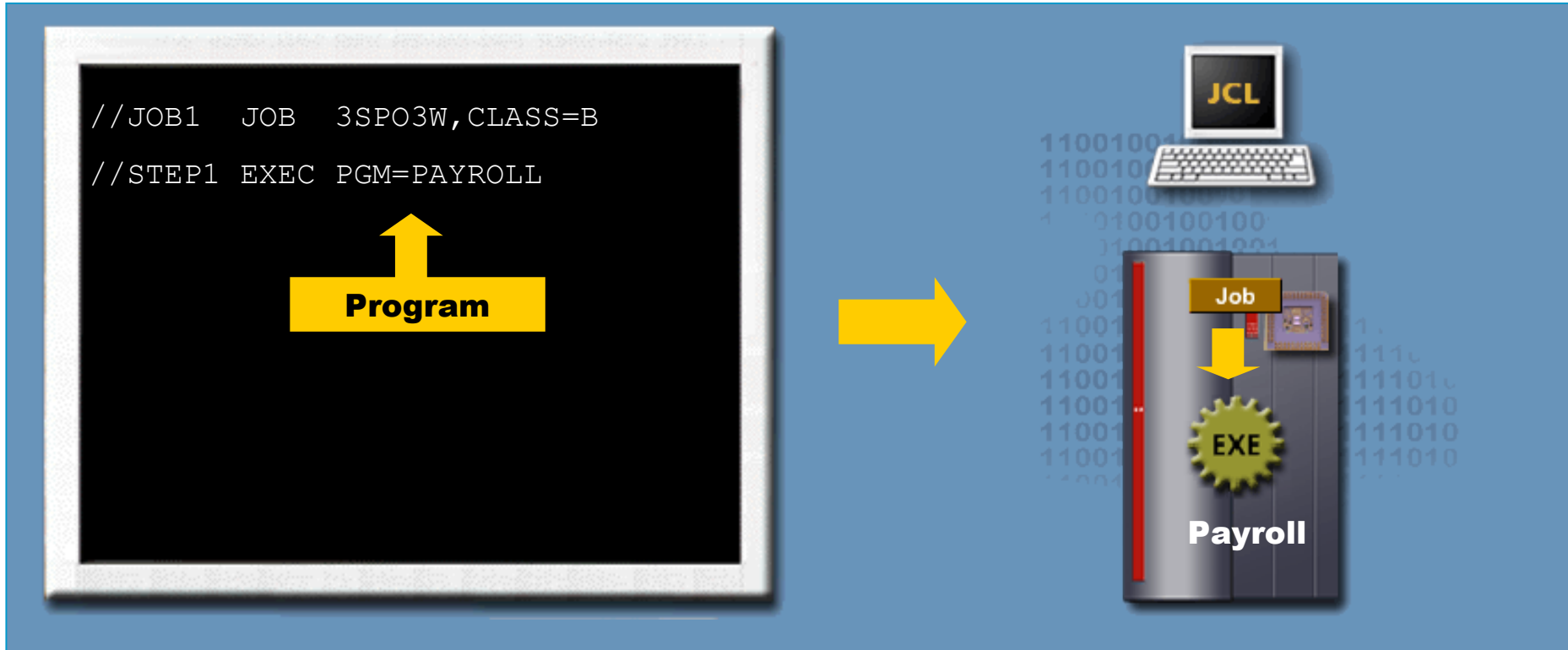
The parameter field follows the EXEC operator and may contain multiple parameters. The first parameter in an EXEC statement is a positional parameter that designates the program or procedure the system executes during the job step.

This positional parameter is often coded like a keyword parameter using either PGM= or PROC=.



## The EXEC parameter field.

## The PGM= positional parameter.



## What does PGM= designate?

PGM= designates a program the system executes during the job step. Shown here is the syntax of PGM as a positional parameter in the EXEC statement.

The EXEC parameter field.

**Are we on track?**

**Code the EXEC statement using the proper operator where the step name is STEP1 and the name of program is PAYROLL?**

**//JOB1      JOB            255,SMITH**



## The EXEC parameter field.

## The PROC= positional parameter.

```
//LA$JOE JOB 3SPO3W,CLASS=B  
//STEP1 EXEC PROC=MYPROC
```

**Procedure**



## What does PROC= designate?

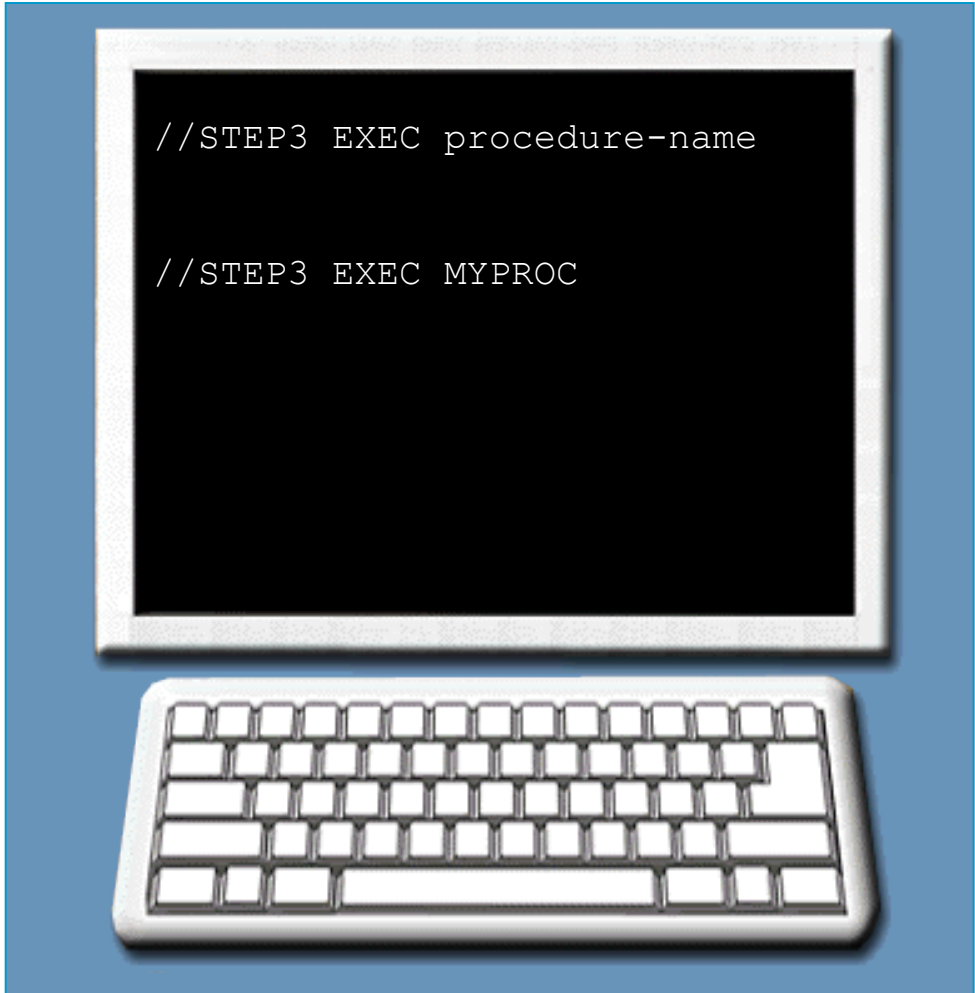
PROC= designates a procedure the system executes. The syntax of PROC as a positional parameter in an EXEC statement is shown here.

## The EXEC parameter field.

### Omitting the PROC= keyword.

If you omit the PGM= or PROC= keyword, the operating system automatically looks for a procedure by the specified name. Shown here is an example of the syntax when omitting PROC= keyword in an EXEC statement is shown.

For example, to call a procedure named MYPROC from a step named STEP3, you will need to code the EXEC statement as shown on the right.



```
//STEP3 EXEC procedure-name  
  
//STEP3 EXEC MYPROC
```

**The EXEC parameter field.**

**Are we on track?**

**If you leave out the PGM= or PROC= keyword when coding an EXEC statement, what does the operating system do automatically?**

- A. Searches for a procedure with the specified name.**
- B. Searches for a program with the specified name.**
- C. Searches all programs and procedures with the specified name**
- D. Displays an error and cause an abnormal termination.**

## The EXEC parameter field.

# Positional parameter coding errors.

JCL errors occur if you, as a JCL programmer fail to follow any of the coding rules regarding the PGM and PROC positional parameters.

For example, the misspelling of PGM (as PGR) in the EXEC statement shown here returns a JCL error.

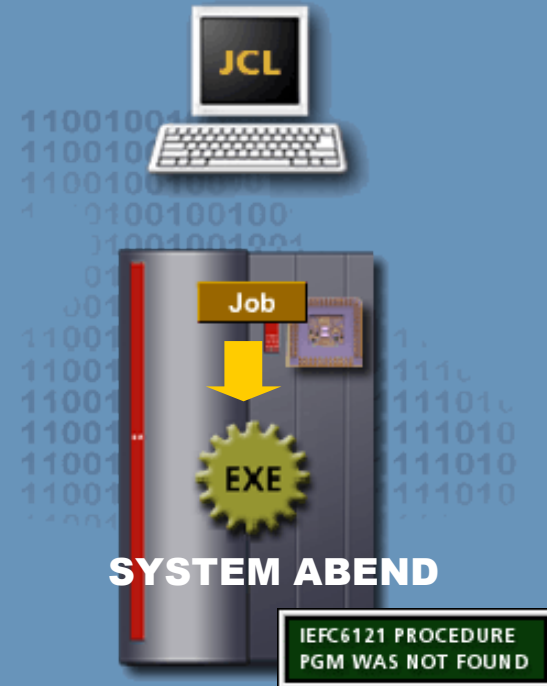
A computer monitor with a white bezel is shown against a blue background. The screen is black and displays two lines of JCL code in white text. The first line is '//LA\$JOE JOB 3SPO3W,CLASS=B' and the second line is '//STEP1 EXEC PGR=IEFBR14'. The word 'PGR' in the second line is highlighted in yellow. Below the monitor is a white keyboard with black keys.

```
//LA$JOE JOB 3SPO3W,CLASS=B
//STEP1 EXEC PGR=IEFBR14
```

## The EXEC parameter field.

### Positional parameter coding errors.

```
//LA$JOE JOB 3SPO3W,CLASS=B  
//STEP1 EXEC PGM = IEFBR14
```



In the example shown here, the equal sign in the parameter field is preceded and followed by a space. The system interprets "PGM" as a procedure name and not as a keyword for the positional parameter of the EXEC statement.

## Additional EXEC parameters.

### Procedures.

**In addition to programs, you can use the EXEC statement to invoke a cataloged procedure, which is a set of JCL statements that you place in a library and retrieve by its name.**

**A procedure can contain one or more EXEC statements, with associated DD statements.**

**Additional EXEC parameters.**

**Are we on track?**

**A \_\_\_\_\_ is a set of associated EXEC and DD statements stored in a procedure library.**

## Additional EXEC parameters.

### Keyword parameters.

You may code keyword parameters on the EXEC statement in any order, following the program or procedure name.

You can use any of the keyword parameters shown here.

If you code one of these keyword parameters on the EXEC statement, the keyword parameter value will apply only to that step.

The two keyword parameters used most frequently with the EXEC statement are:

- The PARM parameter.
- The COND parameter.

PGM=PROGA, keyword



ACCT

RD

DYNAMNBR

ADDRSPC

PERFORM

REGION

PARM

COND

TIME

DPRTY

CCSID



**The PARM parameter.**

**The PARM parameter.**

**What does the PARM parameter do?**

**The PARM parameter passes information to the executing program. Some programs accept information from the PARM parameter about how many times to execute.**

**For example, a program may need to know whether a report cycle is "annual" or "monthly".**

**The records the program uses vary depending on which value is passed to it.**

**Similarly, the PARM parameter can supply a password to the program that is required before the program executes.**

**The syntax for the PARM parameter is:**

**PARM=(SUBPARAMETER,SUBPARAMETER)**

## The PARM parameter.

### Rules for coding the PARM parameter.

The general syntax and rules for coding the PARM parameter are:

- **The PARM parameter can include up to 100 characters.**
- **The PARM parameter can consist of several subvalues separated by commas.**
- **Subparameters must be enclosed in parentheses or apostrophes.**
- **Special characters must be enclosed in apostrophes.**

The **PARM** parameter.

**Are we on track?**

**Which EXEC statement keyword parameter passes up to 100 characters of data to a program?**

- A. COND**
- B. PERFORM**
- C. PARM**
- D. ACCT**

## The PARM parameter.

### Coding the PARM parameter – Example 1.

```
//JOB1 JOB 766,SMITH  
//RUN#2 EXEC PGM=REPORT,  
// PARM=MONTHLY
```

**PARM  
Parameter**

PARM=MONTHLY



This EXEC statement passes one value (MONTHLY) as input to a program named REPORT.

## The PARM parameter.

### Coding the PARM parameter – Example 2.

```
//JOB1 JOB 766,SMITH  
//RUN#2 EXEC PGM=REPORT,  
// PARM='10-31-98'
```

**PARM  
Parameter**

PARM='10-31-98'



This EXEC statement passes the date (10-31-98) as input to the program called REPORT. The subparameter is enclosed in apostrophes because special characters are used.

## The PARM parameter.

### Coding the PARM parameter – Example 3.

```
//JOB1 JOB 766,SMITH  
//RUN#2 EXEC PGM=REPORT,  
// PARM=(MONTHLY,'10-31-98')
```

**PARM  
Parameter**

```
PARM=(MONTHLY,  
'10-31-98')
```



In this example, the EXEC statement passes both the type of report (MONTHLY) and the date (10-31-98) as subparameters of the PARM parameter. The two subparameters are enclosed in parentheses.

The PARM parameter.

Are we on track?

Complete the EXEC statement using the PARM parameter to pass on both the type of report (MONTHLY) and the date ('10-31-98').

```
//JOB1    JOB    776,SMITH  
//RUN#3  EXEC  PGM=REPORT,_____
```

The **PARM** parameter.

## The Loader program.

Occasionally, you might use the **LOADER** program when testing programs. The **LOADER** creates a program module in storage and passes control to it.

If you are coding the **PARM** parameter when using the **LOADER**, use the special syntax as shown here:

```
//GO EXEC PGM=LOADER,  
//      PARM=(loader parameters/user program parameters)
```

**What are the programs to which you can give the PARM parameter to?**

You can give **PARM** values to two programs:

- **The Loader.**
- **The module created by the Loader.**



The PARM parameter.

**Are we on track?**

**The \_\_\_\_\_ creates a program module in storage and passes control to it. The program then executes.**

The **COND** parameter.

**The COND parameter.**

**What does the COND parameter do?**

**To provide control over the whole job, you can code the condition (COND) parameter on the JOB statement.**

**You can also code it on the EXEC statement to control an individual step in the job.**

**The syntax for the COND parameter is:**

**COND=(code,operator)**

**When you use the COND parameter on an EXEC statement, the parameter specifies the conditions that allow the system to bypass a step by testing return codes from any or all previous steps.**

**If the result of any test is true, the system will bypass the step.**

The **COND** parameter.

## The **COND** subparameters.

As on the **JOB** statement, the code subparameter indicates a return code, and the operator subparameter indicates the type of test used for the comparison.

**COND=(code,operator)**

or

**COND=(code,operator,stepname)**

or

**COND=(code,operator,stepname.procstepname)**

**What happens if you specify only the code and operator subparameters?**

**If you specify only the code and operator subparameters, the test runs for all previous return codes in the job.**

**The COND parameter.**

## **The COND subparameters.**

**The stepname.procstepname subparameter (instead of the stepname subparameter) compares a code value against the return code from a specific previous procedure job step.**

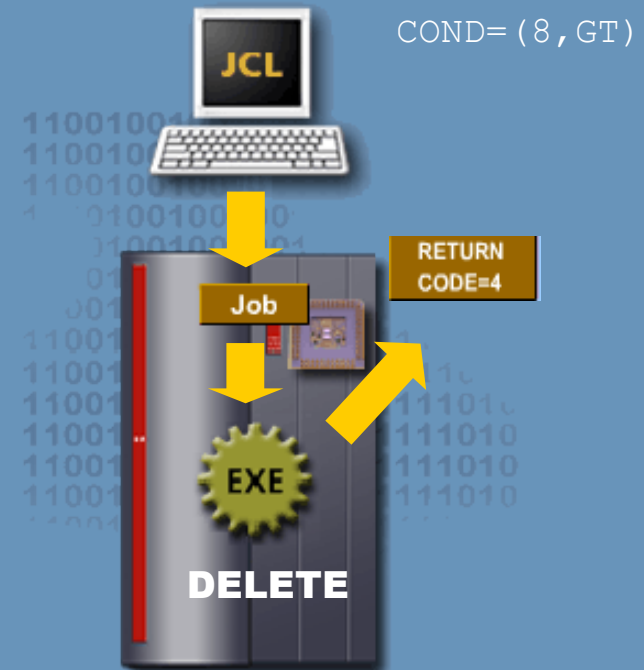
**The actual return code value, which may range from 0 to 4095, is compared with the return code specified in the COND parameter.**

**You can code up to eight comparisons. If any comparison is true, the system bypasses the step.**

## The COND parameter.

### COND parameter – an example.

```
//JOB1      JOB      778, SMITH
//EXAMPLE2 EXEC      PGM=DELETE
//EXAMPLE3 EXEC      PGM=UPDATE,
//          COND= (8, GT)
//DD1      DD      DSN=INPUT
```



In the example shown here, the COND parameter reads as follows:

**"If 8 is greater than the return code, do not execute the EXAMPLE3 step."**

The job executes the EXAMPLE3 step only if the RC is 8 or greater. If the RC is 0 through 7, the EXAMPLE3 step will be bypassed.

The **COND** parameter.

**Are we on track?**

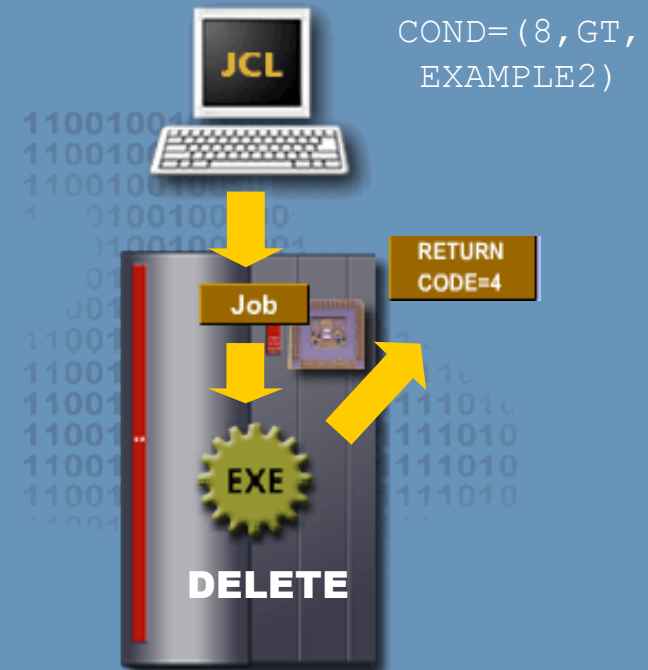
**Code a COND parameter that skips this step if any previous step return code is less than or equal to 8.**

```
//STEP2 EXEC PGM=UPDATE,
```

## The COND parameter.

### COND parameter – an example.

```
//JOB1      JOB      778, SMITH
//EXAMPLE2  EXEC     PGM=DELETE
//EXAMPLE3  EXEC     PGM=UPDATE,
//          COND= (8, GT, EXAMPLE2)
//DD1      DD      DSN=INPUT
```



The COND parameter in the step EXAMPLE3, includes a stepname subparameter. This causes the COND statement to read as follows:

**"If 8 is greater than the return code from step EXAMPLE2, do not execute the EXAMPLE3 step."**

**The COND parameter.**

## **The COND subparameters – EVEN & ONLY.**

**In addition to code, operator, stepname, and procedure stepname, the EVEN and ONLY subparameters may also be coded on the COND parameter.**

**These subparameters do not apply to condition codes returned by a program after normal termination. They relate to abnormal termination of a prior step. Abnormal termination occurs when unexpected conditions arise during execution of a step.**

**Without the use of EVEN or ONLY, a job bypasses all remaining steps following an abnormal program termination.**

**The EVEN subparameter allows the current step to execute even if any previous step terminates abnormally.**

**Conversely, the ONLY subparameter allows the current step to execute only if any previous step terminates abnormally.**



## The COND parameter.

### EVEN subparameter – an example.

```
//JOB1      JOB      778, SMITH
//EXAMPLE1 EXEC      PGM=STEP1
//EXAMPLE2 EXEC      PGM=STEP2
//EXAMPLE3 EXEC      PGM=STEP3
//EXAMPLE4 EXEC      PGM=STEP4,
//          COND=EVEN
//DD1      DD      DSN=INPUT
```



If you code COND=EVEN on an EXEC statement as shown here, the program STEP4 always executes, even if a previous step (e.g. program STEP3) in the job terminates abnormally.

## The COND parameter.

### ONLY subparameter – an example.

```
//JOB1      JOB      778, SMITH
//EXAMPLE1 EXEC      PGM=STEP1
//EXAMPLE2 EXEC      PGM=STEP2
//EXAMPLE3 EXEC      PGM=STEP3
//EXAMPLE4 EXEC      PGM=STEP4,
//          COND=ONLY
//DD1      DD      DSN=INPUT
```



If you code COND=ONLY on an EXEC statement as shown here, the program STEP4 will execute only if a previous step in the job terminates abnormally.


## The COND parameter.

### Using the EVEN & ONLY subparameters.

The EVEN and ONLY subparameters cannot appear on the same step. They are mutually exclusive.

However, EVEN or ONLY can be coded in place of one of the eight RC test allowed for each step. The order in which tests are coded does not matter.

For example, the two EXEC statements shown here mean the same thing.



```
//ST4 EXEC PGM=PROG7,  
//      COND=( (10,EQ,STEP5) ,EVEN)  
  
//ST4 EXEC PGM=PROG7,  
//      COND=(EVEN, (10,EQ,STEP5) )
```

The **COND** parameter.

**Are we on track?**

**Which one of the following statements would you code to run the **FIXIT** program only in case of an abnormal termination in the previous step?**

- A. //STEP EXEC PGM=FIXIT**
- B. //STEP3 EXEC PGM=FIXIT,COND=EVEN**
- C. //STEP3 EXEC PGM=FIXIT,COND=ONLY**

**The system library.**

## **Program libraries.**

**In a job, each job step begins with an EXEC statement that identifies a program name. In order to run the program in the EXEC statement, the system searches for it in program libraries.**

**It will search one or more system program libraries automatically or you can direct the system to search for the program in a private program library.**

## The system library.

### Steps involved in invoking a program.

**When a job step requests a program, the system searches for the program in a system library named SYS1.LINKLIB or in a list of libraries that the installation has defined in a PARMLIB member called LINKLIST.**

**LINKLIST is installation dependent and can include a number of data sets with loadable programs searched by default.**

**Once the system finds a program in the SYS1.LINKLIB or any library in LINKLIST, it invokes the program.**

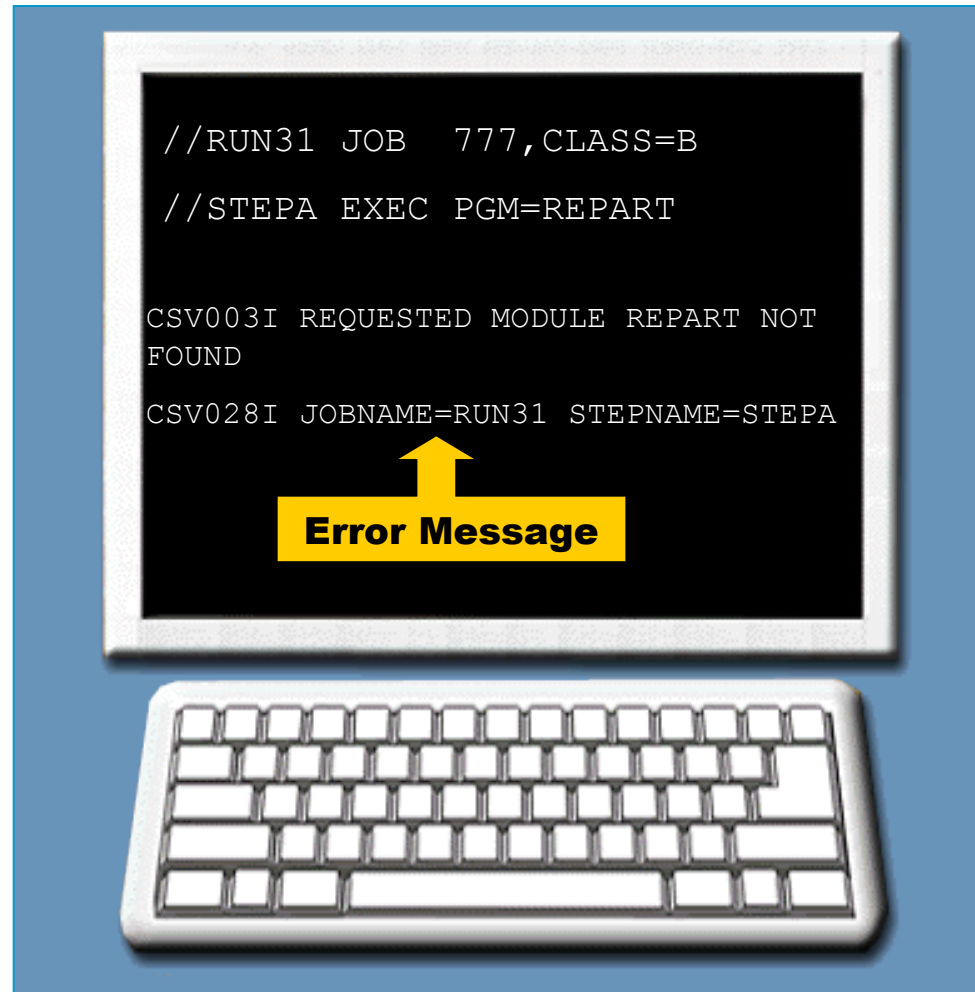
**If the system cannot find the program, it will generate an abnormal termination (or abend) when you try to run the job.**

## The system library.

# Program libraries – an example.

In the example, the programmer wanted to execute a program called REPORT and coded the JCL as shown.

The operating system searches SYS1.LINKLIB or LINKLIST for REPORT. The error messages shown occur because REPORT was misspelled as REPART in the JCL.



## The system library.

# Private program libraries.

You can use private libraries to store programs.

If you want to invoke a program called MYPROG which is stored in a private library, you must tell the operating system the name of the private library by coding a special DD statement named JOBLIB before the first EXEC statement in the job.

The JOBLIB DD statement causes the system to search a private library before searching **SYS1.LINKLIB.**





## The system library.

### Are we on track?

**From the code determine which library the system would search first for UTILITY.**

```
//MYJOB1 JOB 514,SMITH  
//JOB LIB DD DSN=USER1  
//STEP1 EXEC PGM=UTILITY
```



- A. SMITHLIB private library.**
- B. SYS1.LINKLIB or LINKLIST.**
- C. USER1 private library.**


## The STEPLIB DD statement.

# The STEPLIB DD statement.

If most of the programs for a job reside in SYS1.LINKLIB or LINKLIST and only a few are in private libraries, it makes more sense to direct the system to search a private library on a step-by-step basis.

This saves processing time by eliminating unnecessary searching.

To search a private library directly you use a special DD statement called STEPLIB DD statement as shown.



```
//STEP1 EXEC PGM=PROGA  
//STEPLIB DD DSN=MYLIB,DISP=SHR
```



## The STEPLIB DD statement.

# Comparison between JOBLIB and STEPLIB DD statements.

Just like a JOBLIB DD statement, the STEPLIB DD statement searches a private library for a specified program.

But, the STEPLIB DD statement is in effect only for the duration of the step it follows.

```
//JOB1      JOB  777, SMITH
//STEP1     EXEC PGM=PROGA
//STEP2     EXEC PGM=MYPROG
//STEPLIB  DD   DSN=LIBRARY,
//          DISP=SHR
//STEP3     EXEC PGM=PROGB
```

## The STEPLIB DD statement.

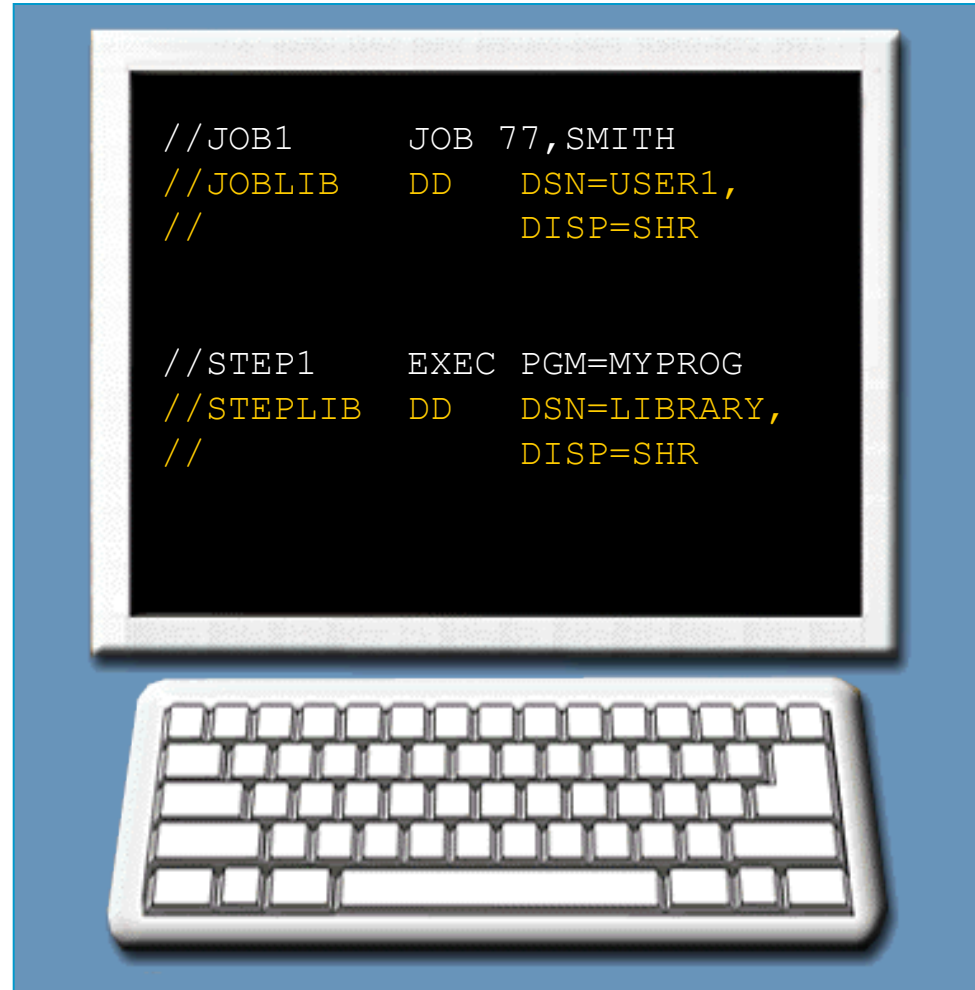
# Comparison between JOBLIB and STEPLIB DD statements.

## What if a STEPLIB and JOBLIB both appear in a job?

In this case the STEPLIB overrides the JOBLIB. The system ignores JOBLIB and does not search it in the step.

If the system does not find the program in library specified by the STEPLIB, it searches the system libraries (SYS1.LINKLIB and LINKLIST) next.

If it does not find the program there, the job step abends.



## Coding EXEC statements.

### Unit summary.

Now that you have completed this unit, you should be able to:

- **Code an EXEC statement to specify a program to be executed.**
- **Correct coding errors in an EXEC statement.**
- **Identify which JCL statement has caused a "PROGRAM NOT FOUND" error message.**
- **Identify the system library from which programs are retrieved at execution time.**
- **Identify the DD statement names used to specify a private library from which programs are retrieved at execution time.**
- **Select the place in the job stream where STEPLIB and JOBLIB DD statements should be located.**
- **Code a JOBLIB DD statement.**

# **JCL**

## **Chapter a3 Coding EXEC statements**

## **Job Control Language**

**Chapter a1. Introduction to JCL**

**Chapter a2. Coding JOB statements**

**Chapter a3. Coding EXEC statements**

**Chapter a4. Coding DD statements**

**Chapter a5. Analyzing job output**

**Chapter a6. Conditional processing**

## **Job Control Language**

**Chapter b1. Using special DD statements**

**Chapter b2. Introducing procedures**

**Chapter b3. Modifying EXEC parameters**

**Chapter b4. Modifying DD parameters**

**Chapter b5. Determining the effective JCL**

**Chapter b6. Symbolic parameters**



## Job Control Language

**Chapter c1. Nested procedures**

**Chapter c2. Cataloging procedures**

**Chapter c3. Using utility programs**

**Chapter c4. Sample utility application**

# Chapter a3

## Coding EXEC statements

Each job step begins with an EXEC statement. This statement identifies the name of the program (or procedure) that has to be executed for a particular job. You have to code an EXEC statement for each program that you have to execute in a job.

This unit will explain to you how to be able to code a simple EXEC statement and identify the reasons for basic JCL errors.

### Coding EXEC statements

#### Course objectives.

##### Be able to:

- Code an EXEC statement to specify a program to be executed.
- Correct coding errors in an EXEC statement.
- Identify which JCL statement has caused a "PROGRAM NOT FOUND" error message.
- Identify the system library from which programs are retrieved at execution time.
- Identify the DD statement names used to specify a private library from which programs are retrieved at execution time.
- Select the place in the job stream where STEPLIB and JOBLIB DD statements should be located.
- Code a JOBLIB DD statement.

6

© 2013 Pearson Education, Inc. All rights reserved. This content is not to be distributed to their respective companies.

### The EXEC statement

#### The EXEC statement.

To execute each program in a job, you need to code one EXEC statement. Each job step begins with an EXEC statement that identifies a program name.

The EXEC statement is used to invoke the program that you want to execute as part of a job. In addition to this, you can also use the EXEC statement to invoke a cataloged procedure.

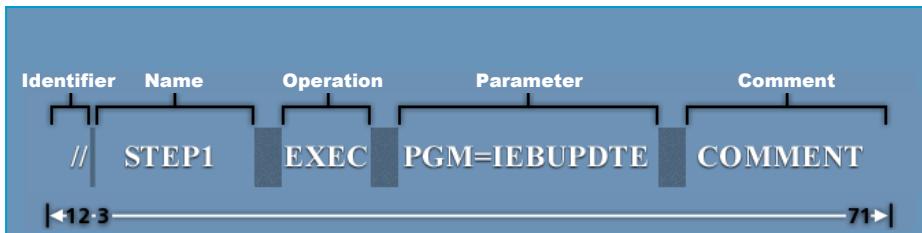
#### What is a cataloged procedure?

A cataloged procedure refers to a set of JCL statements that are stored in a library and retrieved by its name.

A procedure may contain one or more EXEC statements.

## The EXEC statement

### The EXEC statement.



Like the JOB statement, the EXEC statement too has five fields. The EXEC statement format includes the following:

- Identifier field (//): It occupies position 1 and 2.
- Name field: It names the step starting in position 3.
- EXEC operator field: It states the JCL statement type.
- Parameter field: It is used to state the parameters used on an EXEC statement.
- Comment field: This field is optional.

8

Copyright © 2006 CA. All trademarks, trade names, services marks and logos referenced herein belong to their respective companies.

Operation field (EXEC) can start at position 12.  
Parameter field (PGM=) can start at position 18.  
Comment field starts after at least one space.

### The EXEC statement

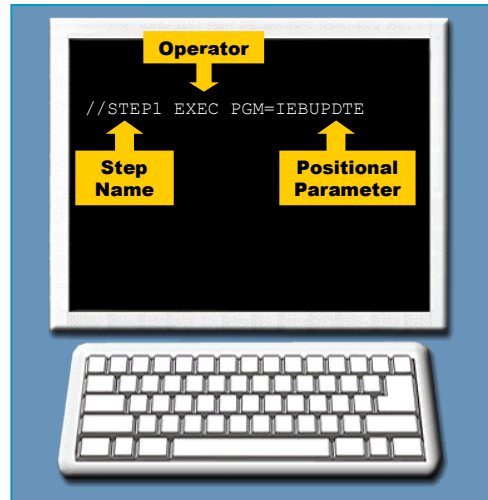
#### The EXEC statement.

Shown here is an example where the step name is STEP1. The operator is EXEC and the positional parameter is defined by PGM=IEBUPDTE.

#### What is IEBUPDTE?

IEBUPDTE is a system utility program that the system invokes during the execution of STEP1.

The step name STEP1 identifies the EXEC statement so that the subsequent JCL statements can refer to it.



I can recommend: code a step name on all your EXEC statements even if there might not be any references to those statements.

IEBUPDTE – system utility to create or modify sequential or partitioned data sets. However, the program can be used only with data sets containing fixed-length records of no more than 80 bytes. It is used primarily for updating procedure, source, and macro libraries, such as those containing JCL.

## The EXEC statement

### The step name.

Following are the coding rules for the step name:

- The step name must begin in position 3.
- The step name must be 1 to 8 characters in length.
- The first character in the step name should be either alphabetic or a national symbol. It cannot be a number.
- Rest of the characters in the step name can either be alphanumeric or they can be national symbols.
- Special characters and spaces cannot be used in a step name.

10

Copyright © 2006 CA. All trademarks, trade names, services marks and logos referenced herein belong to their respective companies.

#### Valid Step Names

```
//STEP1 EXEC  
//EXAMPLE4 EXEC  
//RUN#2 EXEC
```

#### Invalid Step Names

```
//STEP1+ EXEC (Includes a special character)  
//EXAMPLE14 EXEC (More than eight characters)  
// RUN#2 EXEC (Does not begin in position 3)
```

## The EXEC statement

### The step name.

Shown here are two step names  
STEP#FOUR and LA\$JOE.

### Are both the names valid step names?

The step name LA\$JOE is acceptable because it fits all the requirements defined in the rules for coding a step name (the \$ is one of the national symbols).

But, STEP#FOUR is not a valid step name because it contains more than eight characters.





**The EXEC statement.**

**Are we on track?**

**Which of the following step names are valid?**

- A. \$STEP#5**
- B. RUN TWO**
- C. \*STEP4**
- D. EXAMPLE#12**

12

Copyright © 2006 CA. All trademarks, trade names, services marks and logos referenced herein belong to their respective companies.

The correct answer is A.

B. is incorrect, as it contains a blank character.

C. is incorrect, as it starts with asterisk, a special character.

D. is incorrect, as it is longer than 8 characters.

**The EXEC statement.**

## **Glossary.**

**Cataloged procedure**

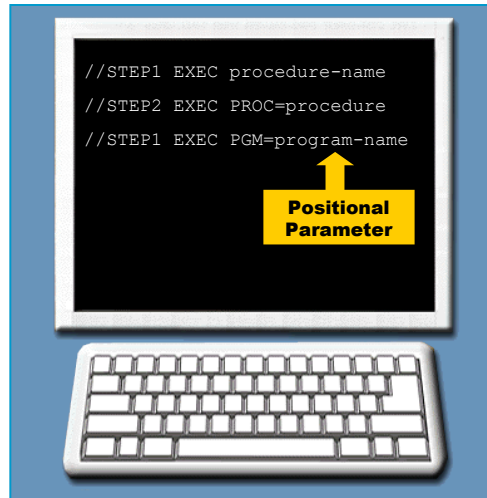
**Prepared sets of JCL statements cataloged in a procedure library.**

### The EXEC parameter field.

## The positional parameter.

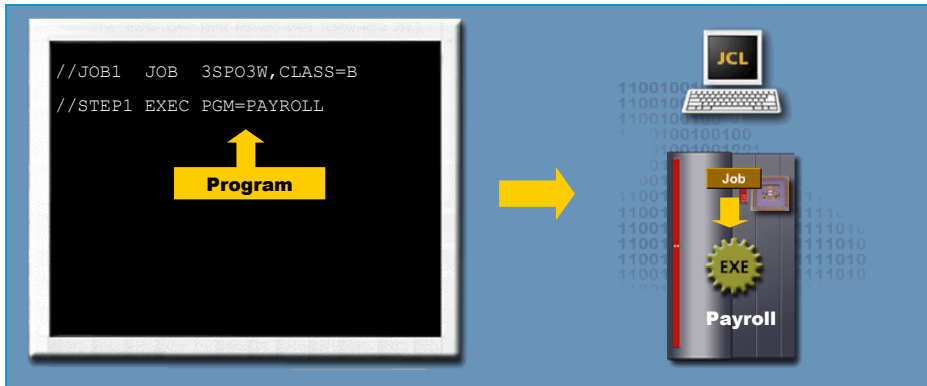
The parameter field follows the EXEC operator and may contain multiple parameters. The first parameter in an EXEC statement is a positional parameter that designates the program or procedure the system executes during the job step.

This positional parameter is often coded like a keyword parameter using either PGM= or PROC=.



The EXEC parameter field.

The PGM= positional parameter.



What does PGM= designate?

PGM= designates a program the system executes during the job step. Shown here is the syntax of PGM as a positional parameter in the EXEC statement.

15

Copyright © 2006 CA. All trademarks, trade names, services marks and logos referenced herein belong to their respective companies.

For example, to call a program named PAYROLL from step called STEP1, you would code the EXEC statement as shown here.

The EXEC parameter field.

**Are we on track?**

**Code the EXEC statement using the proper operator where the step name is STEP1 and the name of program is PAYROLL?**

**//JOB1 JOB 255,SMITH**

16

Copyright © 2006 CA. All trademarks, trade names, services marks and logos referenced herein belong to their respective companies.

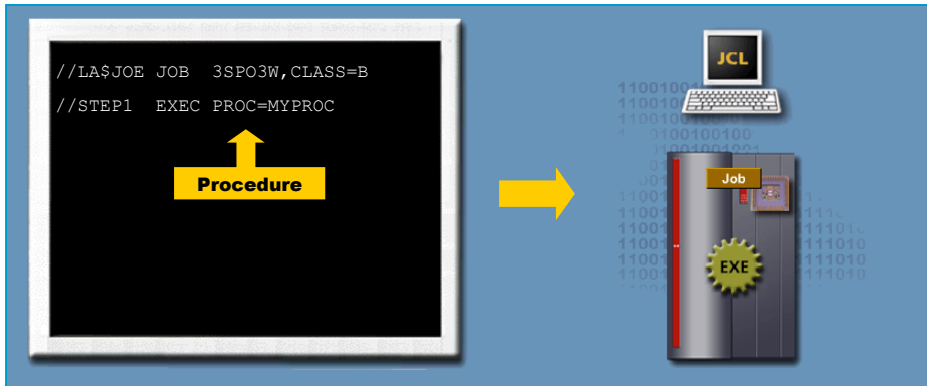
The correct answer is:

**//STEP1 EXEC**

**PGM=PAYROLL**

The EXEC parameter field.

The PROC= positional parameter.



**What does PROC= designate?**

PROC= designates a procedure the system executes. The syntax of PROC as a positional parameter in an EXEC statement is shown here.

17

Copyright © 2006 CA. All trademarks, trade names, services marks and logos referenced herein belong to their respective companies.

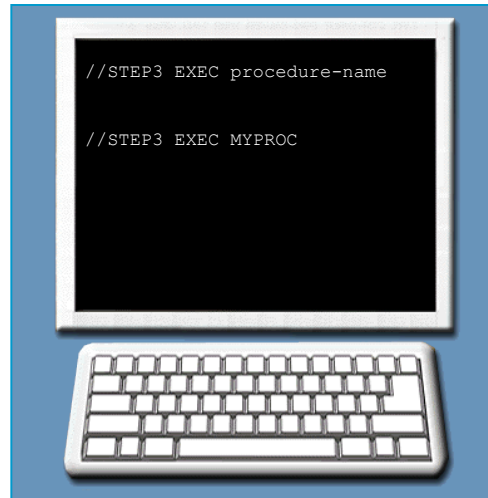
For example, to call a procedure named MYPROC from a step called STEP1, code the EXEC statement as shown here.

### The EXEC parameter field.

## Omitting the PROC= keyword.

If you omit the PGM= or PROC= keyword, the operating system automatically looks for a procedure by the specified name. Shown here is an example of the syntax when omitting PROC= keyword in an EXEC statement is shown.

For example, to call a procedure named MYPROC from a step named STEP3, you will need to code the EXEC statement as shown on the right.



The EXEC parameter field.

**Are we on track?**

**If you leave out the PGM= or PROC= keyword when coding an EXEC statement, what does the operating system do automatically?**

- A. Searches for a procedure with the specified name.**
- B. Searches for a program with the specified name.**
- C. Searches all programs and procedures with the specified name**
- D. Displays an error and cause an abnormal termination.**

The correct answer is A.

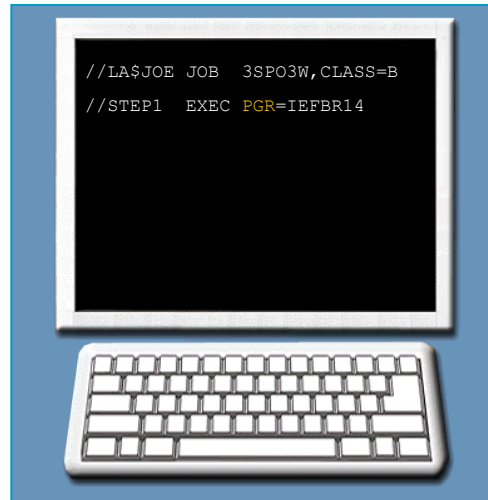


### The EXEC parameter field.

## Positional parameter coding errors.

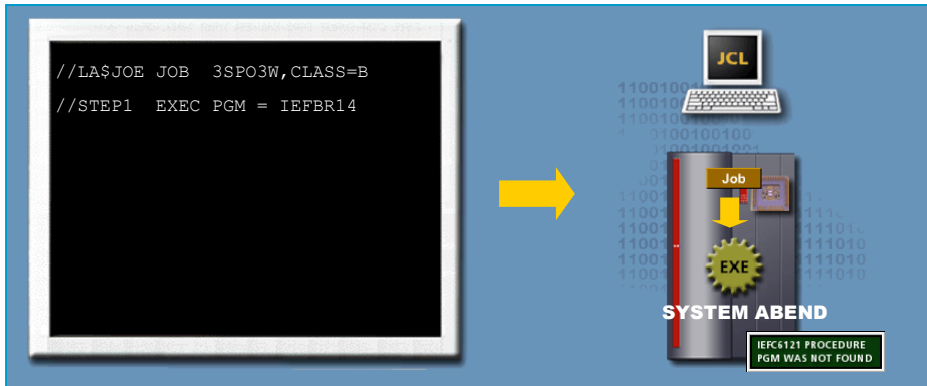
JCL errors occur if you, as a JCL programmer fail to follow any of the coding rules regarding the PGM and PROC positional parameters.

For example, the misspelling of PGM (as PGR) in the EXEC statement shown here returns a JCL error.



## The EXEC parameter field.

### Positional parameter coding errors.



```
//LA$JOE JOB 3SPO3W,CLASS=B
//STEP1 EXEC PGM = IEFBR14
```

**SYSTEM ABEND**  
IEFCS121 PROCEDURE  
PGM WAS NOT FOUND

In the example shown here, the equal sign in the parameter field is preceded and followed by a space. The system interprets "PGM" as a procedure name and not as a keyword for the positional parameter of the EXEC statement.

21

Copyright © 2006 CA. All trademarks, trade names, services marks and logos referenced herein belong to their respective companies.

When this program executes, an error "NOT FOUND" may appear. You can fix this error by eliminating the spaces before and after the equal sign.

**Additional EXEC parameters.**

### **Procedures.**

**In addition to programs, you can use the EXEC statement to invoke a cataloged procedure, which is a set of JCL statements that you place in a library and retrieve by its name.**

**A procedure can contain one or more EXEC statements, with associated DD statements.**

You have already learnt that in a job, you should code one EXEC statement for each program that you want to execute. Each job step begins with an EXEC statement that identifies a program name – or procedure name.

**Additional EXEC parameters.**

**Are we on track?**

**A \_\_\_\_\_ is a set of associated EXEC and DD statements stored in a procedure library.**

23

Copyright © 2006 CA. All trademarks, trade names, services marks and logos referenced herein belong to their respective companies.

The correct answer is procedure.

### Additional EXEC parameters.

#### Keyword parameters.

You may code keyword parameters on the EXEC statement in any order, following the program or procedure name.

You can use any of the keyword parameters shown here.

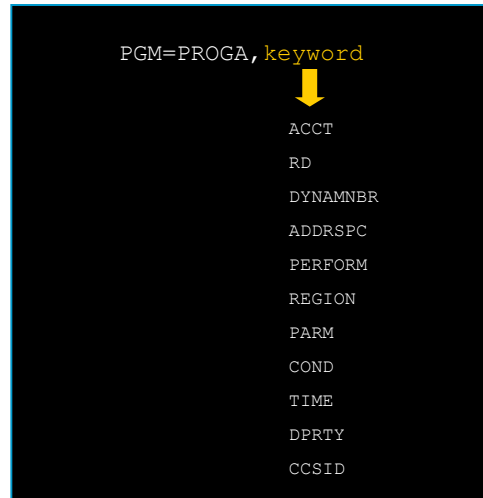
If you code one of these keyword parameters on the EXEC statement, the keyword parameter value will apply only to that step.

The two keyword parameters used most frequently with the EXEC statement are:

- The PARM parameter.
- The COND parameter.

24

Copyright © 2006 CA. All trademarks, trade names, services marks and logos referenced herein belong to their respective companies.



You can code certain keyword parameters to code a JOB statement in order to accomplish a task for the entire job. Some parameters are the same in JOB and EXEC statements. In that case they are valid for the entire job, or only for the step.

See the „z/OS MVS JCL Reference“ manual, Overrides section in each parameter description.

**The PARM parameter.**

**The PARM parameter.**

**What does the PARM parameter do?**

**The PARM parameter passes information to the executing program. Some programs accept information from the PARM parameter about how many times to execute.**

**For example, a program may need to know whether a report cycle is "annual" or "monthly".**

**The records the program uses vary depending on which value is passed to it.**

**Similarly, the PARM parameter can supply a password to the program that is required before the program executes.**

**The syntax for the PARM parameter is:**

**PARM=(SUBPARAMETER,SUBPARAMETER)**

25

Copyright © 2006 CA. All trademarks, trade names, services marks and logos referenced herein belong to their respective companies.

Use the PARM parameter to pass variable information to the processing program executed by this job step. To use the information, the processing program must contain instructions to retrieve the information.

**The PARM parameter.**

**Rules for coding the PARM parameter.**

The general syntax and rules for coding the PARM parameter are:

- **The PARM parameter can include up to 100 characters.**
- **The PARM parameter can consist of several subvalues separated by commas.**
- **Subparameters must be enclosed in parentheses or apostrophes.**
- **Special characters must be enclosed in apostrophes.**

PARM=onehundredcharacters

PARM=(subvalue1,subvalue2)

PARM=('subparameter1',subparameter2')

PARM='#'

The PARM parameter.

Are we on track?

Which EXEC statement keyword parameter passes up to 100 characters of data to a program?

- A. COND
- B. PERFORM
- C. PARM
- D. ACCT

27

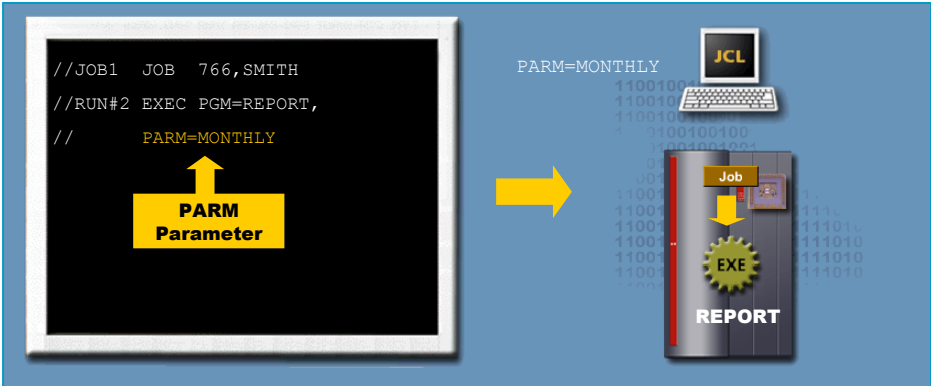
Copyright © 2006 CA. All trademarks, trade names, services marks and logos referenced herein belong to their respective companies.

The correct answer is C.



The PARM parameter.

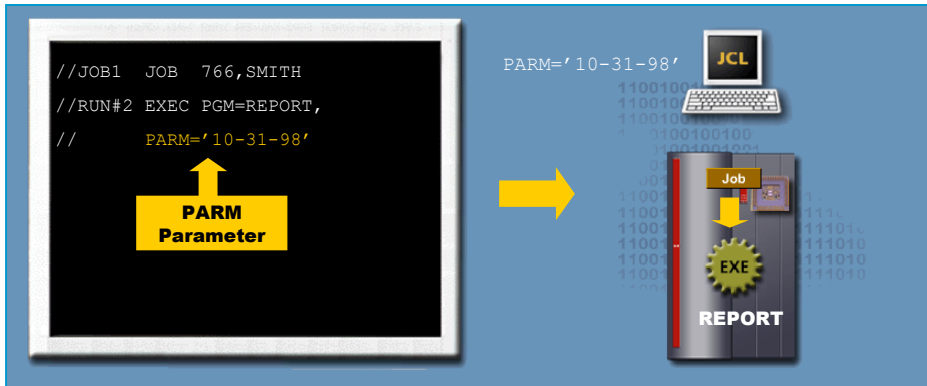
Coding the PARM parameter – Example 1.



This EXEC statement passes one value (MONTHLY) as input to a program named REPORT.

## The PARM parameter.

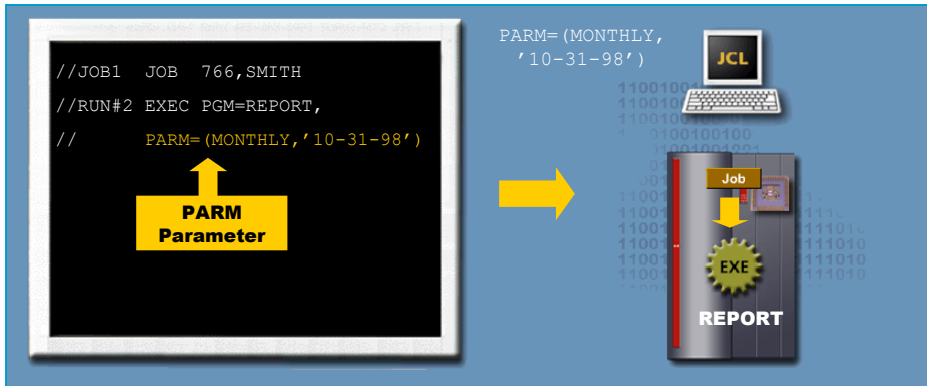
### Coding the PARM parameter – Example 2.



This EXEC statement passes the date (10-31-98) as input to the program called REPORT. The subparameter is enclosed in apostrophes because special characters are used.

**The PARM parameter.**

**Coding the PARM parameter – Example 3.**



In this example, the EXEC statement passes both the type of report (MONTHLY) and the date (10-31-98) as subparameters of the PARM parameter. The two subparameters are enclosed in parentheses.

The PARM parameter.

**Are we on track?**

**Complete the EXEC statement using the PARM parameter to pass on both the type of report (MONTHLY) and the date ('10-31-98').**

```
//JOB1 JOB 776,SMITH  
//RUN#3 EXEC PGM=REPORT,_____
```

31

Copyright © 2006 CA. All trademarks, trade names, services marks and logos referenced herein belong to their respective companies.

The correct answer is // PARM=(MONTHLY,'10-31-98')

The PARM parameter.

## The Loader program.

Occasionally, you might use the LOADER program when testing programs. The LOADER creates a program module in storage and passes control to it.

If you are coding the PARM parameter when using the LOADER, use the special syntax as shown here:

```
//GO EXEC PGM=LOADER,  
//      PARM=(loader parameters/user program parameters)
```

**What are the programs to which you can give the PARM parameter to?**

You can give PARM values to two programs:

- The Loader.
- The module created by the Loader.

32

Copyright © 2006 CA. All trademarks, trade names, services marks and logos referenced herein belong to their respective companies.

The slash (/) within the PARM parameter separates the Loader parameters from those given to the user program.

The PARM parameter.

**Are we on track?**

**The \_\_\_\_\_ creates a program module in storage and passes control to it. The program then executes.**

33

Copyright © 2006 CA. All trademarks, trade names, services marks and logos referenced herein belong to their respective companies.

The correct answer is Loader.

**The COND parameter.**

**The COND parameter.**

**What does the COND parameter do?**

**To provide control over the whole job, you can code the condition (COND) parameter on the JOB statement.**

**You can also code it on the EXEC statement to control an individual step in the job.**

**The syntax for the COND parameter is:**

**COND=(code,operator)**

**When you use the COND parameter on an EXEC statement, the parameter specifies the conditions that allow the system to bypass a step by testing return codes from any or all previous steps.**

**If the result of any test is true, the system will bypass the step.**

34

Copyright © 2006 CA. All trademarks, trade names, services marks and logos referenced herein belong to their respective companies.

Use the COND parameter to test return codes from previous job steps and determine whether to bypass this job step. You can specify one or more tests on the COND parameter, and you can test return codes from particular job steps or from every job step that has completed processing.

If any of the test conditions are satisfied, the system evaluates the COND parameter as true and bypasses the job step. If none of the test conditions specified on the COND parameter are satisfied, the system evaluates the COND parameter as false and executes the job step.

**The COND parameter.**

### **The COND subparameters.**

**As on the JOB statement, the code subparameter indicates a return code, and the operator subparameter indicates the type of test used for the comparison.**

```
COND=(code,operator)
or
COND=(code,operator,stepname)
or
COND=(code,operator,stepname.procstepname)
```

### **What happens if you specify only the code and operator subparameters?**

**If you specify only the code and operator subparameters, the test runs for all previous return codes in the job.**

35

Copyright © 2006 CA. All trademarks, trade names, services marks and logos referenced herein belong to their respective companies.

## **stepname**

Identifies the EXEC statement of a previous job step that issues the return code to be used in the test. If you omit stepname, the code you specify is compared to the return codes from all previous steps. If the return code issued by any of those previous steps causes the test condition to be satisfied, the system evaluates the COND parameter as true and bypasses the job step.

## **stepname.procstepname**

Identifies a step in a cataloged or in-stream procedure called by an earlier job step. Stepname identifies the EXEC statement of the calling job step; procstepname identifies the EXEC statement of the procedure step that issues the return code to be used in the test.



**The COND parameter.**

### **The COND subparameters.**

**The stepname.procstepname subparameter (instead of the stepname subparameter) compares a code value against the return code from a specific previous procedure job step.**

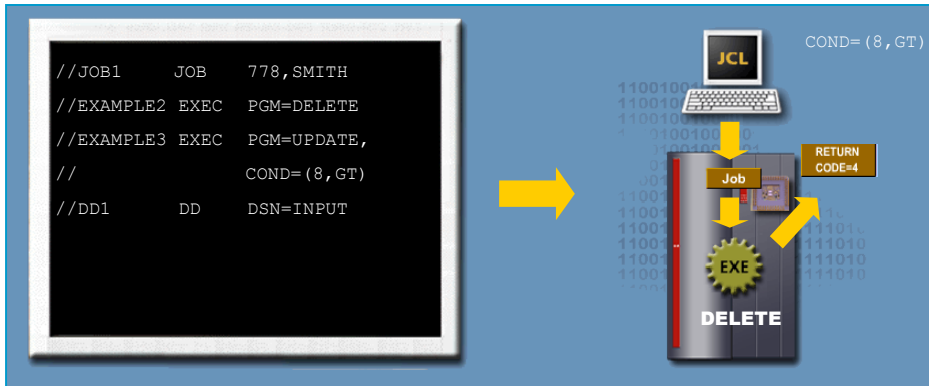
**The actual return code value, which may range from 0 to 4095, is compared with the return code specified in the COND parameter.**

**You can code up to eight comparisons. If any comparison is true, the system bypasses the step.**

The stepname subparameter, if coded, results in comparing the code value against the return code from a previous job step with the specified stepname.

**The COND parameter.**

**COND parameter – an example.**



In the example shown here, the COND parameter reads as follows:

**"If 8 is greater than the return code, do not execute the EXAMPLE3 step."**

The job executes the EXAMPLE3 step only if the RC is 8 or greater. If the RC is 0 through 7, the EXAMPLE3 step will be bypassed.

It means:

- RC is 0-7: consequently 8 is GT RC consequently bypass the EXAMPLE3 step.
- RC is 8: consequently 8 is EQ RC consequently execute the EXAMPLE3 step.
- RC is 9: consequently 8 is LT RC consequently execute the EXAMPLE3 step.

Try MCOE.EDU.JCL.JCL(CONDSAMP).

This example is a little unusual. Mostly step is executed if RC is less than...

The COND parameter.

Are we on track?

Code a COND parameter that skips this step if any previous step return code is less than or equal to 8.

```
//STEP2 EXEC PGM=UPDATE,
```

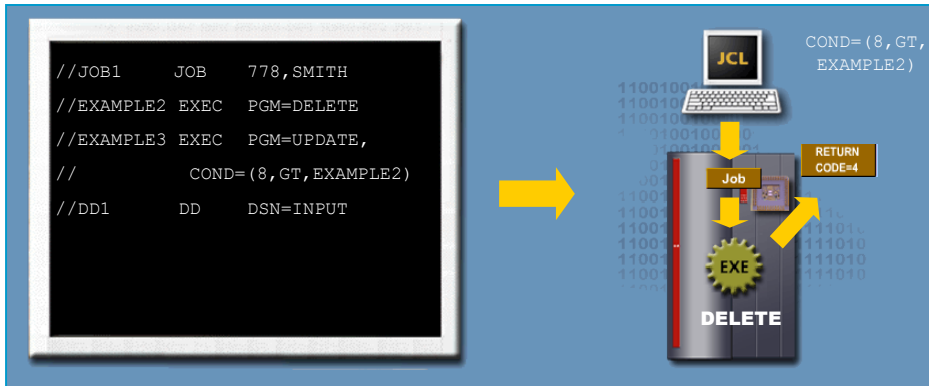
38

Copyright © 2006 CA. All trademarks, trade names, services marks and logos referenced herein belong to their respective companies.

The correct answer is COND=(9,GT) or COND=(8,GE).  
Look to the example on the previous slide.

## The COND parameter.

### COND parameter – an example.



The COND parameter in the step EXAMPLE3, includes a stepname subparameter. This causes the COND statement to read as follows:

**"If 8 is greater than the return code from step EXAMPLE2, do not execute the EXAMPLE3 step."**

Copyright © 2009 CA. All trademarks, trade names, service marks and logos referenced herein belong to their respective companies.

It means:

RC from EXAMPLE2 is 0-7: then 8 is GT RC consequently bypass the EXAMPLE3 step.

RC from EXAMPLE2 is 8: then 8 is EQ RC consequently execute the EXAMPLE3 step.

RC from EXAMPLE2 is 9: then 8 is LT RC consequently execute the EXAMPLE3 step.

Try MCOE.EDU.JCL.JCL(CONDSAMP).

The COND parameter.

### The COND subparameters – EVEN & ONLY.

In addition to code, operator, stepname, and procedure stepname, the EVEN and ONLY subparameters may also be coded on the COND parameter.

These subparameters do not apply to condition codes returned by a program after normal termination. They relate to abnormal termination of a prior step. Abnormal termination occurs when unexpected conditions arise during execution of a step.

Without the use of EVEN or ONLY, a job bypasses all remaining steps following an abnormal program termination.

The EVEN subparameter allows the current step to execute even if any previous step terminates abnormally.

Conversely, the ONLY subparameter allows the current step to execute only if any previous step terminates abnormally.

40

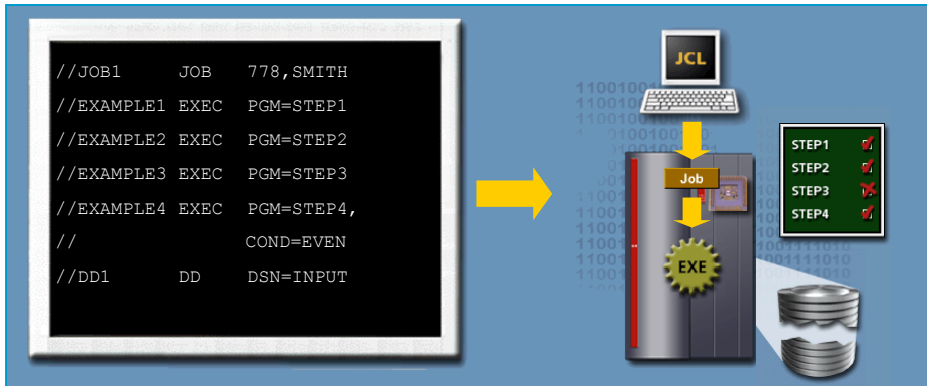
Copyright © 2006 CA. All trademarks, trade names, services marks and logos referenced herein belong to their respective companies.

EVEN specifies that this job step is to be executed **even if** a preceding job step abnormally terminated.

ONLY specifies that this job step is to be executed **only if** a preceding job step abnormally terminated.

## The COND parameter.

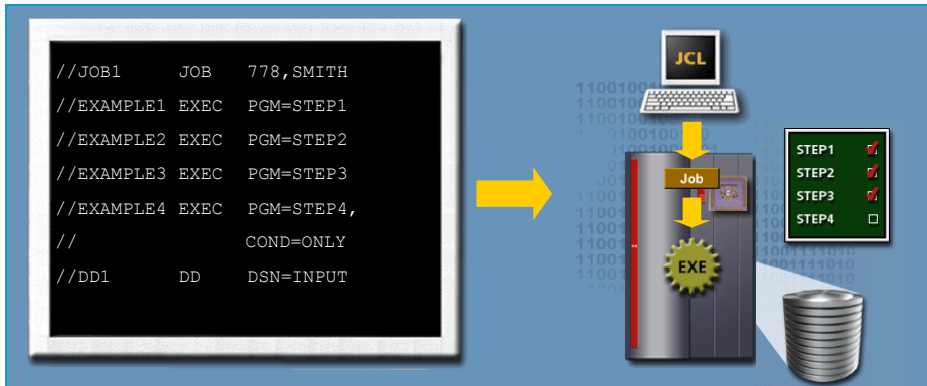
### EVEN subparameter – an example.



If you code COND=EVERN on an EXEC statement as shown here, the program STEP4 always executes, even if a previous step (e.g. program STEP3) in the job terminates abnormally.

**The COND parameter.**

**ONLY subparameter – an example.**



If you code COND=ONLY on an EXEC statement as shown here, the program STEP4 will execute only if a previous step in the job terminates abnormally.

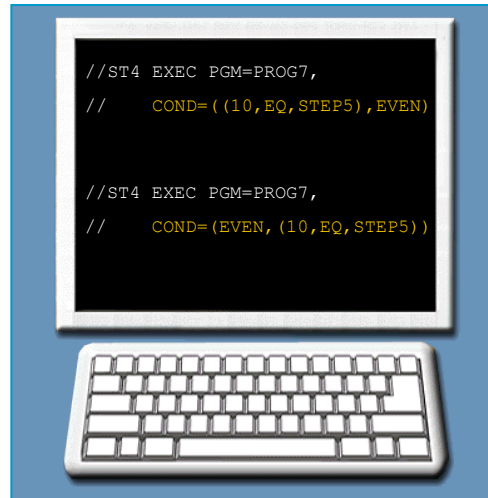
### The COND parameter.

#### Using the EVEN & ONLY subparameters.

The EVEN and ONLY subparameters cannot appear on the same step. They are mutually exclusive.

However, EVEN or ONLY can be coded in place of one of the eight RC test allowed for each step. The order in which tests are coded does not matter.

For example, the two EXEC statements shown here mean the same thing.



43

Copyright © 2006 CA. All trademarks, trade names, services marks and logos referenced herein belong to their respective companies.

If a job step terminates abnormally, the system bypasses all subsequent steps unless they have been coded with an EVEN or ONLY subparameters on the COND parameter.



The COND parameter.

Are we on track?

Which one of the following statements would you code to run the **FIXIT** program only in case of an abnormal termination in the previous step?

- A. //STEP EXEC PGM=FIXIT
- B. //STEP3 EXEC PGM=FIXIT,COND=EVEN
- C. //STEP3 EXEC PGM=FIXIT,COND=ONLY

The correct answer is C.

**The system library.**

## **Program libraries.**

**In a job, each job step begins with an EXEC statement that identifies a program name. In order to run the program in the EXEC statement, the system searches for it in program libraries.**

**It will search one or more system program libraries automatically or you can direct the system to search for the program in a private program library.**

**The system library.**

**Steps involved in invoking a program.**

**When a job step requests a program, the system searches for the program in a system library named SYS1.LINKLIB or in a list of libraries that the installation has defined in a PARMLIB member called LINKLIST.**

**LINKLIST is installation dependent and can include a number of data sets with loadable programs searched by default.**

**Once the system finds a program in the SYS1.LINKLIB or any library in LINKLIST, it invokes the program.**

**If the system cannot find the program, it will generate an abnormal termination (or abend) when you try to run the job.**

See SYS1.PARMLIB(LNKLSTM00) as an example of linklist member:

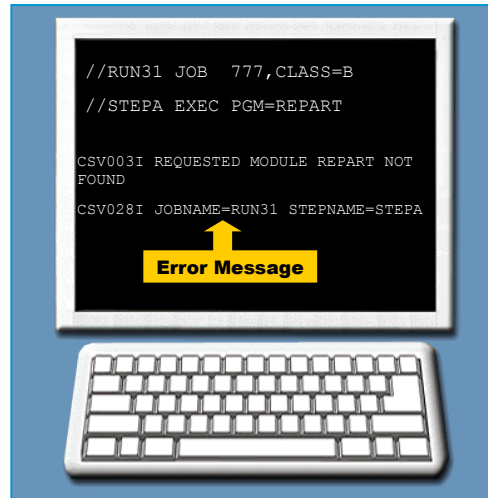
SYS2.XAD1.LINKLIB, STEPLIB	SYSTEMS PROGRAMMING MVS/XA
SYS1.V1R3M0.SHASLINK,	JES2
SYS1.V1R3M0.SHASMIG,	JES2
ISP.SISPLOAD,	ISPF
ISF.SISFLOAD,	SDSF
ISF.SISFLINK,	SDSF
CEE.SCEERUN,	LE/370 RUNTIME
IGY.V1R2M0.SIGYCOMP,	AD/CYCLE COBOL COMPILER
EDC.V1R2M0.SEDCDCMP,	AD/CYCLE C/370
SYS2.CMDLIB, COMMAND LIBRARY	SYSTEMS PROGRAMMING TSO
SYS1.CMDLIB,	IBM TSO COMMAND LIBRARY
SYS2.XDC32.LINKLIB,	XDC LINKLIB
SYS2.LINKLIB,	SYSTEMS PROGRAMMING LIBRARY
FATS.LINKLIB,	FATAR FOR RPLUS PEOPLE

### The system library.

## Program libraries – an example.

In the example, the programmer wanted to execute a program called REPORT and coded the JCL as shown.

The operating system searches SYS1.LINKLIB or LINKLIST for REPORT. The error messages shown occur because REPORT was misspelled as REPART in the JCL.



47

Copyright © 2006 CA. All trademarks, trade names, services marks and logos referenced herein belong to their respective companies.

CSV003I REQUESTED MODULE **REPART** NOT FOUND

### Explanation:

The system could not find the module entry point, **mod**, that a LINK, XCTL, ATTACH, or LOAD macro specified. This can result from having an alias which is not associated with an existing primary name, or an alias that matches a primary name in another concatenated library.

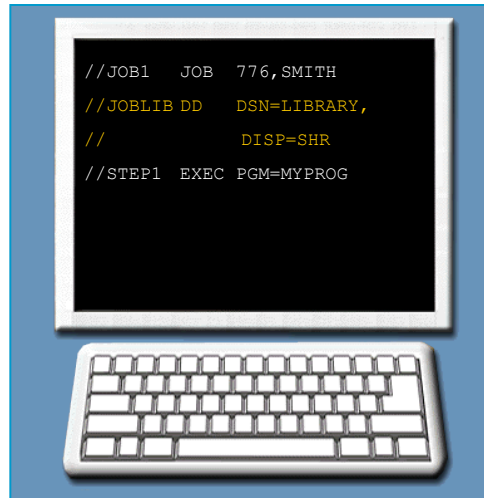
### The system library.

## Private program libraries.

You can use private libraries to store programs.

If you want to invoke a program called MYPROG which is stored in a private library, you must tell the operating system the name of the private library by coding a special DD statement named JOBLIB before the first EXEC statement in the job.

The JOBLIB DD statement causes the system to search a private library before searching  
48  
SYS1.LINKLIB.



If the system does not find the program in the library specified by the JOBLIB DD statement, then it goes on to search the SYS1.LINKLIB and the libraries defined in LINKLIST next. This search sequence repeats for every step in the job.

Use the JOBLIB DD statement to identify a private library that the system is to search for the program named in each EXEC statement PGM parameter in the job. Only if the system does not find the program in the private library, does it search the system libraries.

## Syntax

```
//JOBLIB DD parameter[,parameter]... [comments]
```

**The system library.**

**Are we on track?**

**From the code determine which library the system would search first for UTILITY.**

```
//MYJOB1 JOB 514,SMITH  
//JOB LIB DD DSN=USER1  
//STEP1 EXEC PGM=UTILITY
```



- A. SMITHLIB private library.**
- B. SYS1.LINKLIB or LINKLIST.**
- C. USER1 private library.**

49

Copyright © 2006 CA. All trademarks, trade names, services marks and logos referenced herein belong to their respective companies.

The correct answer is C.

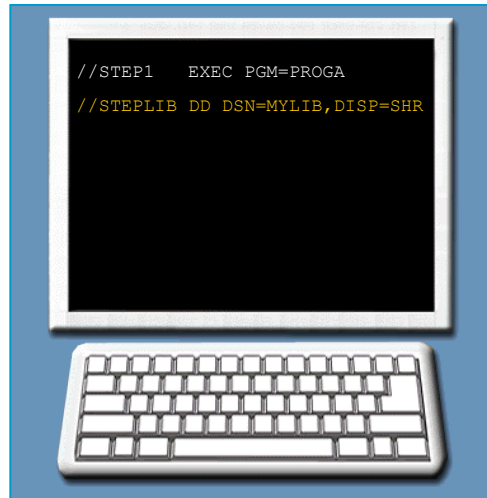
### The STEPLIB DD statement.

#### The STEPLIB DD statement.

If most of the programs for a job reside in SYS1.LINKLIB or LINKLIST and only a few are in private libraries, it makes more sense to direct the system to search a private library on a step-by-step basis.

This saves processing time by eliminating unnecessary searching.

To search a private library directly you use a special DD statement called STEPLIB DD statement as shown.



50

Copyright © 2006 CA. All trademarks, trade names, services marks and logos referenced herein belong to their respective companies.

The STEPLIB DD statement can be placed anywhere in a job step but it typically appears after an EXEC statement.

Use the STEPLIB DD statement to identify a private library that the system is to search for the program named in the EXEC statement PGM parameter. If the system does not find the program in the private library, only then does the system search the system libraries.

The private library is a partitioned data set (PDS) on a direct access device. Each member is an executable, user-written program.

#### Syntax

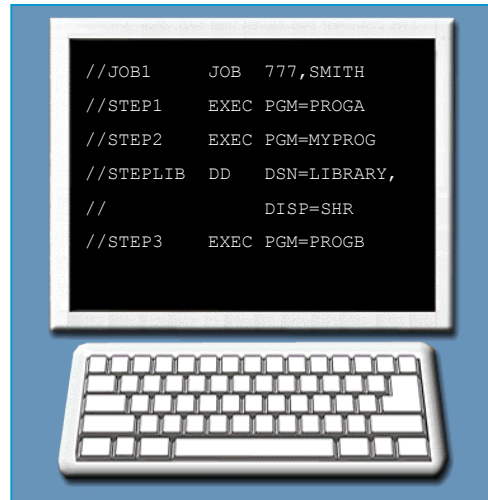
```
//STEPLIB DD parameter[,parameter]... [comments]
```

### The STEPLIB DD statement.

## Comparison between JOBLIB and STEPLIB DD statements.

Just like a JOBLIB DD statement, the STEPLIB DD statement searches a private library for a specified program.

But, the STEPLIB DD statement is in effect only for the duration of the step it follows.



51

Copyright © 2006 CA. All trademarks, trade names, services marks and logos referenced herein belong to their respective companies.

Continued... ➡

In the example, when executing STEP1, the system will search in SYS1.LINKLIB and LINKLIST for PROGA.

But when executing STEP2, the system will look first in the private library named LIBRARY for MYPROG, ignoring SYS1.LINKLIB and LINKLIST. If it does not find MYPROG in LIBRARY it will search in SYS1.LINKLIB and LINKLIST.

And in STEP3 the system will look for PROGB in SYS1.LINKLIB and LINKLIST but not in LIBRARY.



**The STEPLIB DD statement.**

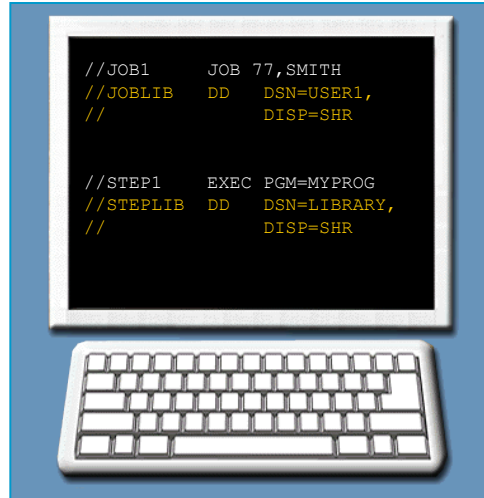
**Comparison between JOBLIB and STEPLIB DD statements.**

**What if a STEPLIB and JOBLIB both appear in a job?**

In this case the STEPLIB overrides the JOBLIB. The system ignores JOBLIB and does not search it in the step.

If the system does not find the program in library specified by the STEPLIB, it searches the system libraries (SYS1.LINKLIB and LINKLIST) next.

If it does not find the program there, the job step abends.



52  
Copyright © 2000, International Business Machines Corporation. All rights reserved. IBM, the IBM logo, and the e-business logo are trademarks of International Business Machines Corporation. Other names, services marks and logos referenced herein belong to their respective companies.

**Coding EXEC statements.**

**Unit summary.**

Now that you have completed this unit, you should be able to:

- **Code an EXEC statement to specify a program to be executed.**
- **Correct coding errors in an EXEC statement.**
- **Identify which JCL statement has caused a "PROGRAM NOT FOUND" error message.**
- **Identify the system library from which programs are retrieved at execution time.**
- **Identify the DD statement names used to specify a private library from which programs are retrieved at execution time.**
- **Select the place in the job stream where STEPLIB and JOBLIB DD statements should be located.**
- **Code a JOBLIB DD statement.**

53

© 2008 IBM Corporation. All rights reserved. IBM and the IBM logo are trademarks of International Business Machines Corporation in the United States and other countries. All other trademarks are the property of their respective companies.

We did not mention every parameters the EXEC statement can have. For more information see IBM books:

„z/OS MVS JCL Reference“ and „z/OS MVS JCL User’s Guide“.