JCL

Chapter b1
Using special DD statements

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

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

Chapter c1. Nested procedures

Chapter c2. Cataloging procedures

Chapter c3. Using utility programs

Chapter c4. Sample utility application

Chapter b1

Using special DD statements

Course objectives.

Be able to:

- **Use backward reference feature with the PGM, DSN, VOL, and DCB parameters.**
- **Code statements to concatenate data sets and create dummy data sets.**
- **Code statements to produce storage dumps.**
- **Invoke procedures for frequently-used job steps.**
- **Analyze the components of a job log to correct common errors in JCL code.**
- **Assign values to DDNAME and symbolic operands at the time of executing a procedure.**

Using backward reference.

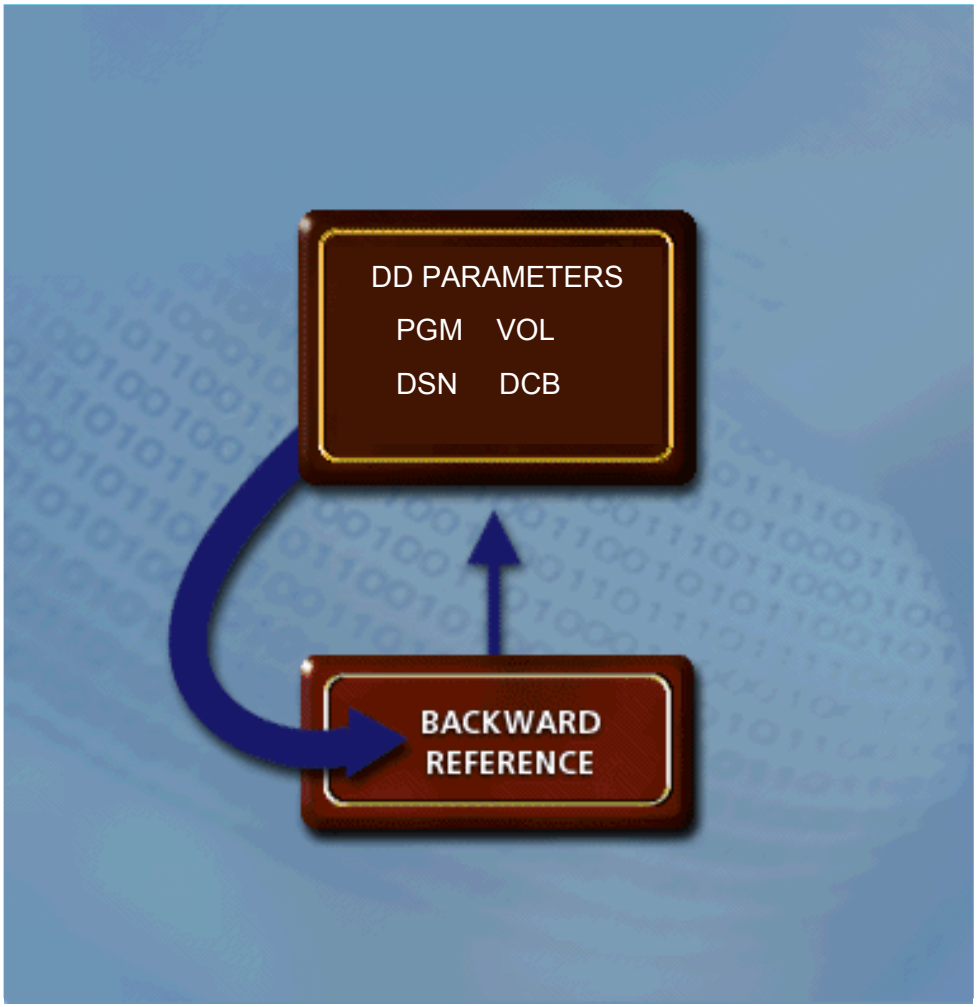
Job Control Language.

What is Backward Reference?

A typical JCL job step may use or create a number of data sets, each requiring a variety of parameter values.

Backward reference is a coding technique that directs the system to copy parameter values from preceding DD statements within the current job.

This technique is more efficient as it saves the programmer from repetitive coding of information.



Types of backward references.

Four common backward references are:

- **PGM Reference: Points to a previous data set to specify a program name.**
- **DSN Reference: Points to a previous data set name.**
- **VOL Reference: Points to a previous volume serial number.**
- **DCB Reference: Points to DCB attributes defined in another previous DD statement.**

Using backward reference.

Syntax for backward reference.

The general form of a backward reference is as follows:

- To refer back to a prior DD statement within the same step:

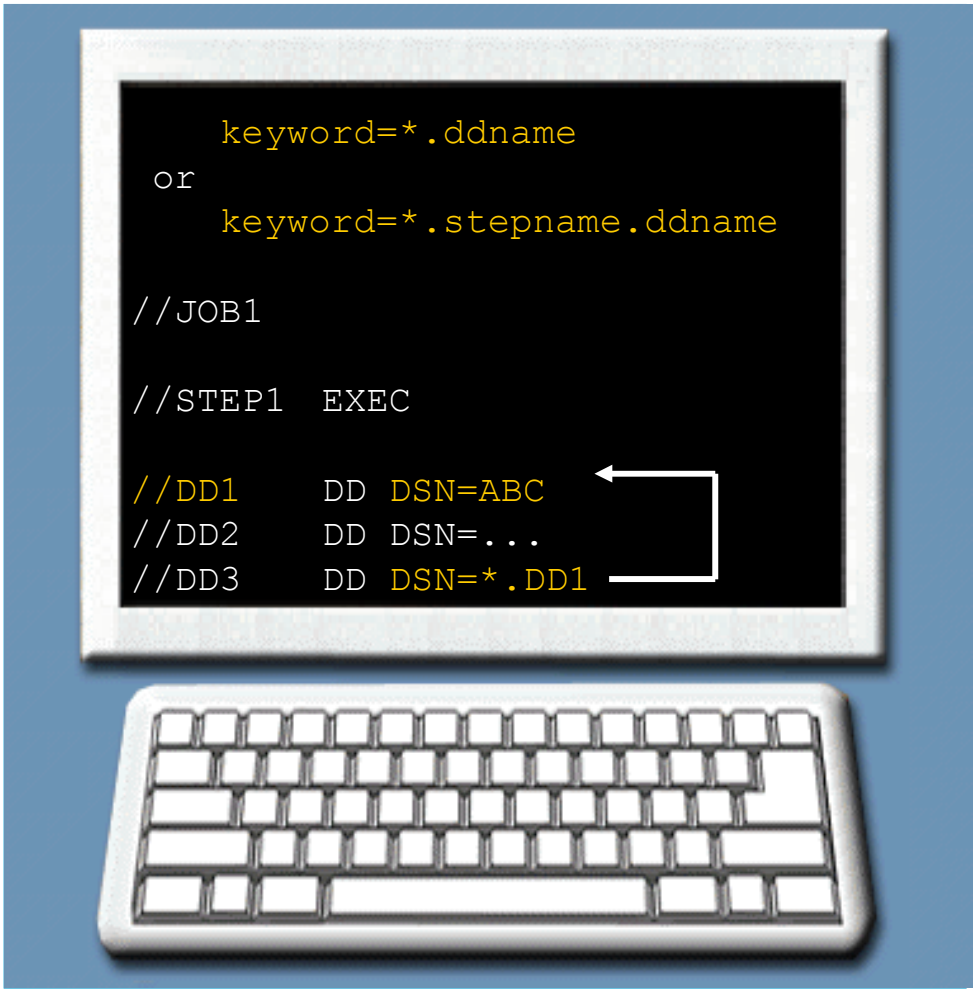
  Keyword=*.ddname

- To refer back to a DD statement in a prior step:

  Keyword=*.stepname.ddname

- To refer back to a DD statement contained in cataloged procedure called by a previous step:

  Keyword=*.stepname.procstep.ddname



Using backward reference.

Are we on track?

The general form of a backward reference to a DD statement in a previous job step is keyword =_____.

Using backward reference.

Are we on track?

Match the backward reference with the parameter to which it points.

1. PGM reference	A. A previous volume serial number.
2. DSN reference	B. A previous data set specifying a program name.
3. VOL reference	C. DCB attributes defined in a previous DD statement.
4. DCB reference	D. A previous data set name.

Using backward reference.

PGM of backward references.

What is PGM Backward Reference?

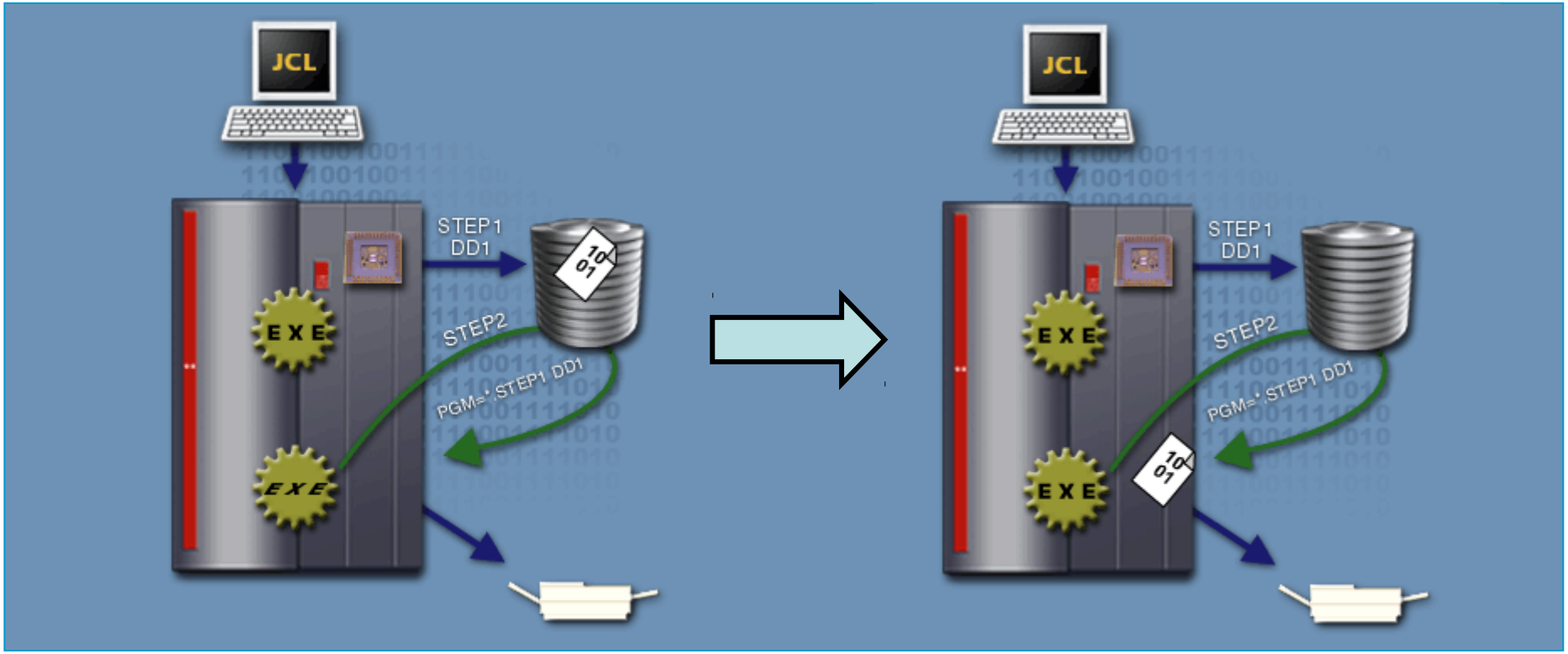
A PGM backward reference is a coding technique that points to a prior DD statement which specifies a member of a program library.

How does this technique help?

A PGM backward reference is useful in a program development environment, in which the output from one job step (typically a linkage edit step) may become the program to execute in a subsequent step. In such a case, instead of naming the program, you can code a PGM backward reference.

Using backward reference.

Syntax for PGM backward reference.

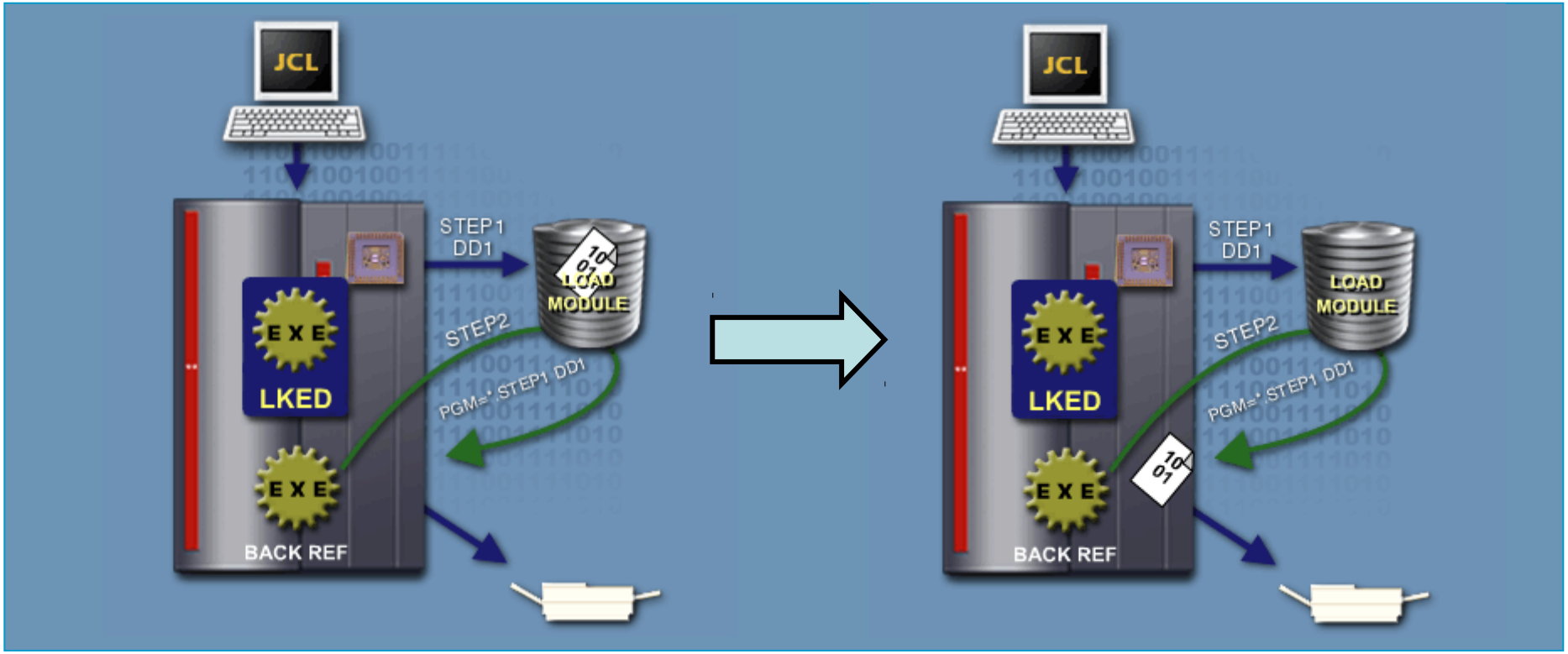


The general form of a PGM backward reference is as follows:

```
//STEP   EXEC PGM=*.stepname.ddname
```

Using backward reference.

PGM backward reference – example 1.



A PGM backward reference is often used following a linkage edit step, in which a load module (program) is stored in a temporary data set. PGM backward reference is used in coding a later step that executes the program. The reference specifies the data set containing the program from the previous step.

Using backward reference.

PGM backward reference – example 2.

In the example shown, the LINKEDIT program instructs the system to place a load module in a temporary library.

The ddname is SYSLMOD and the data name is &&GOSET(GO).

The DISP parameter specifies that the data is NEW and is to be PASSed to another step.

STEPA executes the program, using a PGM backward reference.

```
//LKED     EXEC   PGM=LINKEDIT
//SYSLMOD  DD     DSN=&&GOSET(GO),
//                DISP=(NEW,PASS),
// UNIT=SYSDA,SPACE=(1024,(200,20,1))
//STEPA    EXEC   PGM=*.LKED.SYSLMOD
```

Using backward reference.

Are we on track?

Assume that in step STEPC of a job, you want to execute PROGB using a PGM backward reference. The program is specified in STEPA on a DD statement with ddname LKEDOUT. Complete the following code.

//STEPC EXEC PGM=_____

Using backward reference.

Are we on track?

Which of the following statements are true for a PGM backward reference?

A. It is coded on DD statement.

B. It often follows a LINKEDIT step.

C. It points to the DD statement specifying the program you want to execute.

Using backward reference.

DSN backward reference.

What is DSN Backward Reference?

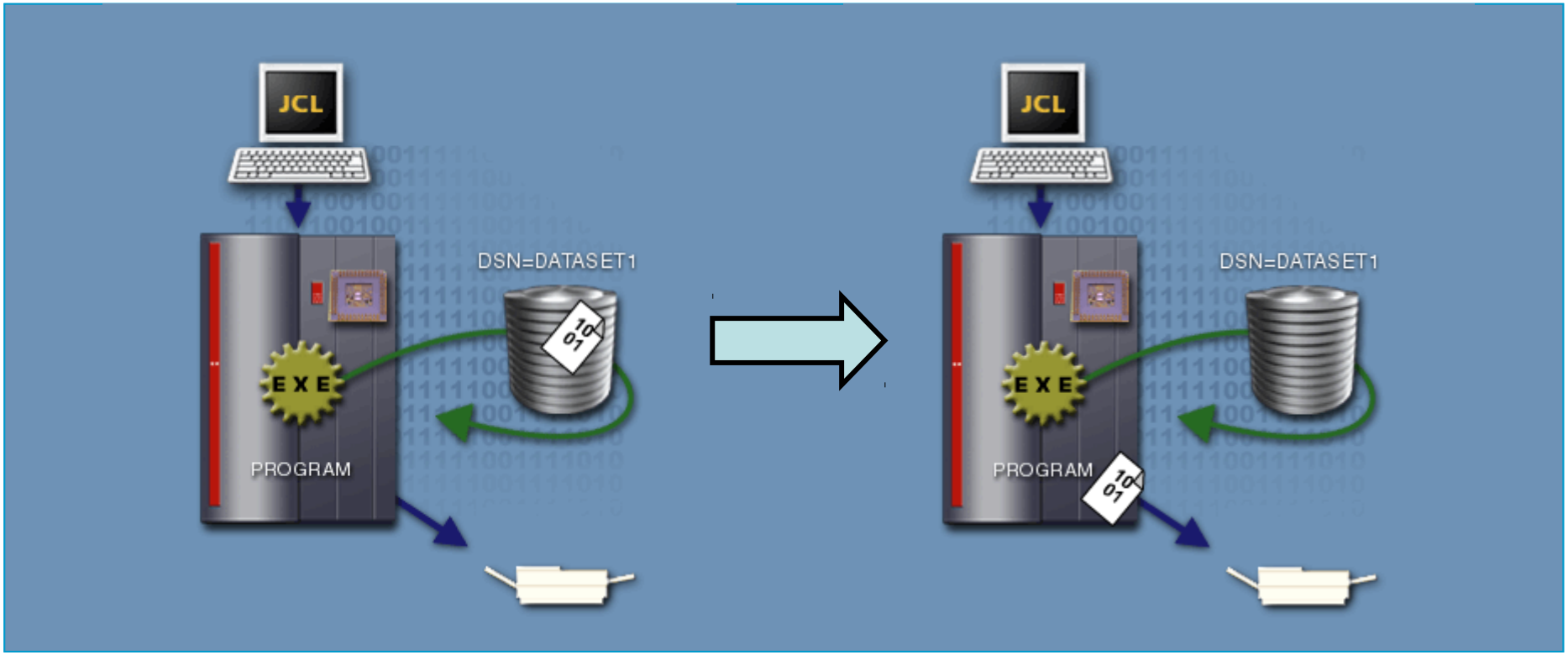
The DSN backward reference is a coding technique that refers to a prior DD statement that names the data set you want to process.

How does this technique help?

This technique is useful when coding jobs that consist of several steps, with multiple references to the same data set. The reference can also be used to retrieve temporary data sets in subsequent job steps, without knowing the name.

Using backward reference.

Syntax for DSN backward reference.



The general form for the DSN backward reference is as follows:

```
DSN=*.stepname.ddname
```

DSN backward reference – an example.

Consider a payroll job consisting of several steps, all referring to the same data set. The job needs to be executed each week using a data set that contains the week's transactions.

This requires that, each week the data set name must be changed in the order WEEK1, WEEK2 and so on.

By using a DSN backward reference, the data set can be retrieved each week by changing only one DD statement, DD1.

```
//STEP1   EXEC   PGM=PROG1
//DD1     DD     UNIT=SYSDA,
//               VOL=SER=PACK12,
//               SPACE=(800,(200,20,2)),
//               DISP=(NEW,PASS),
//               DSN=WEEK1
//STEP2   EXEC   PGM=PROG2
//DD2     DD     DSN=*.STEP1.DD1,
//               DISP=(OLD,KEEP)
```

Continued...

Using backward reference.

Are we on track?

A DSN backward reference points to a _____ in a prior DD statement.

Using backward reference.

Are we on track?

Code a DSN backward reference that refers to a data set in STEP2, ddname (DD3).

DSN= _____

Using backward reference.

VOL backward reference.

What is VOL Backward Reference?

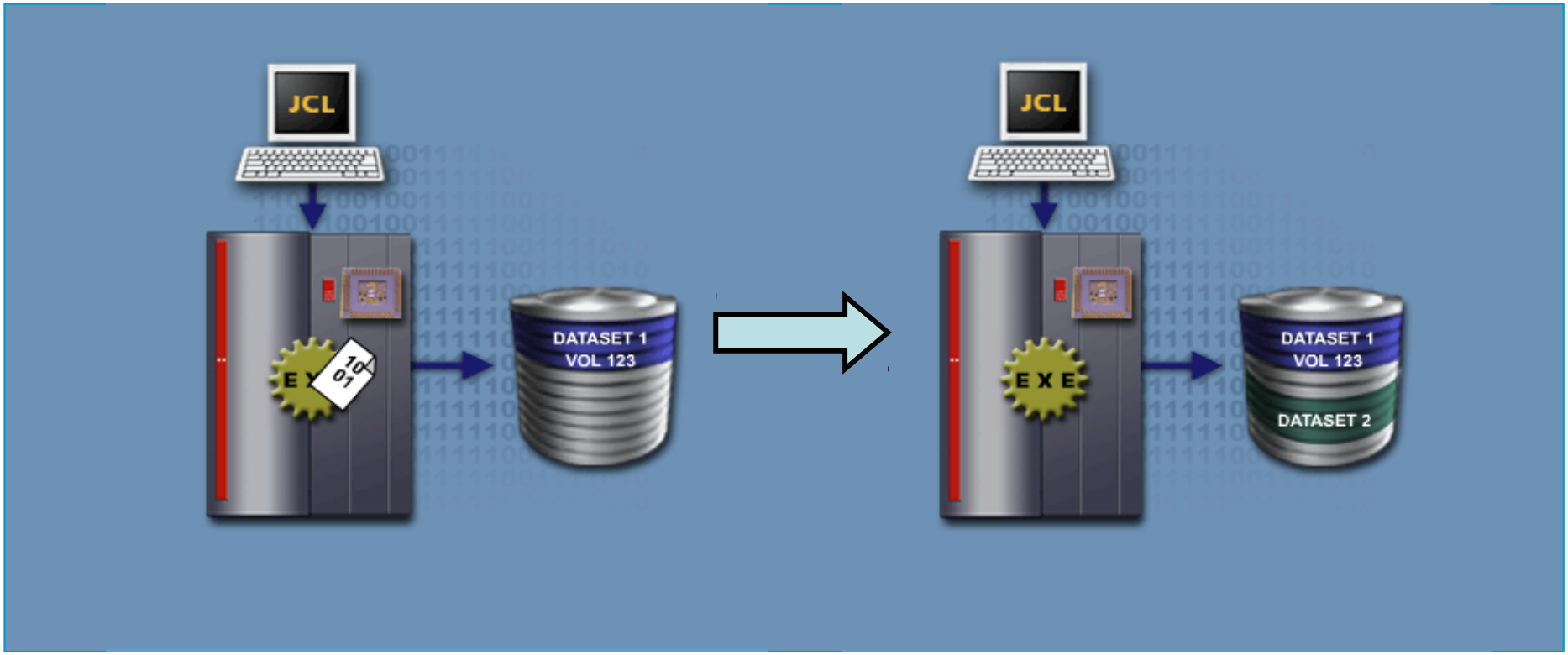
A VOL backward reference is a coding technique that points to the volume serial number of an existing data set.

How does this technique help?

The VOL backward reference is useful when you want to create a new data set on the same volume on which an existing data set resides, but you do not know the volume identification.

Using backward reference.

Syntax for VOL backward reference.



The general form of the VOL backward reference is shown below:

```
//ddname    DD    …VOL=REF=dsname
```

or

```
//ddname    DD    …VOL=REF=*.stepname.procstepname.ddname
```

Using backward reference.

VOL backward reference – example 1.

Consider an example where PROGA creates and catalogs a data set named XYZ. XYZ is to reside on the same volume as an existing, previously catalogued data set named ABC.

To refer the system to data set ABC, a VOL backward reference can be coded as follows:

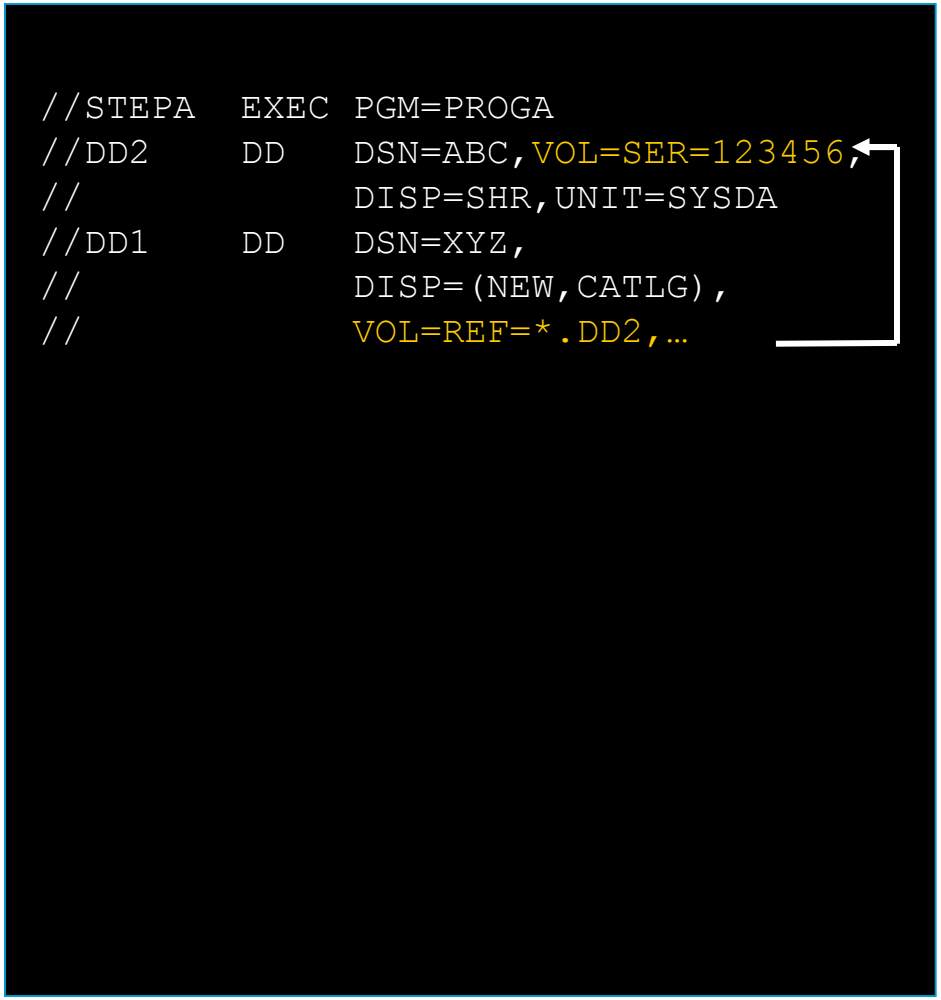
```
//STEP1   EXEC  PGM=PROGA
//DD1      DD   DSN=XYZ,
//             DISP=(NEW,CATLG),
//             VOL=REF=ABC
```

Using backward reference.

VOL backward reference – example 2.

In this example the backward reference refers to a specific volume serial number coded on a prior DD statement.

The data set XYZ will be created on the volume referred to by the DD statement DD2 (volume 123456).



Using backward reference.

Are we on track?

Code a VOL backward reference when:

data set XXX will reside on the same volume as data set YYY.

Using backward reference.

Are we on track?

Code a VOL backward reference when:

data set XXX will be created on the volume identified in the DD statement with ddname DD1.

Using backward reference.

Are we on track?

Code a VOL backward reference when:

data set XXX will be created on the volume identified in STEPC as DD2.

Using backward reference.

Are we on track?

Match the underlined statements in the code with the definitions in the column on the right.

1. VOL=REF=LMN	A. stepname.ddname
2. VOL=REF=*.DD1	B. dsname
3. VOL=REF=*.STEP1.DD1	C. ddname

Using backward reference.

DCB backward reference.

What is DCB Backward Reference?

DCB backward reference is a coding technique that allows you to copy a list of attributes from a prior DD statement in the same or previous job step.

How does this technique help?

This coding technique can be used to ensure that the DCB parameters are consistent within the job.

It can also be used to override or add to the subparameters coded on a previous statement.

Using backward reference.

Syntax for DCB backward reference.

The general form is as follows:

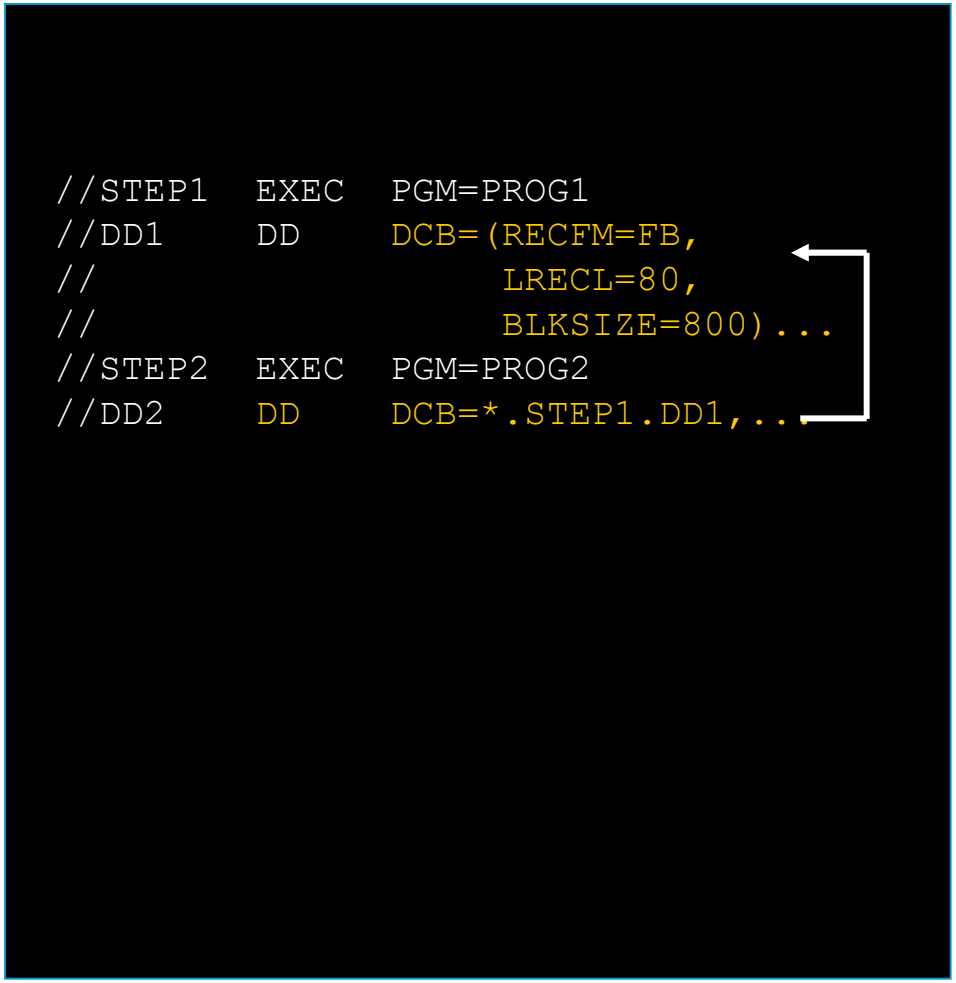
```
//ddname DD
   DCB=*.stepname.ddname
```

Using backward reference.

DCB backward reference – an example.

Assume that in STEP2 you want to create a data set with the same parameters as a data set in STEP1.

The code shown ensures that the attributes on the DD2 statement are the same as those on the DD1 statement.



Using backward reference.

DCB backward reference - overriding.

A DCB backward reference can also be used to override or add to the subparameters coded on a previous statement. The format for overriding a previous statement is as follows:

```
DCB=(*.stepname.ddname,list-of
attributes)
```

The values of the DCB parameters being referred will be overridden by the values that are being coded. Any attributes that do not match the DCB being referred will be added.

Continued...

Using backward reference.

DCB backward reference - overriding.

For example, notice the DCB characteristics in statement DD1 below:

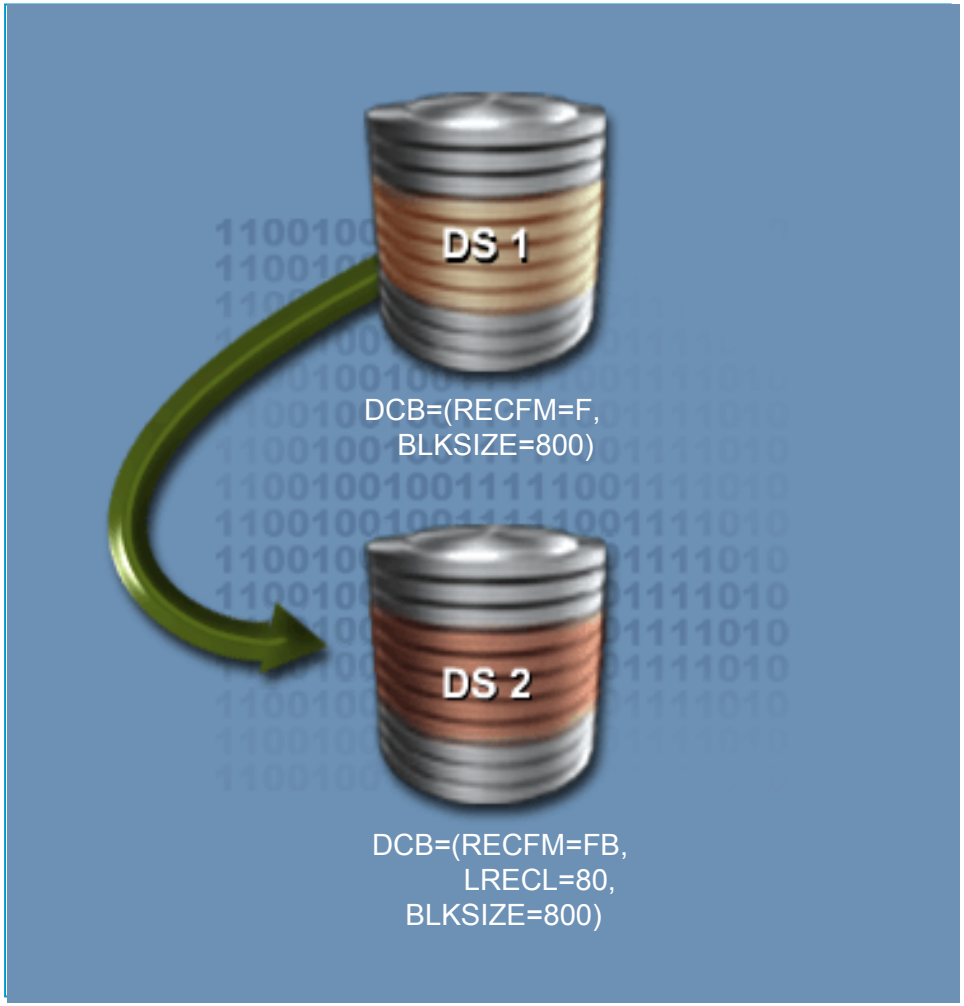
```
//STEP3    EXEC  PGM=PROG3
//DD1      DD    DCB=(RECFM=F,
//                 BLKSIZE=800),…
```

The following override statement:

```
//DD2      DD    DCB=(*.DD1,
//                 RECFM=FB,LRECL=80)
```

would result in these DCB characteristics:

```
//DD2      DD    DCB=(RECFM=FB,
//                  LRECL=80,BLKSIZE=800)
```



Using backward reference.

Are we on track?

The portion of the job stream shown below contains JCL statements, some of which are incomplete.

```
1. //COMPILE   EXEC   PGM=PL1
2. //COMPOUT   DD     UNIT=SYSDA,VOL=SER=PACK12,
   //                 DISP=(NEW,PASS),DSN=&&A
3. //LKED      EXEC   PGM=LINKEDIT
4. //LKEDIN    DD     DISP=OLD,DSN=__________
5. //SYSLMOD   DD     DISP=(NEW,PASS),DSN=&&GOSET(GO),
   //                 VOL=_______________
6. //GO        EXEC   PGM=___________
7. //MYDATA    DD     DSN=MYDATA,DISP=(NEW,CATLG),
   //                 VOL=SER=…,SPACE=(800,50),
   //                 DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
8. //TEMP      DD     UNIT=SYSDA,DCB=___________
```

Complete those statements by coding the appropriate backward references as follows:

The data set name in statement 4 refers to statement 2.

Using backward reference.

Are we on track?

The portion of the job stream shown above contains JCL statements, some of which are incomplete.

```
1. //COMPILE   EXEC   PGM=PL1
2. //COMPOUT   DD     UNIT=SYSDA,VOL=SER=PACK12,
   //                 DISP=(NEW,PASS),DSN=&&A
3. //LKED      EXEC   PGM=LINKEDIT
4. //LKEDIN    DD     DISP=OLD,DSN=*.COMPILE.COMPOUT
5. //SYSLMOD   DD     DISP=(NEW,PASS),DSN=&&GOSET(GO),
   //                 VOL=_______________
6. //GO        EXEC   PGM=____________
7. //MYDATA    DD     DSN=MYDATA,DISP=(NEW,CATLG),
   //                 VOL=SER=…,SPACE=(800,50),
   //                 DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
8. //TEMP      DD     UNIT=SYSDA,DCB=____________
```

Complete those statements by coding the appropriate backward references as follows:

The volume in statement 5 refers to statement 2.

Using backward reference.

Are we on track?

The portion of the job stream shown above contains JCL statements, some of which are incomplete.

```
1.  //COMPILE   EXEC   PGM=PL1
2.  //COMPOUT   DD     UNIT=SYSDA,VOL=SER=PACK12,
    //                 DISP=(NEW,PASS),DSN=&&A
3.  //LKED      EXEC   PGM=LINKEDIT
4.  //LKEDIN    DD     DISP=OLD,DSN=*.COMPILE.COMPOUT
5.  //SYSLMOD   DD     DISP=(NEW,PASS),DSN=&&GOSET(GO),
    //                 VOL=REF=*.COMPILE.COMPOUT
6.  //GO        EXEC   PGM=___________
7.  //MYDATA    DD     DSN=MYDATA,DISP=(NEW,CATLG),
    //                 VOL=SER=…,SPACE=(800,50),
    //                 DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
8.  //TEMP      DD     UNIT=SYSDA,DCB=___________
```

Complete those statements by coding the appropriate backward references as follows:

The program in statement 6 refers to statement 5.

Using backward reference.

Are we on track?

The portion of the job stream shown above contains JCL statements, some of which are incomplete.

```
1. //COMPILE   EXEC   PGM=PL1
2. //COMPOUT   DD     UNIT=SYSDA,VOL=SER=PACK12,
   //                 DISP=(NEW,PASS),DSN=&&A
3. //LKED      EXEC   PGM=LINKEDIT
4. //LKEDIN    DD     DISP=OLD,DSN=*.COMPILE.COMPOUT
5. //SYSLMOD   DD     DISP=(NEW,PASS),DSN=&&GOSET(GO),
   //                 VOL=REF=*.COMPILE.COMPOUT
6. //GO        EXEC   PGM=*.LKED.SYSLMOD
7. //MYDATA    DD     DSN=MYDATA,DISP=(NEW,CATLG),
   //                 VOL=SER=…,SPACE=(800,50),
   //                 DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
8. //TEMP      DD     UNIT=SYSDA,DCB=____________
```

Complete those statements by coding the appropriate backward references as follows:

The DCB attributes in statement 8 refer to statement 7.

Using backward reference.

Glossary.

DD Statement
A JCL statement that describes each data set used within a job.

DDname
A unique name given to each data set used in a job step.

Job Step
The JCL statements that control the execution of a program and request the resources needed to run the program. A job step is identified by an EXEC statement.

Parameter Values
Information that follows a keyword parameter and an equal sign.

PGM
An EXEC statement parameter that names the program to execute.

DSN
A DD statement parameter that names the data set.

Using backward reference.

Glossary.

VOL
A parameter on a DD statement that requests a specific volume.

DCB
Data Control Block. A parameter on a DD statement that describes the attributes of a data set, such as block size and record format.

Load Module
An executable program that results from a link edit step.

SYSLMOD
DD name used by the linkage editor to write its output (a load module).

DISP
Describes the status of a data set to the system and tells the system what to do with the data set after termination of the step or job.

Concatenating data sets.

Data set concatenation – definition.

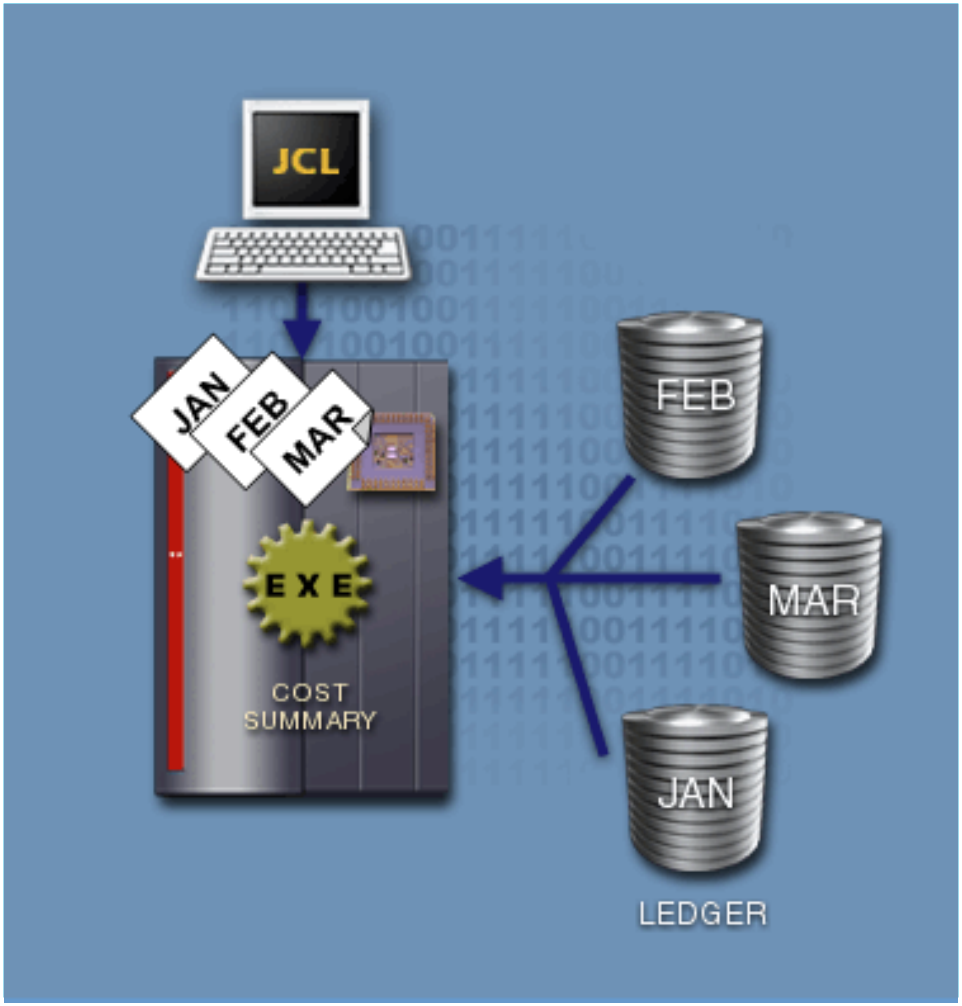
What is data set concatenation?

A programmer can code DD statements to request that several data sets be concatenated.

Data set concatenation enables the system to process several separate physical data sets as one logical data set.

Data set concatenation – an example.

- Consider a cost ledger to produce a monthly cost summary file. At the year end, it is required to process all 12 monthly data sets to produce an annual report. All the data sets are concatenated so they can be processed sequentially.
- In this example, the program uses a ddname of LEDGER and the monthly data sets are named JAN, FEB, MAR and so on.
- The operating system draws the concatenated data sets sequentially, treating them as a single logical data set.



Concatenation of data sets.

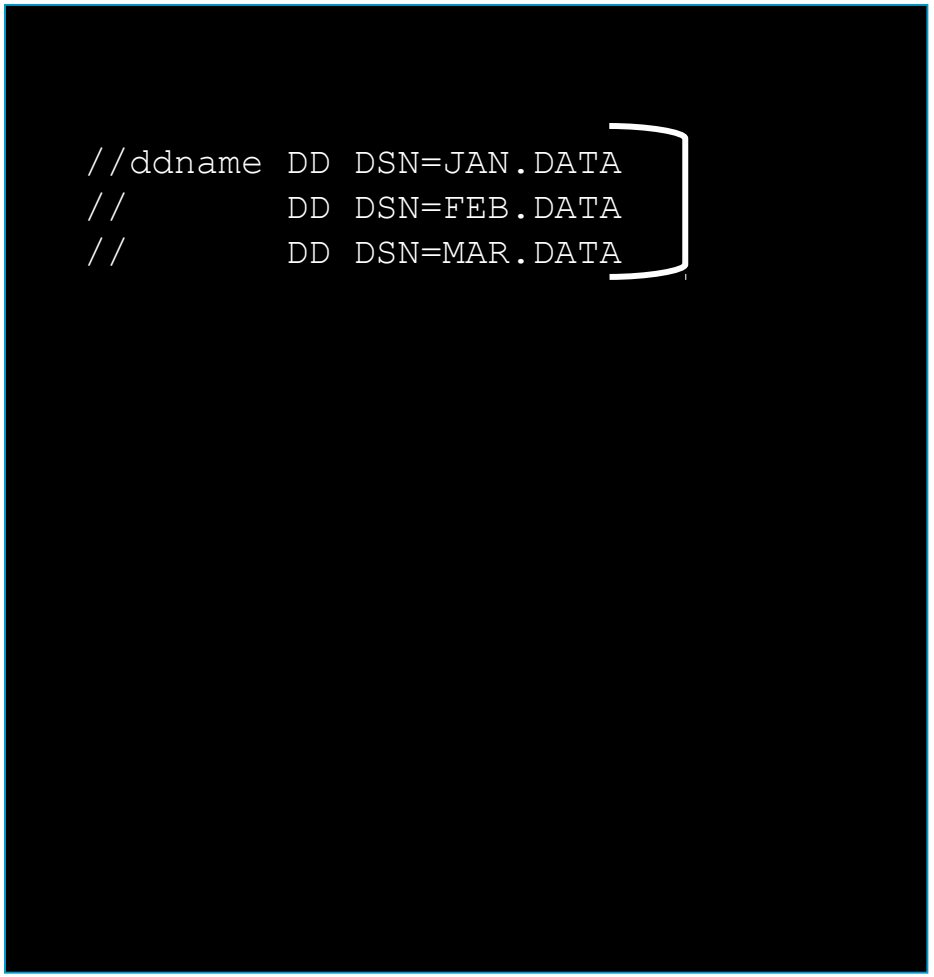
How to concatenate data sets?

Following steps are involved in concatenating data sets:

1. Code a standard DD statement for the first data set only.

2. Add a DD statement without a ddname for each data set to be concatenated.

3. Sequence the statements in the order they are to be processed.



Concatenation of data sets.

How concatenation is useful?

Using concatenation, a program can be run with one or several input data sets by merely changing the DD statement.

While concatenating data sets the following points must be considered:

- **The concatenated data sets must have the same (or compatible) DCB subparameters. Namely, RECFM, LRECL and BLKSIZE.**

- **A maximum of 255 sequential and 16 partitioned data sets can be concatenated.**

Concatenating data sets.

JCL for data set concatenation – an example.

The JCL here shows the concatenation of the monthly data sets considered in the LEDGER example.

The last data set concatenated to LEDGER is DEC.

The occurrence of the ddname SUM indicates that the data set (ACCT.1999) is to be processed separately from the LEDGER data sets.

```
//LEDGER DD DSN=JAN,DISP=SHR
//       DD DSN=FEB,DISP=SHR
           .
           .
           .
//         DD DSN=DEC,DISP=SHR
//SUM     DD DSN=ACCT.1999,DISP=SHR
```

Concatenating data sets.

Are we on track?

Consider three data sets named, CUST.HISTORY.JUL, CUST.HISTORY.APR and CUST.HISTORY.JAN which are to be processed in this order. They are to be concatenated to CUST.HISTORY.OCT, to create a master customer list.

Put the following statements in order.

A.
```
//          DD DSN=CUST.HISTORY.APR
```

B.
```
//MASTCUST DD DSN=CUST.HISTORY.OCT
```

C.
```
//          DD DSN=CUST.HISTORY.JAN
```

D.
```
//          DD DSN=CUST.HISTORY.JUL
```

Glossary.

Concatenated data sets
Data sets that are separate physically, but processed sequentially as one logical data set.

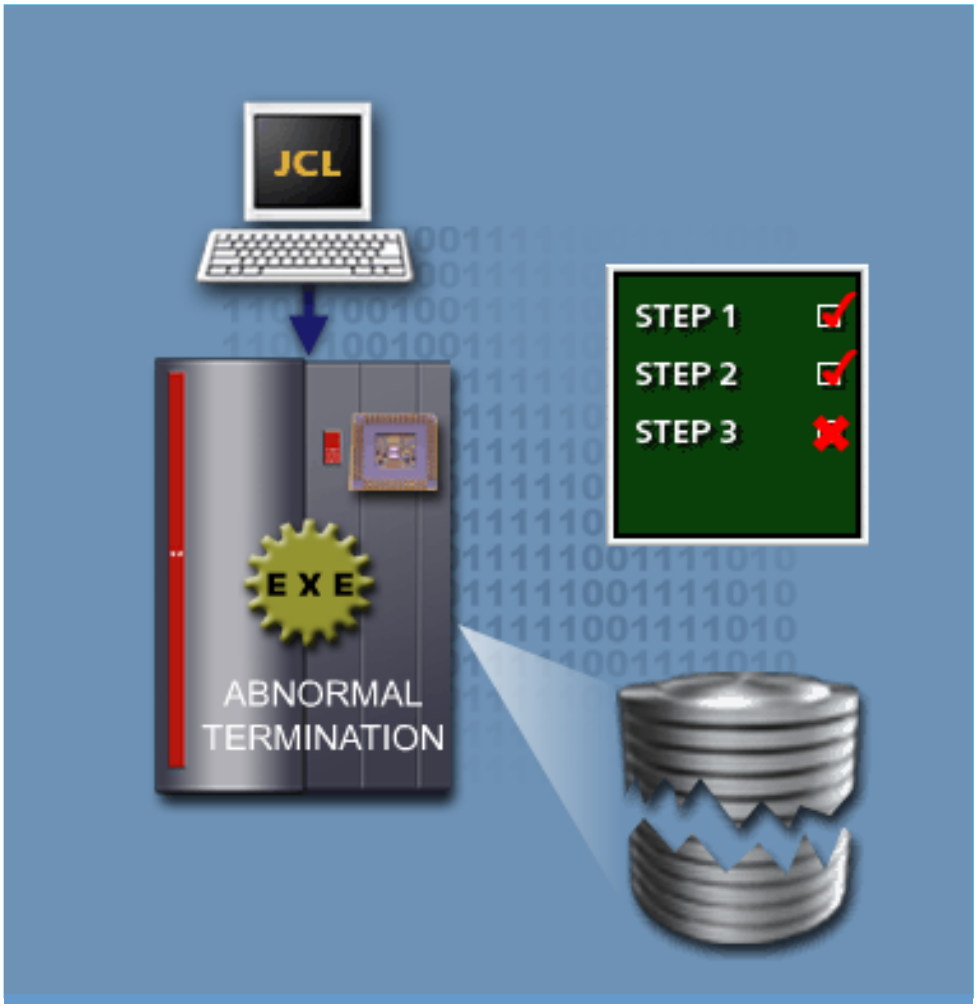
Dummy data sets.

Overview.

Each data set that is referred by a program should have a ddname. The JCL for the program must contain the corresponding DD statements.

If a data set is not coded by a DD statement, then the program will abnormally end (ABEND) as shown.

When an input data set is optional for the program's processing or when an output data set is not required dummy data sets can be used.



Dummy data set.

What is a dummy data set?

A dummy data set is a data set for which all Input or Output (I/O) operations are bypassed.

A special DD statement, DD DUMMY, is used to ignore a data set during the execution of a program.

How does it work?

When a data set is assigned dummy status, all I/O operations are bypassed and device allocation, space allocation and data set disposition are ignored.

Specifying dummy data sets.

Dummy data sets can be specified in DD statements by doing one of the following:

- **Coding DUMMY as the first DD parameter**

syntax:

```
//DDname        DD      DUMMY
```

- **Coding DSN=NULLFILE**

syntax:

```
//DDname        DD      DSN=NULLFILE
```

Dummy data sets – an example.

Consider a payroll program named PAY that processes separate input data sets. The ddname TIMECDS refers to weekly time cards and the ddname ADJUST refers to adjustments to previous pay period information.

The job stream must include:

```
//STEPA     EXEC  PGM=PAY
//TIMECDS   DD    ---
//ADJUST    DD    ---
                  .
                  .
```

Dummy data sets – an example.

Even if there are no adjustments for PAY process, DD statement for ADJUST must be included.

To tell the system that there is no ADJUST data set code can be written as follows:

```
//STEPA      EXEC   PGM=PAY
//TIMECDS    DD     ----
//ADJUST     DD     DUMMY
```

If the data set described by the DD statement named ADJUST is referred to by the PAY program, an immediate end-of-file occurs. The program will continue as if it has processed the entire data set.

Dummy data sets.

Are we on track?

You can specify a dummy data set by coding DSN=_____ on the DD statement.

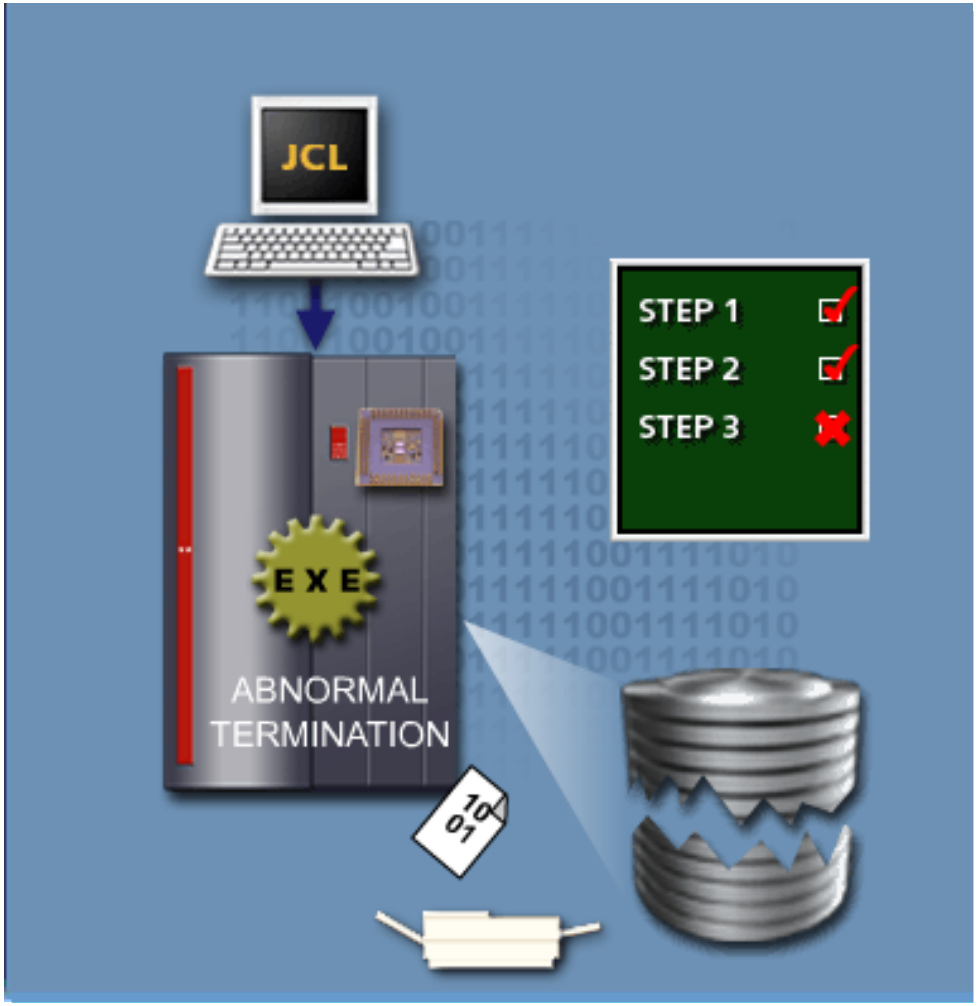
Storage dumps.

What are Storage Dumps?

When a program abnormally terminates, storage dumps are used as a debugging tool to find clues to the cause for abnormal ending. Storage dumps are not the most effective debugging tool.

The main drawbacks of storage dumps are:

- They are difficult to read since they are in hexadecimal code.
- Printing storage dumps is time consuming.



Storage dumps.

Special DDnames.

These reserved ddnames request storage dumps in the event that a program terminates abnormally:

SYSUDUMP: **Requests a formatted dump of the processing program area. It is most generally used for debugging problem programs.**

SYSABEND: **Requests a formatted dump of the processing program area, system programs and the system control blocks. It is often spooled for printing, although it may be written onto any output device.**

SYSMDUMP: **Requests an unformatted dump of the processing program area and the system nucleus in machine readable form. It is generally directed to tape (or to direct access storage) to allow subsequent processing by a dump analysis utility.**

Handling storage dumps.

It is necessary to plan ahead for a possible storage dump.

To obtain a dump, the SYSUDUMP, SYSABEND, or SYSMDUMP DD statements must be coded in the JCL for each job step from which a dump needs to be obtained.

The example shown uses SYSUDUMP DD statement.

If STEP1 or STEP2 terminates abnormally, the system creates a dump of the program storage area.

```
//STEP1    EXEC  PGM=PROG1
//SYSDUMP  DD    SYSOUT=X
//DD1      DD    ...
//STEP2    EXEC  PGM=PROG2
//SYSUDUMP DD    SYSOUT=X
```

Are we on track?

Match the special ddname with its function

1. SYSABEND

A. Requests an unformatted dump in machine-readable form of the processing program area and the system nucleus.

2. SYSMDUMP

B. Requests a formatted dump of the processing program area and of the system control blocks.

3. SYSUDUMP

C. Requests a formatted dump of the processing program area.

Unit summary.

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

- **Code a DD statement to use information from preceding JCL statements.**
- **Identify the purpose of data set concatenation.**
- **Code the JCL to concatenate a data set.**
- **Code a DD statement to indicate that a data set is to be ignored for the current program execution.**
- **Identify the purpose of special ddnames.**

JCL

Chapter b1
Using special DD statements

Chapter b1

Using special DD statements

Course objectives.

Be able to:

- **Use backward reference feature with the PGM, DSN, VOL, and DCB parameters.**
- **Code statements to concatenate data sets and create dummy data sets.**
- **Code statements to produce storage dumps.**
- **Invoke procedures for frequently-used job steps.**
- **Analyze the components of a job log to correct common errors in JCL code.**
- **Assign values to DDNAME and symbolic operands at the time of executing a procedure.**

Using backward reference.

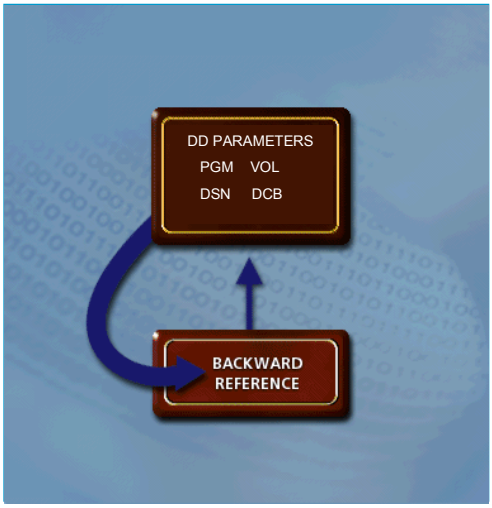
Job Control Language.

What is Backward Reference?

A typical JCL job step may use or create a number of data sets, each requiring a variety of parameter values.

Backward reference is a coding technique that directs the system to copy parameter values from preceding DD statements within the current job.

This technique is more efficient as it saves the programmer from repetitive coding of information.



Using backward reference.

Types of backward references.

Four common backward references are:

- **PGM Reference: Points to a previous data set to specify a program name.**
- **DSN Reference: Points to a previous data set name.**
- **VOL Reference: Points to a previous volume serial number.**
- **DCB Reference: Points to DCB attributes defined in another previous DD statement.**

Using backward reference.

Syntax for backward reference.

The general form of a backward reference is as follows:

- To refer back to a prior DD statement within the same step:

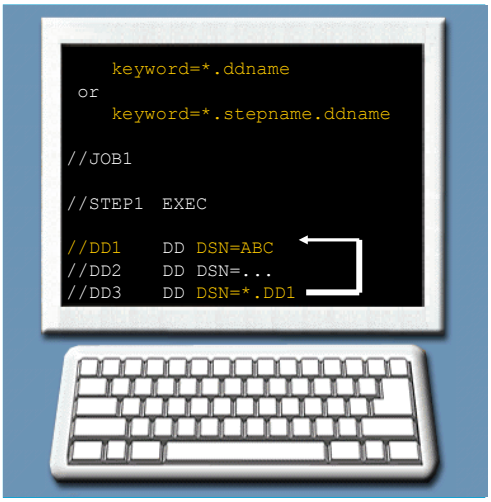
  Keyword=*.ddname

- To refer back to a DD statement in a prior step:

  Keyword=*.stepname.ddname

- To refer back to a DD statement contained in cataloged procedure called by a previous step:

  Keyword=*.stepname.procstep.ddname



9

You will learn more about cataloged procedures in Unit 2 and Unit 3.
The keyword in each statement is either PGM, DSN, VOL or DCB.

Using backward reference.

Are we on track?

The general form of a backward reference to a DD statement in a previous job step is keyword =_____.

10

The correct answer is *.stepname.ddname

Using backward reference.

Are we on track?

Match the backward reference with the parameter to which it points.

1. PGM reference	A. A previous volume serial number.
2. DSN reference	B. A previous data set specifying a program name.
3. VOL reference	C. DCB attributes defined in a previous DD statement.
4. DCB reference	D. A previous data set name.

11

The correct answers is: 1 – B, 2 – D, 3 – A, and 4 – C.

Using backward reference.

PGM of backward references.

What is PGM Backward Reference?

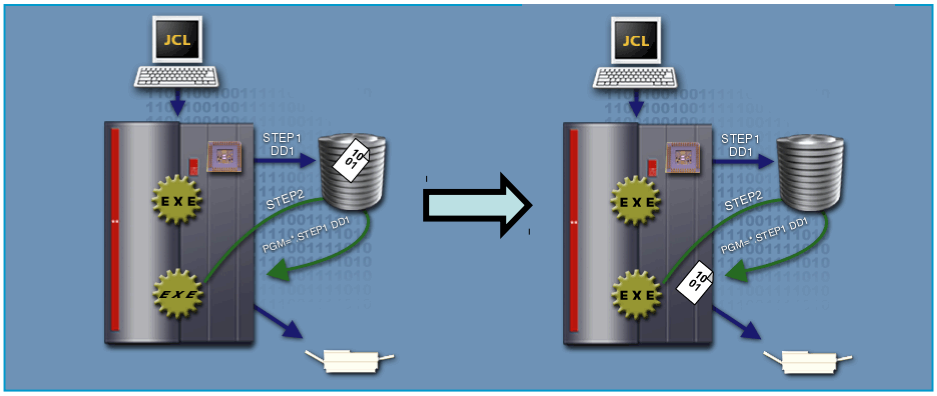
A PGM backward reference is a coding technique that points to a prior DD statement which specifies a member of a program library.

How does this technique help?

A PGM backward reference is useful in a program development environment, in which the output from one job step (typically a linkage edit step) may become the program to execute in a subsequent step. In such a case, instead of naming the program, you can code a PGM backward reference.

Using backward reference.

Syntax for PGM backward reference.



The general form of a PGM backward reference is as follows:

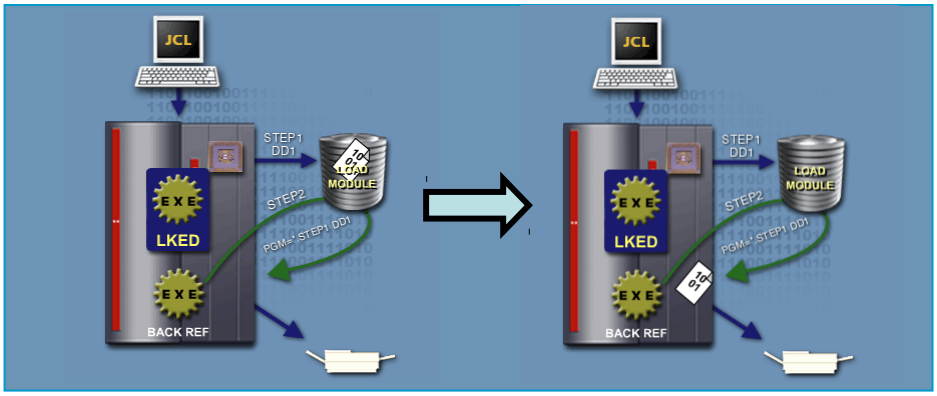
```
//STEP  EXEC PGM=*.stepname.ddname
```

13

A PGM backward reference is coded on the EXEC statement.

Using backward reference.

PGM backward reference – example 1.



A PGM backward reference is often used following a linkage edit step, in which a load module (program) is stored in a temporary data set. PGM backward reference is used in coding a later step that executes the program. The reference specifies the data set containing the program from the previous step.

14

Using backward reference.

PGM backward reference – example 2.

In the example shown, the LINKEDIT program instructs the system to place a load module in a temporary library.

The ddname is SYSLMOD and the data name is &&GOSET(GO).

The DISP parameter specifies that the data is NEW and is to be PASSed to another step.

STEPA executes the program, using a PGM backward reference.

Using backward reference.

Are we on track?

Assume that in step STEPC of a job, you want to execute PROGB using a PGM backward reference. The program is specified in STEPA on a DD statement with ddname LKEDOUT. Complete the following code.

//STEPC EXEC PGM=_____

16

The correct answer is *.STEPA.LKEDOUT

Using backward reference.

Are we on track?

Which of the following statements are true for a PGM backward reference?

A. It is coded on DD statement.

B. It often follows a LINKEDIT step.

C. It points to the DD statement specifying the program you want to execute.

The correct answers are B. and C.

Using backward reference.

DSN backward reference.

What is DSN Backward Reference?

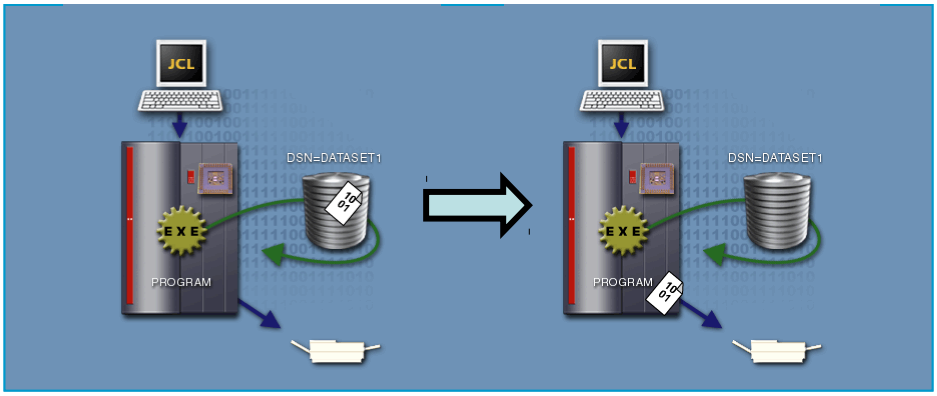
The DSN backward reference is a coding technique that refers to a prior DD statement that names the data set you want to process.

How does this technique help?

This technique is useful when coding jobs that consist of several steps, with multiple references to the same data set. The reference can also be used to retrieve temporary data sets in subsequent job steps, without knowing the name.

Using backward reference.

Syntax for DSN backward reference.



The general form for the DSN backward reference is as follows:

```
DSN=*.stepname.ddname
```

19

Using backward reference.

DSN backward reference – an example.

Consider a payroll job consisting of several steps, all referring to the same data set. The job needs to be executed each week using a data set that contains the week's transactions.

This requires that, each week the data set name must be changed in the order WEEK1, WEEK2 and so on.

By using a DSN backward reference, the data set can be retrieved each week by changing only one DD statement, DD1.

```
//STEP1  EXEC  PGM=PROG1
//DD1    DD    UNIT=SYSDA,
//             VOL=SER=PACK12,
//             SPACE=(800,(200,20,2)),
//             DISP=(NEW,PASS),
//             DSN=WEEK1
//STEP2  EXEC  PGM=PROG2
//DD2    DD    DSN=*.STEP1.DD1,
//             DISP=(OLD,KEEP)
```

Continued…

No subsequent steps using the backward reference need to be changed since they do not directly specify the data set name.

Note that backward reference points to the DD1 statement in STEP1. Each week, as the data set name changes, only the one DD statement, DD1, is changed.

Using backward reference.

Are we on track?

A DSN backward reference points to a _____ in a prior DD statement.

21

The correct answer is DDname.

Using backward reference.

Are we on track?

Code a DSN backward reference that refers to a data set in STEP2, ddname (DD3).

DSN= _____

22

The correct answer is *.STEP2.DD3

Using backward reference.

VOL backward reference.

What is VOL Backward Reference?

A VOL backward reference is a coding technique that points to the volume serial number of an existing data set.

How does this technique help?

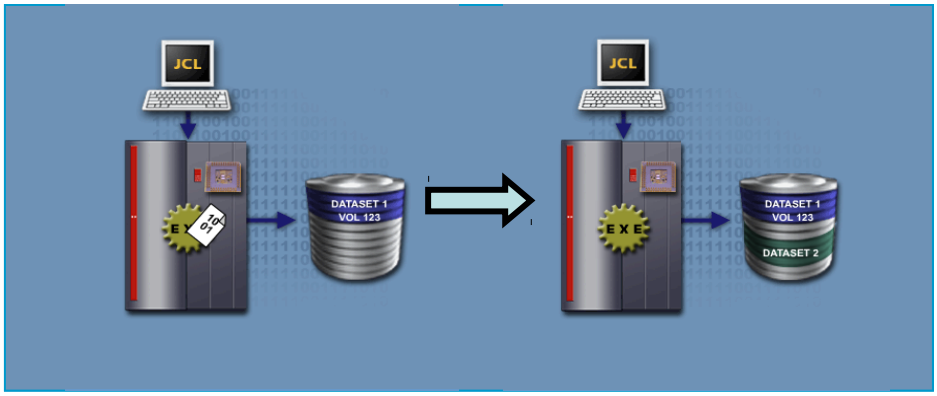
The VOL backward reference is useful when you want to create a new data set on the same volume on which an existing data set resides, but you do not know the volume identification.

23

The VOL parameter specifies the media volume on which a data set resides.

Using backward reference.

Syntax for VOL backward reference.



The general form of the VOL backward reference is shown below:

```
//ddname    DD    ...VOL=REF=dsname
```

or

```
//ddname    DD    ...VOL=REF=*.stepname.procstepname.ddname
```


Code the dsname of the existing data set in the REF subparameter of the VOL parameter for the new data set. The existing data set must be passed or cataloged.

VOL=REF=dsname

VOL=REF=*.ddname

VOL=REF=*.stepname.ddname

VOL=REF=*.stepname.procstepname.ddname

Tells the system to obtain volume serial numbers from another data set or an earlier DD statement. Notice that the * character is not mandatory. See JCL Reference.

VOL=REF obtains ONLY the volume serial numbers from the referenced data set or earlier DD statement. In particular it does not obtain the volume sequence number, volume count, label type, or data set sequence number.

Using backward reference.

VOL backward reference – example 1.

Consider an example where PROGA creates and catalogs a data set named XYZ. XYZ is to reside on the same volume as an existing, previously catalogued data set named ABC.

To refer the system to data set ABC, a VOL backward reference can be coded as follows:

```
//STEP1   EXEC  PGM=PROGA
//DD1      DD   DSN=XYZ,
//             DISP=(NEW,CATLG),
//             VOL=REF=ABC
```

Using backward reference.

VOL backward reference – example 2.

In this example the backward reference refers to a specific volume serial number coded on a prior DD statement.

The data set XYZ will be created on the volume referred to by the DD statement DD2 (volume 123456).

Using backward reference.

Are we on track?

Code a VOL backward reference when:

data set XXX will reside on the same volume as data set YYY.

27

The correct answer is //…VOL=REF=YYY

Using backward reference.

Are we on track?

Code a VOL backward reference when:

data set XXX will be created on the volume identified in the DD statement with ddname DD1.

28

The correct answer is //…VOL=REF=*.DD1

Using backward reference.

Are we on track?

Code a VOL backward reference when:

data set XXX will be created on the volume identified in STEPC as DD2.

29

The correct answer is //…DSN=XXX,VOL=REF=*.STEPC.DD2

Using backward reference.

Are we on track?

Match the underlined statements in the code with the definitions in the column on the right.

1. VOL=REF=LMN	A. stepname.ddname
2. VOL=REF=*.DD1	B. dsname
3. VOL=REF=*.STEP1.DD1	C. ddname

30

The correct answer is 1 – B, 2 – C, and 3 – A.

Using backward reference.

DCB backward reference.

What is DCB Backward Reference?

DCB backward reference is a coding technique that allows you to copy a list of attributes from a prior DD statement in the same or previous job step.

How does this technique help?

This coding technique can be used to ensure that the DCB parameters are consistent within the job.

It can also be used to override or add to the subparameters coded on a previous statement.

31

DCB parameters define the characteristics of a particular data set, such as RECFM (record format) or BLKSIZE (block size).

When DCB backward reference is used for overriding a previous statement, the values of the DCB parameter being referenced will be overridden by the values that you code. Any attributes that do not match the DCB being referenced will be added.

Using backward reference.

Syntax for DCB backward reference.

The general form is as follows:

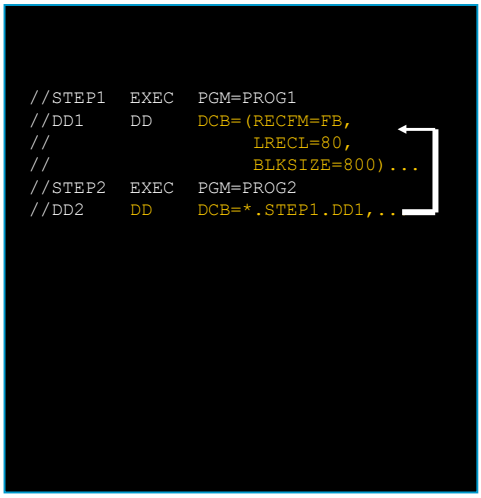
```
//ddname DD
  DCB=*.stepname.ddname
```

Using backward reference.

DCB backward reference – an example.

Assume that in STEP2 you want to create a data set with the same parameters as a data set in STEP1.

The code shown ensures that the attributes on the DD2 statement are the same as those on the DD1 statement.



Using backward reference.

DCB backward reference - overriding.

A DCB backward reference can also be used to override or add to the subparameters coded on a previous statement. The format for overriding a previous statement is as follows:

```
DCB=(*.stepname.ddname,list-of
attributes)
```

The values of the DCB parameters being referred will be overridden by the values that are being coded. Any attributes that do not match the DCB being referred will be added.

Continued…

Using backward reference.

DCB backward reference - overriding.

For example, notice the DCB characteristics in statement DD1 below:

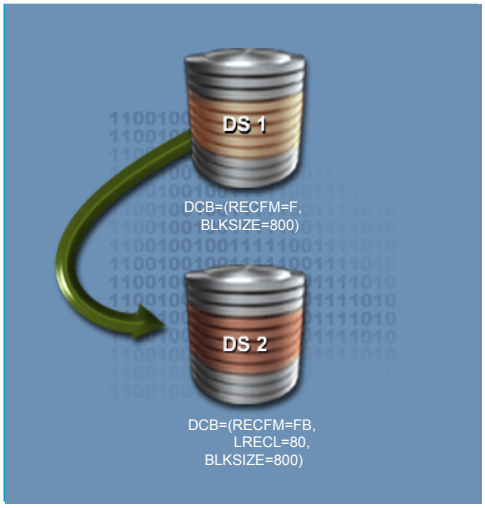
```
//STEP3    EXEC PGM=PROG3
//DD1      DD   DCB=(RECFM=F,
//              BLKSIZE=800),…
```

The following override statement:

```
//DD2      DD   DCB=(*.DD1,
//              RECFM=FB,LRECL=80)
```

would result in these DCB characteristics:

```
//DD2      DD   DCB=(RECFM=FB,
//               LRECL=80,BLKSIZE=800)
```



Using backward reference.

Are we on track?

The portion of the job stream shown below contains JCL statements, some of which are incomplete.

```
1. //COMPILE   EXEC   PGM=PL1
2. //COMPOUT   DD     UNIT=SYSDA,VOL=SER=PACK12,
   //                 DISP=(NEW,PASS),DSN=&&A
3. //LKED      EXEC   PGM=LINKEDIT
4. //LKEDIN    DD     DISP=OLD,DSN=__________
5. //SYSLMOD   DD     DISP=(NEW,PASS),DSN=&&GOSET(GO),
   //                 VOL=_______________
6. //GO        EXEC   PGM=___________
7. //MYDATA    DD     DSN=MYDATA,DISP=(NEW,CATLG),
   //                 VOL=SER=…,SPACE=(800,50),
   //                 DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
8. //TEMP      DD     UNIT=SYSDA,DCB=__________
```

Complete those statements by coding the appropriate backward references as follows:

The data set name in statement 4 refers to statement 2.

The correct answer is *.COMPILE.COMPOUT

The general form for the DSN backward reference is as follows (see slide 19):

`DSN=*.stepname.ddname`

Using backward reference.

Are we on track?

The portion of the job stream shown above contains JCL statements, some of which are incomplete.

```
1. //COMPILE  EXEC  PGM=PL1
2. //COMPOUT  DD    UNIT=SYSDA,VOL=SER=PACK12,
   //                DISP=(NEW,PASS),DSN=&&A
3. //LKED     EXEC  PGM=LINKEDIT
4. //LKEDIN   DD    DISP=OLD,DSN=*.COMPILE.COMPOUT
5. //SYSLMOD  DD    DISP=(NEW,PASS),DSN=&&GOSET(GO),
   //                VOL=_____________
6. //GO       EXEC  PGM=___________
7. //MYDATA   DD    DSN=MYDATA,DISP=(NEW,CATLG),
   //                VOL=SER=…,SPACE=(800,50),
   //                DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
8. //TEMP     DD    UNIT=SYSDA,DCB=__________
```

Complete those statements by coding the appropriate backward references as follows:

37

The volume in statement 5 refers to statement 2.

The correct answer is REF=*.COMPILE.COMPOUT

The general form of the VOL backward reference is shown below (see slide 24):

//ddname DD …VOL=REF=dsname

or

//ddname DD …VOL=REF=*.stepname.procstepname.ddname

Using backward reference.

Are we on track?

The portion of the job stream shown above contains JCL statements, some of which are incomplete.

```
1. //COMPILE  EXEC  PGM=PL1
2. //COMPOUT  DD    UNIT=SYSDA,VOL=SER=PACK12,
   //                DISP=(NEW,PASS),DSN=&&A
3. //LKED     EXEC  PGM=LINKEDIT
4. //LKEDIN   DD    DISP=OLD,DSN=*.COMPILE.COMPOUT
5. //SYSLMOD  DD    DISP=(NEW,PASS),DSN=&&GOSET(GO),
   //                VOL=REF=*.COMPILE.COMPOUT
6. //GO       EXEC  PGM=___________
7. //MYDATA   DD    DSN=MYDATA,DISP=(NEW,CATLG),
   //                VOL=SER=…,SPACE=(800,50),
   //                DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
8. //TEMP     DD    UNIT=SYSDA,DCB=__________
```

Complete those statements by coding the appropriate backward references as follows:

The program in statement 6 refers to statement 5.

28

The correct answer is *.LKED.SYSLMOD

The general form of a PGM backward reference is as follows (see slide 13):

```
//STEP  EXEC PGM=*.stepname.ddname
```

Using backward reference.

Are we on track?

The portion of the job stream shown above contains JCL statements, some of which are incomplete.

```
1. //COMPILE   EXEC   PGM=PL1
2. //COMPOUT   DD     UNIT=SYSDA,VOL=SER=PACK12,
   //                 DISP=(NEW,PASS),DSN=&&A
3. //LKED      EXEC   PGM=LINKEDIT
4. //LKEDIN    DD     DISP=OLD,DSN=*.COMPILE.COMPOUT
5. //SYSLMOD   DD     DISP=(NEW,PASS),DSN=&&GOSET(GO),
   //                 VOL=REF=*.COMPILE.COMPOUT
6. //GO        EXEC   PGM=*.LKED.SYSLMOD
7. //MYDATA    DD     DSN=MYDATA,DISP=(NEW,CATLG),
   //                 VOL=SER=…,SPACE=(800,50),
   //                 DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
8. //TEMP      DD     UNIT=SYSDA,DCB=__________
```

Complete those statements by coding the appropriate backward references as follows:

39

The DCB attributes in statement 8 refer to statement 7.

The correct answer is *.GO.MYDATA

The general form is as follows (see slide 32):

`//ddname DD DCB=*.stepname.ddname`

Using backward reference.

Glossary.

DD Statement
A JCL statement that describes each data set used within a job.

DDname
A unique name given to each data set used in a job step.

Job Step
The JCL statements that control the execution of a program and request the resources needed to run the program. A job step is identified by an EXEC statement.

Parameter Values
Information that follows a keyword parameter and an equal sign.

PGM
An EXEC statement parameter that names the program to execute.

DSN
A DD statement parameter that names the data set.

Using backward reference.

Glossary.

VOL
A parameter on a DD statement that requests a specific volume.

DCB
Data Control Block. A parameter on a DD statement that describes the attributes of a data set, such as block size and record format.

Load Module
An executable program that results from a link edit step.

SYSLMOD
DD name used by the linkage editor to write its output (a load module).

DISP
Describes the status of a data set to the system and tells the system what to do with the data set after termination of the step or job.

Concatenating data sets.

Data set concatenation – definition.

What is data set concatenation?

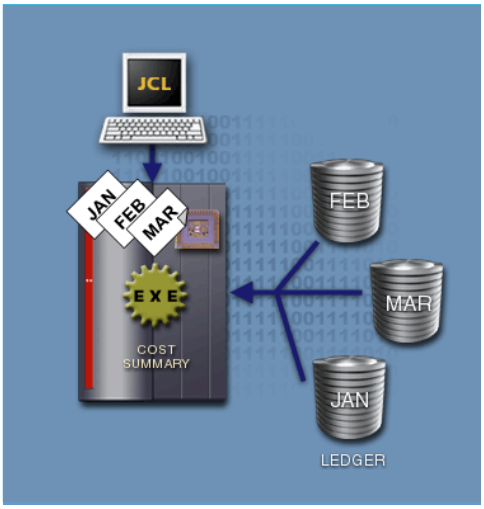
A programmer can code DD statements to request that several data sets be concatenated.

Data set concatenation enables the system to process several separate physical data sets as one logical data set.

Concatenating data sets.

Data set concatenation – an example.

- Consider a cost ledger to produce a monthly cost summary file. At the year end, it is required to process all 12 monthly data sets to produce an annual report. All the data sets are concatenated so they can be processed sequentially.
- In this example, the program uses a ddname of LEDGER and the monthly data sets are named JAN, FEB, MAR and so on.
- The operating system draws the concatenated data sets sequentially, treating them as a single logical data set.



Concatenating data sets.

Concatenation of data sets.

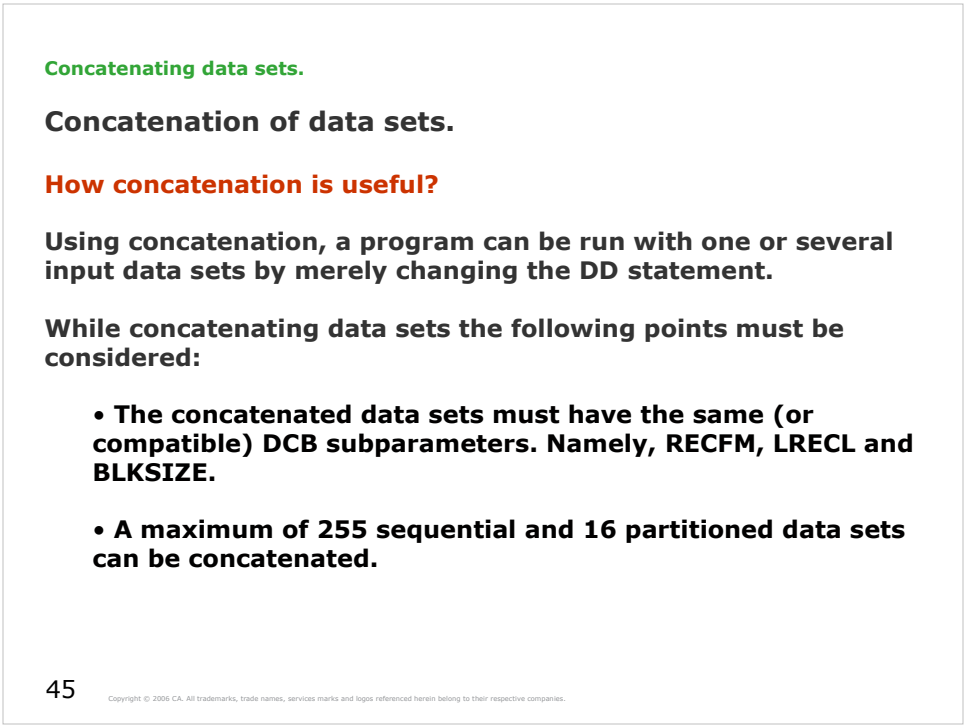
How to concatenate data sets?

Following steps are involved in concatenating data sets:

1. Code a standard DD statement for the first data set only.

2. Add a DD statement without a ddname for each data set to be concatenated.

3. Sequence the statements in the order they are to be processed.



If the concatenated data sets do not have the same block size, the data set with the largest block size should be in the first DD statement.

Can I concatenate an output data set?

Yes, I can. The concatenated output data set will be allocated but a program does not write anything to the data set.

Concatenating data sets.

JCL for data set concatenation – an example.

The JCL here shows the concatenation of the monthly data sets considered in the LEDGER example.

The last data set concatenated to LEDGER is DEC.

The occurrence of the ddname SUM indicates that the data set (ACCT.1999) is to be processed separately from the LEDGER data sets.

```
//LEDGER DD DSN=JAN,DISP=SHR
//       DD DSN=FEB,DISP=SHR
           .
           .
           .
//       DD DSN=DEC,DISP=SHR
//SUM    DD DSN=ACCT.1999,DISP=SHR
```

Concatenating data sets.

Are we on track?

Consider three data sets named, CUST.HISTORY.JUL, CUST.HISTORY.APR and CUST.HISTORY.JAN which are to be processed in this order. They are to be concatenated to CUST.HISTORY.OCT, to create a master customer list.

Put the following statements in order.

A.
```
//          DD DSN=CUST.HISTORY.APR
```

B.
```
//MASTCUST DD DSN=CUST.HISTORY.OCT
```

C.
```
//          DD DSN=CUST.HISTORY.JAN
```

D.
```
//          DD DSN=CUST.HISTORY.JUL
```


The correct order is B., D., A., and C.

Glossary.

Concatenated data sets
Data sets that are separate physically, but processed sequentially as one logical data set.

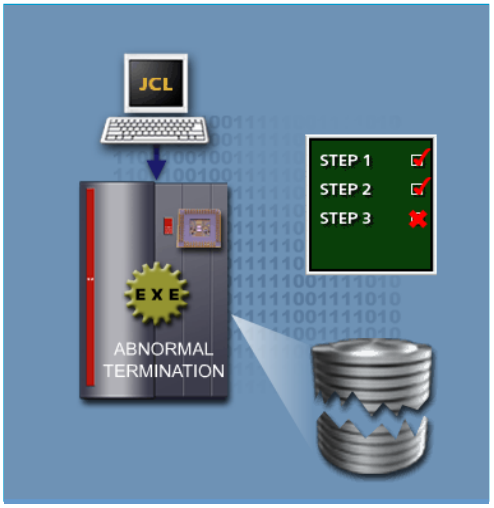
Dummy data sets.

Overview.

Each data set that is referred by a program should have a ddname. The JCL for the program must contain the corresponding DD statements.

If a data set is not coded by a DD statement, then the program will abnormally end (ABEND) as shown.

When an input data set is optional for the program's processing or when an output data set is not required dummy data sets can be used.



Dummy data set.

What is a dummy data set?

A dummy data set is a data set for which all Input or Output (I/O) operations are bypassed.

A special DD statement, DD DUMMY, is used to ignore a data set during the execution of a program.

How does it work?

When a data set is assigned dummy status, all I/O operations are bypassed and device allocation, space allocation and data set disposition are ignored.

Dummy data sets.

Specifying dummy data sets.

Dummy data sets can be specified in DD statements by doing one of the following:

- **Coding DUMMY as the first DD parameter**

syntax:

```
//DDname        DD      DUMMY
```

- **Coding DSN=NULLFILE**

syntax:

```
//DDname        DD      DSN=NULLFILE
```

51

While coding DUMMY as the first parameter, DUMMY acts as a positional parameter. When the DUMMY or NULLFILE parameter is coded, all other parameters on the DD statement are ignored except for DCB information.

NULLFILE is a reserved word and a data set cannot be named as NULLFILE.

Dummy data sets.

Dummy data sets – an example.

Consider a payroll program named PAY that processes separate input data sets. The ddname TIMECDS refers to weekly time cards and the ddname ADJUST refers to adjustments to previous pay period information.

The job stream must include:

```
//STEPA     EXEC  PGM=PAY
//TIMECDS   DD    ---
//ADJUST    DD    ---
                  .
                  .
```

Dummy data sets.

Dummy data sets – an example.

Even if there are no adjustments for PAY process, DD statement for ADJUST must be included.

To tell the system that there is no ADJUST data set code can be written as follows:

```
//STEPA      EXEC  PGM=PAY
//TIMECDS    DD    ----
//ADJUST     DD    DUMMY
```

If the data set described by the DD statement named ADJUST is referred to by the PAY program, an immediate end-of-file occurs. The program will continue as if it has processed the entire data set.

53

If the program issues a READ from a dummy data set, end-of-file condition occurs. If WRITE to a dummy data set is issued, nothing will be written.

Dummy data sets.

Are we on track?

You can specify a dummy data set by coding DSN=_____ on the DD statement.

54

The correct answer is NULLFILE.

Storage dumps.

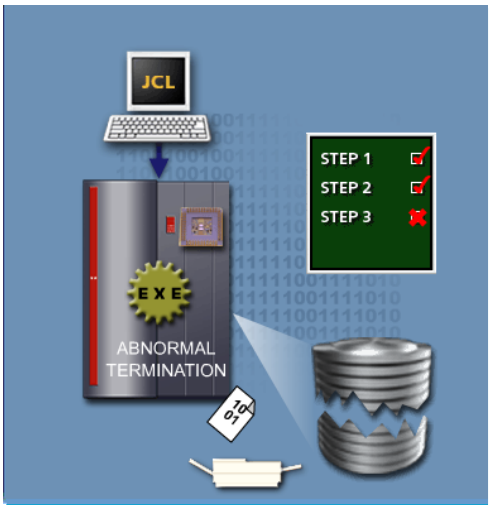
Storage dumps.

What are Storage Dumps?

When a program abnormally terminates, storage dumps are used as a debugging tool to find clues to the cause for abnormal ending. Storage dumps are not the most effective debugging tool.

The main drawbacks of storage dumps are:

- They are difficult to read since they are in hexadecimal code.
- Printing storage dumps is time consuming.



55

When a program abnormally terminates, the user can often find clues to the reason for the ABEND in the contents of the computer's storage.

Special DDnames.

These reserved ddnames request storage dumps in the event that a program terminates abnormally:

SYSUDUMP: **Requests a formatted dump of the processing program area. It is most generally used for debugging problem programs.**

SYSABEND: **Requests a formatted dump of the processing program area, system programs and the system control blocks. It is often spooled for printing, although it may be written onto any output device.**

SYSMDUMP: **Requests an unformatted dump of the processing program area and the system nucleus in machine readable form. It is generally directed to tape (or to direct access storage) to allow subsequent processing by a dump analysis utility.**

Storage dumps.

Handling storage dumps.

It is necessary to plan ahead for a possible storage dump.

To obtain a dump, the SYSUDUMP, SYSABEND, or SYSMDUMP DD statements must be coded in the JCL for each job step from which a dump needs to be obtained.

The example shown uses SYSUDUMP DD statement.

If STEP1 or STEP2 terminates abnormally, the system creates a dump of the program storage area.

```
//STEP1    EXEC  PGM=PROG1
//SYSDUMP  DD    SYSOUT=X
//DD1      DD    ...
//STEP2    EXEC  PGM=PROG2
//SYSUDUMP DD    SYSOUT=X
```


Notice in the example that a SYSUDUMP DD statement must be included for each step of the job in order to obtain the storage dump for the step.

Storage dumps.

Are we on track?

Match the special ddname with its function

1. SYSABEND	**A. Requests an unformatted dump in machine-readable form of the processing program area and the system nucleus.**
2. SYSMDUMP	**B. Requests a formatted dump of the processing program area and of the system control blocks.**
3. SYSUDUMP	**C. Requests a formatted dump of the processing program area.**

58

The correct answer is 1- B, 2 – A, and 3 – C.

Unit summary.

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

- **Code a DD statement to use information from preceding JCL statements.**
- **Identify the purpose of data set concatenation.**
- **Code the JCL to concatenate a data set.**
- **Code a DD statement to indicate that a data set is to be ignored for the current program execution.**
- **Identify the purpose of special ddnames.**