z/OS Basic Skills Information Center

# Reusable JCL collection

z/OS Basic Skills Information Center

# Reusable JCL collection

**Note**

Before using this information and the product it supports, read the information in “Notices” on page 59.

This edition applies to z/OS (product number 5694-A01).

We appreciate your comments about this publication. Comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Send your comments through this Web site: http://publib.boulder.ibm.com/infocenter/zoslnctr/v1r7/index.jsp?topic=/com.ibm.zcontact.doc/webqs.html

# Contents

# Introducing the reusable JCL collection

The reusable JCL collection consists of working samples of job control language (JCL) that new z/OS® professionals can copy, edit, and reuse to accomplish basic tasks on the job.

This collection is designed to help new users quickly become productive in the z/OS environment, while teaching JCL keywords and syntax within the context of realistic samples. The samples in the collection are based on tasks that existing z/OS customers have identified as most likely to be given to new z/OS professionals as part of their on-the-job training.

The samples in the reusable JCL collection are as complete as possible, and include line-by-line instructions and descriptions. Correctly modifying the samples for use, however, requires some basic knowledge of JCL techniques and of your own work environment. This collection includes a company checklist and other educational materials that can help you use the samples, and eventually code your own JCL.

# Chapter 1. Preparing to use JCL samples in this collection

The reusable JCL collection consists of working samples of JCL that you can copy, edit, and reuse to quickly accomplish basic tasks on the job. Using these samples requires some basic knowledge of JCL techniques and of your own work environment; start with this checklist.

## About this task

Although the samples in the reusable JCL collection are as complete as possible, and include line-by-line instructions and descriptions:

- You need to understand the basic process for creating or editing JCL and submitting a job.
- You need to memorize certain JCL techniques that are required for almost all of the JCL that you modify or write.
- You must replace certain variables in the samples with company-specific information.
- You will need to use additional reference materials if you want to significantly alter these reusable samples.

Before you begin to work with the JCL samples, use the following checklist to collect or learn what you need to know.

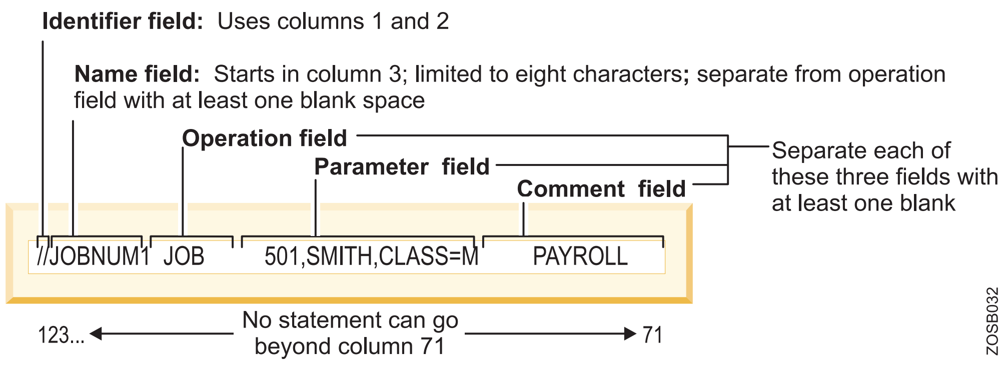
## Procedure

1. Complete or review the exercise described in “JCL exercise: Creating and submitting a job” on page 47, which takes you through the process of creating a data set member for JCL, coding JCL (using a predefined sample), submitting the job, and viewing the job output. Unless your mentor has already set up a JCL data set for you, you will need to create your own data set to contain any JCL you write or use, including any of the reusable samples in this collection.
2. Make sure you have access to the latest editions of the following resources:
   - *z/OS MVS™ JCL Guide* (SA22-7598) and *z/OS MVS JCL Reference* (SA22-7597). The latter is an especially indispensable resource, containing detailed descriptions of JCL parameters and keywords.
   - *z/OS MVS System Messages* and *z/OS MVS System Codes* (SA22-7626). Both are useful for debugging error messages or return codes.

   All are available in several formats. Check for them online in the z/OS Internet Library:

   http://www.ibm.com/servers/eserver/zseries/zos/bkserv/

   You might need additional resources from the z/OS product library as well; any specific references are noted in the instructions for each sample.
3. Memorize the names and functions of the five fields that constitute a JCL statement.


Identifier field: Uses columns 1 and 2
Name field: Starts in column 3; limited to eight characters; separate from operation field with at least one blank space
Operation field
Parameter field
Comment field
Separate each of these three fields with at least one blank
//JOBNUM1 JOB 501,SMITH,CLASS=M PAYROLL
123...
No statement can go beyond column 71
71
ZOSB032

4. Memorize the syntax rules for continuing JCL statements on more than one line. When you modify one of the reusable JCL samples, your changes often will make a complete JCL statement exceed 71 characters in length. When this happens, you must use multiple lines to complete the statement. To continue a JCL statement on one or more separate lines:
   a. In the parameter field of the JCL statement, end the line before reaching position 72, but after coding a complete parameter or subparameter and the comma that follows it.
   b. On the next line, code two forward slashes (//) in positions 1 and 2.
   c. Beginning in any position from 4 to 16, resume coding the JCL statement with the next complete parameter or subparameter.

   You also may use this technique to make your JCL more easily readable. This example illustrates a correctly coded multiple-line JCL statement:
   ```
   //NEWDS DD  DSNAME=ZUSER03.ACTDATA.LIST,DISP=(NEW,KEEP),
   //          DATACLAS=DSCLAS01,STORCLAS=STRCLS20
   ```
5. Learn how to correctly code data set names on the **DSNAME** parameter (often abbreviated **DSN**) on data definition (DD) statements. Naming and syntax rules for coding data set names vary depending on the type of data set you are identifying.

   Table 1 on page 3 lists the different types and examples of correctly coded names. Unless another resource is noted in the table, *z/OS MVS JCL Reference* (SA22-7597) is the definitive source for complete details about the data set types and permissible names, along with syntax rules.

Table 1. Summary of data set types and correctly coded DSNAME (DSN) parameter values

| Type of data set | DSNAME (DSN) parameter value formats and examples |
|---|---|
| **Permanent** | **Unqualified names:** One through 8 alphanumeric or special ($, #, @) characters, a hyphen, or a character X'C0'. The first character must be alphabetic or special ($, #, @).<br><br>**Example of an unqualified name:**<br>`DSNAME=ALPHA` |
| | **Qualified names:** Multiple unqualified names joined by periods. Each qualifier is coded like an unqualified name; therefore, the name must contain a period after every 8 characters or fewer. The maximum length of a qualified data set name is:<br>• 44 characters, including periods.<br>• For a generation data group, 35 characters, including periods.<br>• For an output tape data set, 17 characters, including periods.<br><br>**Example of a qualified name:**<br>`DSNAME=ALPHA.PGM` |
| | **RACF-protected data sets:** Use the same format as for a qualified name, and make sure the high-level qualifier of the name is defined to RACF®. Further details are documented in *z/OS Security Server RACF Security Administrator's Guide* (SA22-7683). |
| | **Formats for names of cataloged data sets:**<br>`dsname`<br>`dsname(member)`<br>`dsname(gen_data_group)`<br>`dsname(INDEX \| PRIME \| OVFLOW)`<br><br>**Example for a cataloged data set:**<br>`DSNAME=LIB1(PROG12)`<br><br>Further details are documented in *z/OS DFSMS™ Access Method Services for Catalogs* (SC26-7394). |
| **Temporary** | When you define a temporary data set, you can code the **DSNAME** parameter or omit it; in either case, the system generates a qualified name for the temporary data set.<br><br>When you use the **DSNAME** parameter for a temporary data set, code the name as two ampersands (&&) followed by a character string 1 to 8 characters in length:<br>• The first character following the ampersands must be alphabetic or special ($, #, @).<br>• The remaining characters must be alphanumeric or special ($, #, @).<br><br>**Formats for temporary data set names:**<br>`&&dsname`<br>`&&dsname(member)`<br>`&&dsname(INDEX \| PRIME \| OVFLOW)`<br><br>**Example for a temporary data set:**<br>`//DD3 DD DSNAME=&&WORK,UNIT=3420` |

*Table 1. Summary of data set types and correctly coded DSNAME (DSN) parameter values (continued)*

| Type of data set | DSNAME (DSN) parameter value formats and examples |
|---|---|
| **In-stream or system output (sysout)** | When defining an in-stream or sysout data set, you can code the **DSNAME** parameter or omit it; if omitted, the system generates a name for the data set.<br><br>The data set name for in-stream and sysout data sets consists of two ampersands (&&) followed by one through eight 8 alphanumeric or special ($, #, @) characters, a hyphen, or a character X'C0'. The first character following the ampersands must be alphabetic or special ($, #, @).**Example for an in-stream data set:**<br>`//DDIN DD DATA,DSNAME=&&PAYIN1`<br><br>**Example for a sysout data set:**<br>`//DDOUT DD DSNAME=&&PAYOUT1,SYSOUT=P` |
| **Backward reference** | A *backward reference* is a reference to an earlier statement in the job or in a cataloged or in-stream procedure called by this or an earlier job step. A backward reference can be coded in the **DSNAME** parameter to copy a data set name from an earlier DD statement.<br><br>**Formats for backward references:**<br>`*.ddname`<br>`*.stepname.ddname`<br>`*.stepname.procstepname.ddname`<br><br>**Example of a backward reference in DD5 statement in STEP2:**<br>`//STEP1 EXEC PGM=CREATE`<br>`//DD4   DD DSNAME=&&ISDATA(PRIME),DISP=(,PASS),`<br>`//         UNIT=(3350,2),VOLUME=SER=334859,`<br>`//         SPACE=(CYL,(10,,2),,CONTIG),DCB=DSORG=IS`<br>`//STEP2 EXEC PGM=OPER`<br>`//DD5   DD DSNAME=*.STEP1.DD4,DISP=(OLD,DELETE)` |
| **Dummy data set** | The parameter **NULLFILE** specifies a dummy data set. **NULLFILE** has the same effect as coding the DD **DUMMY** parameter. |

6. Ask your system programmer or mentor to help you complete the following list, which identifies elements of your work environment that might affect the JCL that you code. Use Table 2 on page 5 to record your answers. When this worksheet is complete, you should have the company-specific information you need for most of the jobs you will run on z/OS.
   - Determine which job entry subsystem (JES2 or JES3) is installed on the z/OS system you will use. For many jobs, the type of JES does not affect JCL parameters; for certain jobs, however, the JES in use does dictate which JCL parameters, values, or job entry control (JECL) statements you may code.
   - Determine which access methods your company uses for its data sets. An access method defines the technique that is used to store and retrieve data. Access methods have their own data set structures to organize data, system-provided programs (or macros) to define data sets, and utility programs to process data sets. Access methods, therefore, determine which JCL parameters and parameter values that you need to code.
   - For direct-access storage devices (DASD), determine which naming conventions are used, as well as default or recommended values for data set attributes.
   - For storing or backing up data on tape, determine which tape device volume numbers and types are available for your use.

- Determine whether your company uses the Storage Management Subsystem (SMS) to automate the use of storage for data sets. The JCL parameters for SMS-managed (also called system-managed) data sets are different from some parameters used for non-SMS data sets.
- Determine the information (account number, programmer name, and so on) your company requires for each job that you submit.

*Table 2. JCL worksheet*

| Company convention or z/OS environment specifics | | Notes® / Values to code on JCL statements |
|---|---|---|
| **Job entry subsystem** JES2 or JES3 | | |
| **Access methods** Queued Sequential (QSAM) Basic Partitioned (BPAM) Virtual Sequential (VSAM) Basic Sequential (BSAM) Basic Direct (BDAM) | | |
| **Direct-access storage devices (DASD)** | | DSN= UNIT= VOL=SER= |
| **Magnetic tape devices** | | LABEL= UNIT= VOL=SER= |
| **Data management system** | | |
| Conventions for SMS-managed data sets | | |
| | Average record | AVGREC= |
| | Data classes | DATACLAS= |
| | Management classes | MGMTCLAS= |
| | RACF profile names | SECMODEL= |
| | Storage classes | STORCLAS= |
| Conventions for non-SMS-managed data sets | | |
| | Data set attributes or requirements | BLKSIZE |
| | | LRECL= |
| | | RECFM |
| | | SPACE= |
| | | SYSOUT= |
| **Conventions for the JOB statement** | | |
| | Account number | |
| | Other accounting information | |
| | Programmer name | |
| | Class | CLASS= |

*Table 2. JCL worksheet (continued)*

| **Company convention or z/OS environment specifics** | | **Notes® / Values to code on JCL statements** |
|---|---|---|
| | Message class | MSGCLASS= |
| | Message level | MSGLEVEL= |
| | Region size | REGION= |
| | Time limit | TIME= |

# Chapter 2. Selecting a reusable JCL sample

These JCL samples constitute the starter set for the reusable JCL collection.

## Reusable JCL: Creating a data set

Modify this JCL sample to create a new data set, using the IBM® program IEFBR14.

### Before you begin

- If you have not already done so, allocate a data set to contain your modified version of this JCL sample. Use the instructions in "JCL exercise: Creating and submitting a job" on page 47 to create this data set.
- Determine the information (account number, programmer name, and so on) your company requires for each job that you submit.
- Determine whether your company uses the Storage Management Subsystem (SMS) to automate the use of storage for data sets. The JCL parameters for SMS-managed (also called system-managed) data sets are different from some parameters used for non-SMS data sets.
- For direct-access storage devices (DASD), determine which naming conventions are used, as well as default or recommended values for data set attributes.

### About this task

The steps that follow provide line-by-line instructions for modifying this JCL sample:

```
//jobname  JOB  (start of JOB statement parameters)
//stepname EXEC PGM=IEFBR14
//ddname   DD   DSN=dsname,
//     DISP=(NEW,CATLG),
//     UNIT=SYSALLDA,SPACE=(TRK,1)
/*
```

In the JCL statements that appear in code examples, lowercase text indicates items that you need to modify. Except for a few cases, lowercase alphabetic characters cannot be used in JCL.

All jobs require JOB and EXEC statements, so this sample contains both:

- The JOB statement marks the beginning of a job, specifies the job name, and also might provide company-specific details or JCL parameters that apply to all job steps within the job.
- The EXEC statement marks the beginning of a job step. In this case, the job step is to run IEFBR14, which is a program that simply passes control back to z/OS. For steps that call IEFBR14, then, other JCL statements within the step specify any work that z/OS does.

If you modify this sample to complete more complex tasks, or if you encounter JCL errors, see *z/OS MVS JCL Reference* (SA22-7597), which is the comprehensive source of syntax rules and statement parameter descriptions.

Other useful references include:

- *z/OS MVS JCL Guide* (SA22-7598), which contains instructions and examples for using IEFBR14 to test your JCL.

- *z/OS DFSMS Using Data Sets* (SC26-7410), which contains instructions for using IEFBR14 to create different types of data sets, including HFS files.

## Procedure

1. Required: Modify the JOB statement to uniquely identify your job and to provide additional company-specific information.

   ```
   //jobname  JOB  (start of JOB statement parameters)
   ```

   a. Replace `jobname` with a unique name to identify this job. A common convention is to use your TSO logon ID followed by a number (for example: `ZUSER031`).

      Syntax rules for the name field are:
      - The name must begin in column 3 of the JCL statement.
      - The name can be one through eight characters in length.
      - The first character in the name must be an alphabetic character (the letters A through Z) or a special character (the symbols #, @, and $).
      - The remaining characters can be alphanumeric (the letters A through Z and numbers 0 through 9) or special characters.
      - Blank spaces cannot be included in a name.

   b. Replace `(start of JOB statement parameters)` with parameters and values that conform to guidelines set at your company.

2. Required: Modify the EXEC statement to uniquely identify the job step and the utility to be run.

   ```
   //stepname EXEC PGM=IEFBR14
   ```

   a. Replace `stepname` with a unique name to identify this step. Syntax rules for `stepname` are identical to those listed for `jobname`. Aside from changing the step name, no further changes are required.

3. Required: Modify this input DD statement to define the data set to be created.

   ```
   //ddname   DD   DSN=dsname,
   //      DISP=(NEW,CATLG),
   //      UNIT=SYSALLDA,SPACE=(TRK,1)
   ```

   a. Replace `ddname` with a unique name for this JCL DD statement; this label in the name field of a DD statement is known as a ddname. Syntax rules for ddnames are identical to those listed for job and step names on JOB and EXEC statements.

   b. Replace `dsname` with the name that you want to use for your new data set. Naming and syntax rules for **DSN** parameter values vary depending on the type of data set you are identifying. If you want to create a permanent data set to store your own collection of JCL samples, you can specify a qualified name that starts with your TSO logon ID; for example: `DSN=ZUSER03.JCL`

   c. Change the value for the **DISP** parameter, if necessary. The **DISP** parameter tells the system about the status of your data set and what to do with it when your job ends, either normally or abnormally. As coded in this sample, the status subparameter value `NEW` tells the system to create your data set. Only one subparameter value for job-end processing is specified, so the system will add an entry in the system or user catalog (`CATLG`), whether the job step ends normally or abnormally.

      If you want to understand more about disposition processing, refer to the summary of disposition processing in *z/OS MVS JCL Reference* (SA22-7597), in the DD statement topic for the **DISP** parameter.

   d. Change or replace the **UNIT** parameter, if necessary. Coding `UNIT=SYSALLDA` tells z/OS to find the most appropriate storage unit for your new data set. You may use this parameter and value to define either an SMS-managed or non-SMS-managed data set. If you are using SMS, however, you may replace the **UNIT** parameter with the **STORCLAS** parameter and a storage class name that your company uses.

e. Change or replace the **SPACE** parameter, if necessary. Coding `SPACE=(TRK,1)` tells z/OS to do one of the following:
   - For a non-SMS-managed data set, allocate one track of space for your new data set.
   - For an SMS-managed data set, override the space attributes specified through the default data class. If you are defining an SMS-managed data set, you have these choices:
     – Leave the **SPACE** parameter as shown in the sample.
     – Replace the **SPACE** parameter value to specify different space attributes.
     – Replace the **SPACE** parameter and its value with a DATACLAS parameter and data class name that your company uses.
     – Remove the **SPACE** parameter and its value to accept the space attributes defined in the default data class.

4. Optional: Check for JCL syntax errors by submitting the job with `TYPRUN=SCAN` on the JOB statement.

```
//jobname  JOB  (start of JOB statement parameters),TYPRUN=SCAN
```

   Using `TYPRUN=SCAN` does not catch all possible JCL errors, but it's a good start to ensuring that your job will run. `TYPRUN=SCAN` requests that the system scan this job's JCL for syntax errors, without executing the job or allocating devices. This parameter asks the system to check for:
   - Spelling of parameters and some subparameters that is not correct.
   - Characters that are not correct.
   - Unbalanced parentheses.
   - Misplaced positional parameters on some statements.
   - In a JES3 system only, parameter value errors or excessive parameters.
   - Incorrect syntax on JCL statements in cataloged procedures invoked by any scanned EXEC statements.

   You might still encounter JCL errors after using TYPRUN=SCAN, because this request checks the JCL only through the converter, not the interpreter. The difference is that the converter basically checks all expressions to the **left** of an equal sign plus **some** expressions to the right of an equal sign (and issues messages that start with IEFC), while the interpreter checks all expressions to the **right** of an equal sign (and issues messages that start with IEF). For example, a data set name containing a qualifier that exceeds eight characters, such as `DSN=L9755TB.JCL.TEST19970103` would not be flagged by TYPRUN=SCAN but would be caught by the interpreter.

5. Required: Remove `TYPRUN=SCAN` from the JOB statement and submit the job. The system response is:

```
JOB jobname(jobnumber) SUBMITTED
***
```

## Results

When the job ends, you will receive a message indicating one of three conditions: job successful, JCL error, or program abend. Use your installation's viewing facility (for example, SDSF) to view the output and determine whether the job completed successfully.

## Reusable JCL: Copying a data set to tape

Modify this JCL sample to copy one cataloged data set to a tape device, using the DFSMSdfp™ utility IEBGENER.

### Before you begin

- If you have not already done so, allocate a data set to contain your modified version of this JCL sample. Use the instructions in "JCL exercise: Creating and submitting a job" on page 47 to create this data set.
- Determine the information (account number, programmer name, and so on) your company requires for each job that you submit.
- Determine whether your company uses the Storage Management Subsystem (SMS) to automate the use of storage for data sets. The JCL parameters for SMS-managed (also called system-managed) data sets are different from some parameters used for non-SMS data sets.
- For storing or backing up data on tape, determine which tape device volume numbers and types are available for your use.

### About this task

The steps that follow provide line-by-line instructions for modifying this JCL sample:

```
//jobname  JOB  (start of JOB statement parameters)
//stepname EXEC PGM=IEBGENER
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  DUMMY
//SYSUT1   DD  DSN=dsname,DISP=SHR
//SYSUT2   DD  DSN=dsname,DISP=(NEW,CATLG),
//     UNIT=tapedevice,
//     VOL=SER=volser
/*
```

In the JCL statements that appear in code examples, lowercase text indicates items that you need to modify. Except for a few cases, lowercase alphabetic characters cannot be used in JCL.

All jobs require JOB and EXEC statements, so this sample contains both:

- The JOB statement marks the beginning of a job, specifies the job name, and also might provide company-specific details or JCL parameters that apply to all job steps within the job.
- The EXEC statement marks the beginning of a job step. In this case, the job step is to run the program IEBGENER, which is a DFSMSdfp utility with a variety of uses.

When you use the IEBGENER utility to copy a data set to tape, you must define several input and output data definition (DD) statements in the job step:

- The `SYSPRINT DD` statement tells the system where to print IEBGENER messages.
- The `SYSIN DD` statement identifies a control data set that IEBGENER uses in some cases.
- The `SYSUT1 DD` statement identifies the input data set; that is, the data set that IEBGENER is to copy onto tape.
- The `SYSUT2 DD` statement identifies the output data set and its location; that is, the copied data set (which may have a different name than the original), and the tape device on which the copied data set will reside. To use a tape device, you will need to consult with your mentor to find out how to get a tape and have it

mounted before you run this job. Your mentor can help you determine how to correctly modify this output DD statement.

You might want to refer to *z/OS MVS JCL Reference* (SA22-7597), which contains other syntax rules and statement parameter descriptions that might help if you modify this sample to complete more complex tasks, or if you encounter JCL errors.

Another useful reference is *z/OS DFSMSdfp Utilities* (SC26-7414), which contains details about using IEBGENER.

## Procedure

1. Required: Modify the JOB statement to uniquely identify your job and to provide additional company-specific information.

   ```
   //jobname  JOB  (start of JOB statement parameters)
   ```

   a. Replace `jobname` with a unique name to identify this job. A common convention is to use your TSO logon ID followed by a number (for example: `ZUSER031`).

      Syntax rules for the name field are:
      - The name must begin in column 3 of the JCL statement.
      - The name can be one through eight characters in length.
      - The first character in the name must be an alphabetic character (the letters A through Z) or a special character (the symbols #, @, and $).
      - The remaining characters can be alphanumeric (the letters A through Z and numbers 0 through 9) or special characters.
      - Blank spaces cannot be included in a name.

   b. Replace `(start of JOB statement parameters)` with parameters and values that conform to guidelines set at your company.

2. Required: Modify the EXEC statement to uniquely identify the job step and the utility to be run.

   ```
   //stepname EXEC PGM=IEBGENER
   ```

   a. Replace `stepname` with a unique name to identify this step. Syntax rules for `stepname` are identical to those listed for `jobname`. Aside from changing the step name, no further changes are required.

3. Required: Include a `SYSPRINT DD` statement to tell the system where to print IEBGENER messages.

   ```
   //SYSPRINT DD  SYSOUT=*
   ```

   The `SYSPRINT DD` statement with `SYSOUT=*` tells the system to print the informational or error messages from IEBGENER in the job log. Although you may use other parameter values for `SYSPRINT`, no changes are required for this DD statement.

4. Required: Include a `SYSIN DD` statement to identify a control data set that contains additional instructions for IEBGENER.

   ```
   //SYSIN    DD  DUMMY
   ```

   When you are copying a data set to tape, IEBGENER does not need a control data set, but you must include the `SYSIN DD` statement in your JCL anyway. Using the DUMMY parameter tells the system that no resources are required for the control data set. Although you may use other parameter values for `SYSIN`, no changes are required for this DD statement.

5. Required: Modify this input DD statement to identify the cataloged data set that IEBGENER is to copy onto tape, and the data set disposition.

```
//SYSUT1   DD  DSN=dsname,DISP=SHR
```

a. Do **not** replace SYSUT1 as the name for this JCL DD statement. Although you may select your own labels (known as ddnames) for most DD statements, the IEBGENER utility requires the use of SYSUT1 for the input data set.
b. Replace `dsname` with the name of the cataloged data set to be copied. Although this sample assumes you are using a cataloged data set for input, you may use an uncataloged data set. If you are, however, the JCL requirements for this input DD statement are slightly different: If your uncataloged data set is not SMS-managed, you need to add **UNIT** and **VOL=SER** parameters to this input DD statement. Use the instructions in the following step for modifying the output DD statement.
c. Change the value for the **DISP** parameter, if necessary. The **DISP** parameter tells the system about the status of your data set and what to do with it when your job ends, either normally or abnormally. As coded in this sample, the status subparameter value `SHR` (the abbreviation for "share") tells the system that your data set already exists, and can be used by other programs while your job is running. The subparameter values for job-end processing are not specified, so default values are in effect: Whether the job ends normally or not, the system will keep, rather than delete, this data set.

   If you want to understand more about disposition processing, refer to the summary of disposition processing in *z/OS MVS JCL Reference* (SA22-7597), in the DD statement topic for the **DISP** parameter.

6. Required: Modify this output DD statement to name the new copy and its location on tape.

```
//SYSUT2   DD  DSN=dsname,DISP=(NEW,CATLG),
//     UNIT=tapedevice,
//     VOL=SER=volser
```

a. Do **not** replace SYSUT2 as the name for this JCL DD statement. The IEBGENER utility requires the use of SYSUT2 for the output data set.
b. Replace `dsname` with the name that you want to use for the copy of your data set on tape.
c. If necessary, modify the disposition for the copied data set. As coded in this sample, this **DISP** parameter tells the system to create the data set (`NEW`) on tape, and add an entry for it in the system or user catalog (`CATLG`), whether the job step ends normally or abnormally. No changes are required for this parameter.

The remaining parameters that you use for the output DD statement depend on whether your company uses SMS.

- If you are using SMS, replace the **UNIT** parameter with the **STORCLAS** parameter and a storage class name that your company uses for tape devices (for example, `STORCLAS=SCLAS01`). Also, remove the **VOL=SER** parameter.
- If you are not using SMS and the output data set is cataloged, you may remove the **UNIT** and **VOL=SER** parameters. If the output data set is not cataloged:
  - Replace the **UNIT** parameter value `tapedevice` with a value that identifies tape devices. The value is usually the symbolic name of a group of devices; for example, `UNIT=SYS3480R` (SYS3480R is an IBM-assigned group name that includes several models of Magnetic Tape Subsystems).
  - Check with your mentor to determine whether you need to specify the **VOL=SER** parameter; company guidelines determine what you supply for it.

7. Optional: Check for JCL syntax errors by submitting the job with `TYPRUN=SCAN` on the JOB statement.

   ```
   //jobname  JOB  (start of JOB statement parameters),TYPRUN=SCAN
   ```

   Using `TYPRUN=SCAN` does not catch all possible JCL errors, but it's a good start to ensuring that your job will run. `TYPRUN=SCAN` requests that the system scan this job's JCL for syntax errors, without executing the job or allocating devices. This parameter asks the system to check for:
   - Spelling of parameters and some subparameters that is not correct.
   - Characters that are not correct.
   - Unbalanced parentheses.
   - Misplaced positional parameters on some statements.
   - In a JES3 system only, parameter value errors or excessive parameters.
   - Incorrect syntax on JCL statements in cataloged procedures invoked by any scanned EXEC statements.

   You might still encounter JCL errors after using TYPRUN=SCAN, because this request checks the JCL only through the converter, not the interpreter. The difference is that the converter basically checks all expressions to the **left** of an equal sign plus **some** expressions to the right of an equal sign (and issues messages that start with IEFC), while the interpreter checks all expressions to the **right** of an equal sign (and issues messages that start with IEF). For example, a data set name containing a qualifier that exceeds eight characters, such as `DSN=L9755TB.JCL.TEST19970103` would not be flagged by TYPRUN=SCAN but would be caught by the interpreter.
8. Required: Remove `TYPRUN=SCAN` from the JOB statement and submit the job. The system response is:

   ```
   JOB jobname(jobnumber) SUBMITTED
   ***
   ```

### Results

When the job ends, you will receive a message indicating one of three conditions: job successful, JCL error, or program abend. Use your installation's viewing facility (for example, SDSF) to view the output and determine whether the job completed successfully.

## Reusable JCL: Copying a partitioned data set

Modify this JCL sample to copy one cataloged partitioned data set (PDS) to a new PDS, using the DFSMSdfp utility IEBCOPY.

### Before you begin

- If you have not already done so, allocate a data set to contain your modified version of this JCL sample. Use the instructions in "JCL exercise: Creating and submitting a job" on page 47 to create this data set.
- Determine the information (account number, programmer name, and so on) your company requires for each job that you submit.
- Determine whether your company uses the Storage Management Subsystem (SMS) to automate the use of storage for data sets. The JCL parameters for SMS-managed (also called system-managed) data sets are different from some parameters used for non-SMS data sets.
- For direct-access storage devices (DASD), determine which naming conventions are used, as well as default or recommended values for data set attributes.

## About this task

The steps that follow provide line-by-line instructions for modifying this JCL sample:

```
//jobname  JOB  (start of JOB statement parameters)
//stepname EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=dsname,DISP=SHR
//SYSUT2   DD DSN=dsname,DISP=(NEW,CATLG),
//     SPACE=(CYL,(1,1,45)),
//     DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160),
//     UNIT=unit,
//     VOL=SER=volser
//SYSIN    DD *
    COPY INDD=SYSUT1,OUTDD=SYSUT2
/*
```

In the JCL statements that appear in code examples, lowercase text indicates items that you need to modify. Except for a few cases, lowercase alphabetic characters cannot be used in JCL.

All jobs require JOB and EXEC statements, so this sample contains both:
- The JOB statement marks the beginning of a job, specifies the job name, and also might provide company-specific details or JCL parameters that apply to all job steps within the job.
- The EXEC statement marks the beginning of a job step. In this case, the job step is to run IEBCOPY, which is a DFSMSdfp utility for copying or merging full or partial members between one or more partitioned data sets.

When you use the IEBCOPY utility, you must define several input and output data definition (DD) statements in the job step:
- The `SYSPRINT DD` statement tells the system where to print the informational or error messages from IEBCOPY.
- The `SYSUT1 DD` statement identifies the input data set to be copied.
- The `SYSUT2 DD` statement identifies the output data set where the copy of the input data set is to be placed.
- The `SYSIN DD` statement contains instructions for IEBCOPY to process.

You might want to refer to *z/OS MVS JCL Reference* (SA22-7597), which contains other syntax rules and statement parameter descriptions that might help if you modify this sample to complete more complex tasks, or if you encounter JCL errors.

Another useful reference is *z/OS DFSMSdfp Utilities* (SC26-7414), which contains instructions for using IEBCOPY.

## Procedure

1. Required: Modify the JOB statement to uniquely identify your job and to provide additional company-specific information.

   ```
   //jobname  JOB  (start of JOB statement parameters)
   ```

   a. Replace `jobname` with a unique name to identify this job. A common convention is to use your TSO logon ID followed by a number (for example: `ZUSER031`).

      Syntax rules for the name field are:
      - The name must begin in column 3 of the JCL statement.

- The name can be one through eight characters in length.
- The first character in the name must be an alphabetic character (the letters A through Z) or a special character (the symbols #, @, and $).
- The remaining characters can be alphanumeric (the letters A through Z and numbers 0 through 9) or special characters.
- Blank spaces cannot be included in a name.

b. Replace `(start of JOB statement parameters)` with parameters and values that conform to guidelines set at your company.

2. Required: Modify the EXEC statement to uniquely identify the job step and the utility to be run.

```
//stepname EXEC PGM=IEBCOPY
```

a. Replace `stepname` with a unique name to identify this step. Syntax rules for `stepname` are identical to those listed for `jobname`. Aside from changing the step name, no further changes are required.

3. Required: Include a `SYSPRINT DD` statement to tell the system where to print IEBCOPY messages.

```
//SYSPRINT DD SYSOUT=*
```

The `SYSPRINT DD` statement with `SYSOUT=*` tells the system to print the informational or error messages from IEBCOPY in the job log. Although you may use other parameter values for `SYSPRINT`, no changes are required for this `DD` statement.

4. Required: Modify this input DD statement to identify the cataloged PDS that you want to copy, and the data set disposition.

```
//SYSUT1   DD DSN=dsname,DISP=SHR
```

a. You may replace SYSUT1 as the label (or ddname) for this JCL DD statement. If you use a ddname other than SYSUT1, you must also replace SYSUT1 on the SYSIN DD statement with the new ddname.

   Syntax rules for ddnames are identical to those listed for job and step names on JOB and EXEC statements. By default, SYSUT1 is used as the ddname for the DD statement that identifies the input data set.

b. Replace `dsname` with the name of the data set to be copied. This sample assumes that your input data set is cataloged. If you are copying an uncataloged data set that is not SMS-managed, you must add the **UNIT** parameter to this input DD statement; you might need to add the **VOL=SER** parameter as well. Use the instructions in the following step (modifying the output DD statement `SYSUT2`) for providing**UNIT** and **VOL=SER** parameter values.

c. Change the value for the **DISP** parameter, if necessary. The **DISP** parameter tells the system about the status of your data set and what to do with it when your job ends, either normally or abnormally. As coded in this sample, the status subparameter value `SHR` tells the system that your data set already exists, and can be used by other programs while your job is running. The subparameter values for job-end processing are not specified, so default values are in effect. Whether the job ends normally or not, the system will keep, rather than delete, this data set.

   If you want to understand more about disposition processing, refer to the summary of disposition processing in *z/OS MVS JCL Reference* (SA22-7597), in the DD statement topic for the **DISP** parameter.

5. Required: Modify this output DD statement to create a new PDS to which IEBCOPY is to copy the contents of the input PDS.

```
//SYSUT2   DD DSN=dsname,DISP=(NEW,CATLG),
//     SPACE=(CYL,(1,1,45)),
//     DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160),
//     UNIT=unit,
//     VOL=SER=volser
```

a. You may replace SYSUT2 as the label (or ddname) for this JCL DD statement. If you use a ddname other than SYSUT2, you must also replace SYSUT2 on the SYSIN DD statement with the new ddname.

   By default, SYSUT2 is used as the ddname for the DD statement that identifies the output data set.

b. Change `dsname` to the name that you want to use for the copy of your input data set.

c. If necessary, modify the disposition for the output data set. Because this sample assumes you want to create a new PDS to contain a copy of the input data set, the code provided for the **DISP** parameter tells the system to create the data set (`NEW`), and add an entry for it in the system or user catalog (`CATLG`), whether the job step ends normally or abnormally. Although you may change these parameter values to use an existing data set, no changes are required for this parameter.

The remaining parameters that you use for the output DD statement depend on whether your company uses the SMS.

- If you are using SMS, replace the **SPACE**, **DCB**, **UNIT** and **VOL=SER** parameters with this sample code:

  ```
  //   DATACLAS=dataclassname,
  //   STORCLAS=storageclassname
  ```

  – Replace `dataclassname` with a data class name with data set characteristics that are similar to those of the input data set. Check with your mentor to find an appropriate data class value.

  – Replace `storageclassname` with a storage class name that your company uses for PDSes (for example, `STORCLAS=SCLAS01`).

- If you are not using SMS:

  – Change the **SPACE** parameter values to space attributes that are similar to those of the input data set. The **SPACE** parameter values are positional, so use the syntax diagram in *z/OS MVS JCL Reference* to make sure you code it correctly. The **SPACE** parameter in this sample is:

  ```
  SPACE=(CYL,(1,1,45))
  ```

  These parameter values tell z/OS to allocate space for the data set:

  - In cylinders (`CYL`), which is one of the four different ways in which z/OS measures space for data set allocation.
  - In quantities indicated by the set of parameter values (`1,1,45`) as follows:
    - The first number (1) is the primary quantity of cylinders.
    - The second number (1) is the secondary quantity of cylinders, which are allocated only when the primary quantity is not sufficient to hold the data set's contents.
    - The last number (45) is the number of 256-byte records needed for the directory of the new PDS.

  – Change the **DCB** subparameter values to space attributes that are similar to those of the input data set. The **DCB** parameter in this sample defines three attributes:

  ```
  DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160)
  ```

As written, this code tells z/OS to allocate a data set with:
- A record format (**RECFM**) of fixed blocks (FB).
- A logical record length (**LRECL**) of 80 bytes.
- A block size (**BLKSIZE**) of 6160 bytes.

You have a lot of flexibility with syntax for these parameters, so it's best to check with your mentor or read the descriptions of each parameter in *z/OS MVS JCL Reference*.

– Replace the **UNIT** parameter value `unit` with an appropriate value, which usually is the symbolic name of a group of devices; for example, `UNIT=SYSALLDA` (SYSALLDA is an IBM-assigned group name that includes contains all direct access devices defined to the system).

– Check with your mentor to determine whether you need to specify the **VOL=SER** parameter; company guidelines determine what you supply for it.

6. Required: Include a SYSIN DD statement, which contains instructions for IEBCOPY to process.

```
//SYSIN    DD *
    COPY INDD=SYSUT1,OUTDD=SYSUT2
/*
```

The SYSIN DD statement identifies an in-stream data set as the source of input for IEBCOPY to process. DD * or DD DATA marks the beginning of the in-stream data set; the delimiter /* marks the end of data.

The instream data set in this sample contains one job control statement for IEBCOPY: The COPY statement tells IEBCOPY to copy the input data set (INDD=SYSUT1) to the output data set (OUTDD=SYSUT2).

• If you used a ddname other than SYSUT1 for your input data set DD statement, replace SYSUT1 with that new ddname.
• If you used a ddname other than SYSUT2 for your output data set DD statement, replace SYSUT2 with that new ddname.

7. Optional: Check for JCL syntax errors by submitting the job with TYPRUN=SCAN on the JOB statement.

```
//jobname  JOB  (start of JOB statement parameters),TYPRUN=SCAN
```

Using TYPRUN=SCAN does not catch all possible JCL errors, but it's a good start to ensuring that your job will run. TYPRUN=SCAN requests that the system scan this job's JCL for syntax errors, without executing the job or allocating devices. This parameter asks the system to check for:
• Spelling of parameters and some subparameters that is not correct.
• Characters that are not correct.
• Unbalanced parentheses.
• Misplaced positional parameters on some statements.
• In a JES3 system only, parameter value errors or excessive parameters.
• Incorrect syntax on JCL statements in cataloged procedures invoked by any scanned EXEC statements.

You might still encounter JCL errors after using TYPRUN=SCAN, because this request checks the JCL only through the converter, not the interpreter. The difference is that the converter basically checks all expressions to the **left** of an equal sign plus **some** expressions to the right of an equal sign (and issues messages that start with IEFC), while the interpreter checks all expressions to the **right** of an equal sign (and issues messages that start with IEF). For example, a data set name containing a qualifier that exceeds eight characters, such as DSN=L9755TB.JCL.TEST19970103 would not be flagged by TYPRUN=SCAN but would be caught by the interpreter.

8. Required: Remove TYPRUN=SCAN from the JOB statement and submit the job. The system response is:

   ```
   JOB jobname(jobnumber) SUBMITTED
   ***
   ```

### Results

When the job ends, you will receive a message indicating one of three conditions: job successful, JCL error, or program abend. Use your installation's viewing facility (for example, SDSF) to view the output and determine whether the job completed successfully.

One possible error condition that you might encounter is a region size that is too small to successfully complete the job step. If your job ends with system completion code 804 or 80A, which result when this condition is true, you can add a **REGION** parameter to the EXEC statement and resubmit the job. The optional **REGION** parameter overrides the default limit of storage that the system allocates to a particular job step. Check with your mentor to determine an appropriate value to specify for **REGION**; it is not a parameter to be used without advice.

## Reusable JCL: Copying a sequential data set

Modify this JCL sample to copy one cataloged sequential data set into another existing sequential data set, using the DFSMSdfp utility IEBGENER.

### Before you begin

- If you have not already done so, allocate a data set to contain your modified version of this JCL sample. Use the instructions in "JCL exercise: Creating and submitting a job" on page 47 to create this data set.
- Determine the information (account number, programmer name, and so on) your company requires for each job that you submit.
- Determine whether your company uses the Storage Management Subsystem (SMS) to automate the use of storage for data sets. The JCL parameters for SMS-managed (also called system-managed) data sets are different from some parameters used for non-SMS data sets.

### About this task

The steps that follow provide line-by-line instructions for modifying this JCL sample:

```
//jobname  JOB  (start of JOB statement parameters)
//stepname EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//SYSUT1   DD DSN=dsname,DISP=SHR
//SYSUT2   DD DSN=dsname,DISP=OLD,
//     UNIT=unit,
//     VOL=SER=volser
/*
```

In the JCL statements that appear in code examples, lowercase text indicates items that you need to modify. Except for a few cases, lowercase alphabetic characters cannot be used in JCL.

All jobs require JOB and EXEC statements, so this sample contains both:

- The JOB statement marks the beginning of a job, specifies the job name, and also might provide company-specific details or JCL parameters that apply to all job steps within the job.
- The EXEC statement marks the beginning of a job step. In this case, the job step is to run the program IEBGENER, which is a DFSMSdfp utility with a variety of uses.

When you use the IEBGENER utility, you must define several input and output data definition (DD) statements in the job step:

- The `SYSPRINT DD` statement tells the system where to print the informational or error messages from IEBGENER.
- The `SYSIN DD` statement identifies a control data set that IEBGENER uses in some cases.
- The `SYSUT1 DD` statement identifies the cataloged sequential data set that you want IEBGENER to copy.
- The `SYSUT2 DD` statement identifies the output data set where your sequential input data set is to be copied.

If you modify this sample to complete more complex tasks, or if you encounter JCL errors, see *z/OS MVS JCL Reference* (SA22-7597), which is the comprehensive source of syntax rules and statement parameter descriptions.

Another useful reference is *z/OS DFSMSdfp Utilities* (SC26-7414), which contains instructions for using IEBGENER.

## Procedure

1. Required: Modify the JOB statement to uniquely identify your job and to provide additional company-specific information.

   ```
   //jobname  JOB  (start of JOB statement parameters)
   ```

   a. Replace `jobname` with a unique name to identify this job. A common convention is to use your TSO logon ID followed by a number (for example: `ZUSER031`).

      Syntax rules for the name field are:
      - The name must begin in column 3 of the JCL statement.
      - The name can be one through eight characters in length.
      - The first character in the name must be an alphabetic character (the letters A through Z) or a special character (the symbols #, @, and $).
      - The remaining characters can be alphanumeric (the letters A through Z and numbers 0 through 9) or special characters.
      - Blank spaces cannot be included in a name.

   b. Replace `(start of JOB statement parameters)` with parameters and values that conform to guidelines set at your company.

2. Required: Modify the EXEC statement to uniquely identify the job step and the utility to be run.

   ```
   //stepname EXEC PGM=IEBGENER
   ```

   a. Replace `stepname` with a unique name to identify this step. Syntax rules for `stepname` are identical to those listed for `jobname`. Aside from changing the step name, no further changes are required.

3. Include a `SYSPRINT DD` statement to tell the system where to print IEBGENER messages.

   ```
   //SYSPRINT DD SYSOUT=*
   ```

The SYSPRINT DD statement with SYSOUT=* tells the system to print the informational or error messages from IEBGENER in the job log. Although you may use other parameter values for SYSPRINT, no changes are required for this DD statement.

4. Include a SYSIN DD statement.

   ```
   //SYSIN    DD DUMMY
   ```

   The SYSIN DD statement identifies a control data set that IEBGENER uses in some cases. When you are copying a sequential data set, however, IEBGENER does not need a control data set unless the input data set is to be edited as part of the copy operation, or the output data set is a partitioned data set. In this case, the output data set is sequential, so this **DUMMY** parameter tells the system that no resources are required for the control data set. No changes are required for this DD statement.

5. Modify this input DD statement to identify the sequential data set that you want to copy, and the data set disposition.

   ```
   //SYSUT1   DD DSN=dsname,DISP=SHR
   ```

   a. Do **not** replace SYSUT1 as the name for this JCL DD statement. Although you may select your own labels (known as ddnames) for most DD statements, the IEBGENER utility requires the use of SYSUT1 for the input data set.
   b. Replace dsname with the name of the sequential data set to be copied. Although this sample assumes you are using a cataloged data set for input, you may use an uncataloged data set. If you are, however, the JCL requirements for this input DD statement are slightly different: If your uncataloged data set is not SMS-managed, you need to add **UNIT** and **VOL=SER** parameters to this input DD statement. Use the instructions in the following step for modifying the output DD statement.
   c. Change the value for the **DISP** parameter, if necessary. The **DISP** parameter tells the system about the status of your data set and what to do with it when your job ends, either normally or abnormally. As coded in this sample, the status subparameter value SHR tells the system that your data set already exists, and can be used by other programs while your job is running. The subparameter values for job-end processing are not specified, so default values are in effect. Whether the job ends normally or not, the system will keep, rather than delete, this data set. No changes are required for this parameter.

      If you want to understand more about disposition processing, refer to the summary of disposition processing in *z/OS MVS JCL Reference* (SA22-7597), in the DD statement topic for the **DISP** parameter.

6. Modify this output DD statement to identify the existing sequential data set to which IEBGENER is to copy the input data set.

   ```
   //SYSUT2   DD DSN=dsname,DISP=OLD,
   //     UNIT=unit,
   //     VOL=SER=volser
   ```

   a. Do **not** replace SYSUT2 as the name for this JCL DD statement. The IEBGENER utility requires the use of SYSUT2 for the output data set.
   b. Replace dsname with the name of the data set to which IEBGENER is to copy the input sequential data set.
   c. Change the value for the **DISP** parameter, if necessary. The disposition in this sample is OLD, which indicates that the output data set exists before this step and that this step requires exclusive (unshared) use of the data set. The subparameter values for job-end processing are not specified, so default

values are in effect. Whether the job ends normally or not, the system will keep, rather than delete, this data set. No changes are required for this parameter.

The remaining parameters that you use for the output DD statement depend on whether your company uses SMS.

- If you are using SMS, replace the **UNIT** parameter with the **STORCLAS** parameter and a storage class name that your company uses for PDSes (for example, `STORCLAS=SCLAS01`). Also, remove the **VOL=SER** parameter.
- If you are not using SMS and the output data set is cataloged, you may remove the UNIT and VOL=SER parameters. If the output data set is not cataloged:
  - Replace the **UNIT** parameter value `unit` with an appropriate value, which usually is the symbolic name of a group of devices; for example, `UNIT=SYSALLDA` (SYSALLDA is an IBM-assigned group name that includes contains all direct access devices defined to the system).
  - Check with your mentor to determine whether you need to specify the **VOL=SER** parameter; company guidelines determine what you supply for it.

7. Optional: Check for JCL syntax errors by submitting the job with `TYPRUN=SCAN` on the JOB statement.

   ```
   //jobname  JOB  (start of JOB statement parameters),TYPRUN=SCAN
   ```

   Using `TYPRUN=SCAN` does not catch all possible JCL errors, but it's a good start to ensuring that your job will run. `TYPRUN=SCAN` requests that the system scan this job's JCL for syntax errors, without executing the job or allocating devices. This parameter asks the system to check for:
   - Spelling of parameters and some subparameters that is not correct.
   - Characters that are not correct.
   - Unbalanced parentheses.
   - Misplaced positional parameters on some statements.
   - In a JES3 system only, parameter value errors or excessive parameters.
   - Incorrect syntax on JCL statements in cataloged procedures invoked by any scanned EXEC statements.

   You might still encounter JCL errors after using TYPRUN=SCAN, because this request checks the JCL only through the converter, not the interpreter. The difference is that the converter basically checks all expressions to the **left** of an equal sign plus **some** expressions to the right of an equal sign (and issues messages that start with IEFC), while the interpreter checks all expressions to the **right** of an equal sign (and issues messages that start with IEF). For example, a data set name containing a qualifier that exceeds eight characters, such as `DSN=L9755TB.JCL.TEST19970103` would not be flagged by TYPRUN=SCAN but would be caught by the interpreter.
8. Required: Remove `TYPRUN=SCAN` from the JOB statement and submit the job. The system response is:

   ```
   JOB jobname(jobnumber) SUBMITTED
   ***
   ```

## Results

When the job ends, you will receive a message indicating one of three conditions: job successful, JCL error, or program abend. Use your installation's viewing facility (for example, SDSF) to view the output and determine whether the job completed successfully.

One possible error condition that you might encounter is a region size that is too small to successfully complete the job step. If your job ends with system completion code 804 or 80A, which result when this condition is true, you can add a **REGION** parameter to the EXEC statement and resubmit the job. The optional **REGION** parameter overrides the default limit of storage that the system allocates to a particular job step. Check with your mentor to determine an appropriate value to specify for **REGION**; it is not a parameter to be used without advice.

## Reusable JCL: Copying a load module

Modify this JCL sample to copy one load module from one cataloged partitioned data set (PDS) to another existing PDS, using the DFSMSdfp utility IEBCOPY.

### Before you begin

- If you have not already done so, allocate a data set to contain your modified version of this JCL sample. Use the instructions in "JCL exercise: Creating and submitting a job" on page 47 to create this data set.
- Determine the information (account number, programmer name, and so on) your company requires for each job that you submit.
- For direct-access storage devices (DASD), determine which naming conventions are used, as well as default or recommended values for data set attributes.
- Determine which job entry subsystem (JES2 or JES3) is installed on the z/OS system you will use. For many jobs, the type of JES does not affect JCL parameters; for certain jobs, however, the JES in use does dictate which JCL parameters, values, or job entry control (JECL) statements you may code.
- Determine whether your company uses the Storage Management Subsystem (SMS) to automate the use of storage for data sets. The JCL parameters for SMS-managed (also called system-managed) data sets are different from some parameters used for non-SMS data sets.

### About this task

The steps that follow provide line-by-line instructions for modifying this JCL sample:

```
//jobname  JOB  (start of JOB statement parameters)
//stepname EXEC PGM=IEBCOPY,REGION=4M
/*JOBPARM SYSAFF=*,LINES=99
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=dsname,DISP=SHR
//SYSUT2   DD DSN=dsname,DISP=SHR,
//     UNIT=unit,
//     VOL=SER=volser
//SYSUT3   DD UNIT=VIO,SPACE=(CYL,(10))
//SYSUT4   DD UNIT=VIO,SPACE=(CYL,(10))
//SYSIN    DD *
  COPYMOD INDD=SYSUT1,OUTDD=SYSUT2,MAXBLK=32760
  SELECT M=(membername)
/*
```

In the JCL statements that appear in code examples, lowercase text indicates items that you need to modify. Except for a few cases, lowercase alphabetic characters cannot be used in JCL.

All jobs require JOB and EXEC statements, so this sample contains both:

- The JOB statement marks the beginning of a job, specifies the job name, and also might provide company-specific details or JCL parameters that apply to all job steps within the job.
- The EXEC statement marks the beginning of a job step. In this case, the job step is to run IEBCOPY, which is a DFSMSdfp utility for copying or merging full or partial members between one or more partitioned data sets.

When you use the IEBCOPY utility, you must define several input and output data definition (DD) statements in the job step:
- The `SYSPRINT DD` statement tells the system where to print the informational or error messages from IEBCOPY.
- The `SYSUT1 DD` statement identifies the input data set where your load module resides.
- The `SYSUT2 DD` statement identifies the output data set where the copy of your load module is to be placed.
- The `SYSIN DD` statement contains instructions for IEBCOPY to process.

You also may define two optional spill data sets on a virtual I/O (VIO) device; these temporary data sets are used only when an insufficient amount of virtual storage is available for some or all of the current input data set directory entries, or for the output data set directory. This JCL sample contains DD statements for both of these temporary data sets.

If you modify this sample to complete more complex tasks, or if you encounter JCL errors, see *z/OS MVS JCL Reference* (SA22-7597), which is the comprehensive source of syntax rules and statement parameter descriptions.

Another useful reference is *z/OS DFSMSdfp Utilities* (SC26-7414), which contains information about using IEBCOPY.

## Procedure

1. Required: Modify the JOB statement to uniquely identify your job and to provide additional company-specific information.
   ```
   //jobname  JOB  (start of JOB statement parameters)
   ```
   a. Replace `jobname` with a unique name to identify this job. A common convention is to use your TSO logon ID followed by a number (for example: `ZUSER031`).

      Syntax rules for the name field are:
      - The name must begin in column 3 of the JCL statement.
      - The name can be one through eight characters in length.
      - The first character in the name must be an alphabetic character (the letters A through Z) or a special character (the symbols #, @, and $).
      - The remaining characters can be alphanumeric (the letters A through Z and numbers 0 through 9) or special characters.
      - Blank spaces cannot be included in a name.

   b. Replace `(start of JOB statement parameters)` with parameters and values that conform to guidelines set at your company.
2. Required: Modify the EXEC statement to uniquely identify the job step and the utility to be run.
   ```
   //stepname EXEC PGM=IEBCOPY
   ```
   a. Replace `stepname` with a unique name to identify this step. Syntax rules for `stepname` are identical to those listed for `jobname`. Aside from changing the step name, no further changes are required.

3. Optional: Override company-defined default values for maximum lines of output, for the system on which this job is to run, or both. If you do not want to override these default values, remove this line of code from the sample.

| To override defaults on this job entry subsystem: | Use this job entry control language (JECL) statement: |
|---|---|
| JES2 | `/*JOBPARM SYSAFF=*,LINES=99` |
| JES3 | `//*MAIN SYSTEM=JLOCAL,LINES=99` |

   - The **SYSAFF** or **SYSTEM** parameter ensures that the conversion and execution of the job will be done on a specific system. Specifying SYSAFF=* or SYSTEM=JLOCAL means that the job will be run on the system into which the job is read.
   - The **LINES** parameter specifies the maximum output, in thousands of lines, that the job entry subsystem is to allow on spool data sets for this job's sysout data sets.

   Note the syntax rules for the JES3 //*MAIN statement: Place the characters //* in columns 1 through 3, MAIN in columns 4 through 7, a blank in column 8, and parameters beginning in column 9.
4. Required: Include a SYSPRINT DD statement to tell the system where to print IEBCOPY messages.

```
//SYSPRINT DD SYSOUT=*
```

   The SYSPRINT DD statement with SYSOUT=* tells the system to print the informational or error messages from IEBCOPY in the job log. Although you may use other parameter values for SYSPRINT, no changes are required for this DD statement.
5. Required: Modify this input DD statement to identify the partitioned data set that contains the load module that you want to copy, and the data set disposition.

```
//SYSUT1   DD DSN=dsname,DISP=SHR
```

   a. You may replace SYSUT1 as the label (or ddname) for this JCL DD statement. If you use a ddname other than SYSUT1, you must also replace SYSUT1 on the SYSIN DD statement with the new ddname.

      Syntax rules for ddnames are identical to those listed for job and step names on JOB and EXEC statements. By default, SYSUT1 is used as the ddname for the DD statement that identifies the input data set.
   b. Replace dsname with the name of the data set that contains the load module. This sample assumes that your input data set is cataloged. If you are copying a load module from an uncataloged data set that is not SMS-managed, you must add the **UNIT** parameter to this input DD statement; you might need to add the **VOL=SER** parameter as well. Use the instructions in the following step (modifying the output DD statement SYSUT2) for providing**UNIT** and **VOL=SER** parameter values.
   c. Change the value for the **DISP** parameter, if necessary. The **DISP** parameter tells the system about the status of your data set and what to do with it when your job ends, either normally or abnormally. As coded in this sample, the status subparameter value SHR tells the system that your data set already exists, and can be used by other programs while your job is running. The subparameter values for job-end processing are not

specified, so default values are in effect. Whether the job ends normally or not, the system will keep, rather than delete, this data set. No changes are required for this parameter.

If you want to understand more about disposition processing, refer to the summary of disposition processing in *z/OS MVS JCL Reference* (SA22-7597), in the DD statement topic for the **DISP** parameter.

6. Required: Modify this output DD statement to identify the partitioned data set to which IEBCOPY is to copy the load module, the data set disposition, and the device on which the data set resides.

```
//SYSUT2   DD DSN=dsname,DISP=SHR,
//     UNIT=unit,
//     VOL=SER=volser
```

   a. You may replace SYSUT2 as the label (or ddname) for this JCL DD statement. If you use a ddname other than SYSUT2, you must also replace SYSUT2 on the SYSIN DD statement with the new ddname.

      By default, SYSUT2 is used as the ddname for the DD statement that identifies the output data set.

   b. Replace `dsname` with the name of the data set into which IEBCOPY is to copy the load module. Remember, this output data set must exist before you run this job.

   c. Change the value for the **DISP** parameter, if necessary. The disposition in this sample is share (DISP=SHR) so that other programs can use the data set while this job is running. The subparameter values for job-end processing are not specified, so default values are in effect. Whether the job ends normally or not, the system will keep, rather than delete, this data set. No changes are required for this parameter.

   The remaining parameters that you use for the output DD statement depend on whether your company uses the Storage Management Subsystem (SMS).

   - If you are using SMS, replace the **UNIT** and **VOL=SER** parameters with the **STORCLAS** parameter and a storage class name that your company uses for PDSes (for example, `STORCLAS=SCLAS01`).
   - If you are not using SMS:
     – Replace the **UNIT** parameter value `unit` with an appropriate value, which usually is the symbolic name of a group of devices; for example, `UNIT=SYSALLDA` (SYSALLDA is an IBM-assigned group name that contains all direct access devices defined to the system).
     – Check with your mentor to determine whether you need to specify the **VOL=SER** parameter; company guidelines determine what you supply for it.

7. Optional: Remove or modify the DD statements that define temporary data sets for this copy operation.

```
//SYSUT3   DD UNIT=VIO,SPACE=(CYL,(10))
//SYSUT4   DD UNIT=VIO,SPACE=(CYL,(10))
```

   SYSUT3 defines an optional spill data set on a virtual I/O (VIO) device. SYSUT3 is used only when an insufficient amount of virtual storage is available for some or all of the current input data set directory entries. Similarly, SYSUT4 defines an optional spill data set to be used only when an insufficient amount of virtual storage is available for the output data set directory.

   This sample's SYSUT3 and SYSUT4 DD statements tell the system to allocate 10 cylinders of space for the temporary VIO data sets. If you use these two DD statements, you must replace at least the value specified for the **UNIT** parameter; companies typically choose their own unit name for VIO devices. Ask your mentor for advice about coding these temporary data sets.

8. Required: Modify the SYSIN DD statement, which contains instructions for IEBCOPY to process.

   ```
   //SYSIN    DD *
     COPYMOD INDD=SYSUT1,OUTDD=SYSUT2,MAXBLK=32760
     SELECT M=(membername)
   /*
   ```

   The SYSIN DD statement identifies an in-stream data set as the source of input for IEBCOPY to process. DD * or DD DATA marks the beginning of the in-stream data set; the delimiter /* marks the end of data.

   The instream data set in this sample contains two job control statements for IEBCOPY:

   - The COPYMOD statement tells IEBCOPY not only to copy your load module, but also to reblock it on the data set to which it will be copied. Reblocking is usually done to make the record length of the output data set compatible with different devices or programs. Reblocking must be done if the input data set has a larger block size than the output data set.
     - If you used a ddname other than SYSUT1 for your input data set DD statement, replace SYSUT1 with that new ddname.
     - If you used a ddname other than SYSUT2 for your output data set DD statement, replace SYSUT2 with that new ddname.
     - MAXBLK specifies the maximum block size for records in the output partitioned data set. This sample shows a MAXBLK value in decimal form; instead, you may use the *nn*K format where *nn* is a decimal number and the letter K indicates that the *nn* value is to be multiplied by 1024 bytes.

     If your changes make the COPYMOD statement exceed 71 characters in length, note that continuation rules are slightly different than those for continuing other JCL statements on subsequent lines:
     - End the first line with a comma that separates parameters, pad with blanks out to column 72, and place a nonblank character in column 72.
     - Start the next line in column 16 with a parameter and its value.
   - The SELECT statement identifies which member of the input data set is to be loaded. Replace `membername` with the name of the load module that you want IEBCOPY to copy.

9. Optional: Check for JCL syntax errors by submitting the job with `TYPRUN=SCAN` on the JOB statement.

   ```
   //jobname  JOB  (start of JOB statement parameters),TYPRUN=SCAN
   ```

   Using `TYPRUN=SCAN` does not catch all possible JCL errors, but it's a good start to ensuring that your job will run. `TYPRUN=SCAN` requests that the system scan this job's JCL for syntax errors, without executing the job or allocating devices. This parameter asks the system to check for:
   - Spelling of parameters and some subparameters that is not correct.
   - Characters that are not correct.
   - Unbalanced parentheses.
   - Misplaced positional parameters on some statements.
   - In a JES3 system only, parameter value errors or excessive parameters.
   - Incorrect syntax on JCL statements in cataloged procedures invoked by any scanned EXEC statements.

You might still encounter JCL errors after using TYPRUN=SCAN, because this request checks the JCL only through the converter, not the interpreter. The difference is that the converter basically checks all expressions to the **left** of an equal sign plus **some** expressions to the right of an equal sign (and issues messages that start with IEFC), while the interpreter checks all expressions to the **right** of an equal sign (and issues messages that start with IEF). For example, a data set name containing a qualifier that exceeds eight characters, such as `DSN=L9755TB.JCL.TEST19970103` would not be flagged by TYPRUN=SCAN but would be caught by the interpreter.

10. Required: Remove `TYPRUN=SCAN` from the JOB statement and submit the job. The system response is:

    ```
    JOB jobname(jobnumber) SUBMITTED
    ***
    ```

### Results

When the job ends, you will receive a message indicating one of three conditions: job successful, JCL error, or program abend. Use your installation's viewing facility (for example, SDSF) to view the output and determine whether the job completed successfully.

One possible error condition that you might encounter is a region size that is too small to successfully complete the job step. If your job ends with system completion code 804 or 80A, which result when this condition is true, you can add a **REGION** parameter to the EXEC statement and resubmit the job. The optional **REGION** parameter overrides the default limit of storage that the system allocates to a particular job step. Check with your mentor to determine an appropriate value to specify for **REGION**; it is not a parameter to be used without advice.

## Reusable JCL: Deleting a data set

Modify this JCL sample to delete an existing data set, using the IBM program IEFBR14.

### Before you begin

- If you have not already done so, allocate a data set to contain your modified version of this JCL sample. Use the instructions in "JCL exercise: Creating and submitting a job" on page 47 to create this data set.
- Determine the information (account number, programmer name, and so on) your company requires for each job that you submit.

### About this task

The steps that follow provide line-by-line instructions for modifying this JCL sample:

```
//jobname  JOB (start of JOB statement parameters)
//stepname EXEC PGM=IEFBR14
//SYSPRINT DD  SYSOUT=*
//ddname   DD  DSN=dsname,
//     DISP=(OLD,DELETE,DELETE)
/*
```

In the JCL statements that appear in code examples, lowercase text indicates items that you need to modify. Except for a few cases, lowercase alphabetic characters cannot be used in JCL.

All jobs require JOB and EXEC statements, so this sample contains both:

- The JOB statement marks the beginning of a job, specifies the job name, and also might provide company-specific details or JCL parameters that apply to all job steps within the job.
- The EXEC statement marks the beginning of a job step. In this case, the job step is to run IEFBR14, which is a program that simply passes control back to z/OS. For steps that call IEFBR14, then, other JCL statements within the step specify any work that z/OS does.

If you modify this sample to complete more complex tasks, or if you encounter JCL errors, see *z/OS MVS JCL Reference* (SA22-7597), which is the comprehensive source of syntax rules and statement parameter descriptions.

## Procedure

1. Required: Modify the JOB statement to uniquely identify your job and to provide additional company-specific information.
   ```
   //jobname  JOB  (start of JOB statement parameters)
   ```
   a. Replace `jobname` with a unique name to identify this job. A common convention is to use your TSO logon ID followed by a number (for example: `ZUSER031`).

      Syntax rules for the name field are:
      - The name must begin in column 3 of the JCL statement.
      - The name can be one through eight characters in length.
      - The first character in the name must be an alphabetic character (the letters A through Z) or a special character (the symbols #, @, and $).
      - The remaining characters can be alphanumeric (the letters A through Z and numbers 0 through 9) or special characters.
      - Blank spaces cannot be included in a name.

   b. Replace `(start of JOB statement parameters)` with parameters and values that conform to guidelines set at your company.
2. Required: Modify the EXEC statement to uniquely identify the job step and the utility to be run.
   ```
   //stepname EXEC PGM=IEFBR14
   ```
   a. Replace `stepname` with a unique name to identify this step. Syntax rules for `stepname` are identical to those listed for `jobname`. Aside from changing the step name, no further changes are required.
3. Optional: Include a `SYSPRINT DD` statement to tell the system where to print messages.
   ```
   //SYSPRINT DD  SYSOUT=*
   ```
   The `SYSPRINT DD` statement with `SYSOUT=*` tells the system to print the informational or error messages in the job log. Although you may use other parameter values for `SYSPRINT`, no changes are required for this `DD` statement.
4. Required: Modify this input DD statement to define the data set to be deleted.
   ```
   //ddname   DD  DSN=dsname,
   //     DISP=(OLD,DELETE,DELETE)
   ```
   a. Replace `ddname` with a unique name for this JCL DD statement; this label in the name field of a DD statement is known as a ddname. Syntax rules for ddnames are identical to those listed for job and step names on JOB and EXEC statements.

   b. Replace `dsname` with the name of the data set to be deleted.

c. Change the value for the **DISP** parameter, if necessary. The **DISP** parameter tells the system about the status of your data set and what to do with it when your job ends, either normally or abnormally. As coded in this sample, the status subparameter value (`OLD`) tells the system that the data set already exists. Both subparameter values for job-end processing (`DELETE`) are specified, so the system will delete the data set whether the job step ends normally or abnormally.

5. Optional: Check for JCL syntax errors by submitting the job with `TYPRUN=SCAN` on the JOB statement.

   ```
   //jobname  JOB  (start of JOB statement parameters),TYPRUN=SCAN
   ```

   Using `TYPRUN=SCAN` does not catch all possible JCL errors, but it's a good start to ensuring that your job will run. `TYPRUN=SCAN` requests that the system scan this job's JCL for syntax errors, without executing the job or allocating devices. This parameter asks the system to check for:
   - Spelling of parameters and some subparameters that is not correct.
   - Characters that are not correct.
   - Unbalanced parentheses.
   - Misplaced positional parameters on some statements.
   - In a JES3 system only, parameter value errors or excessive parameters.
   - Incorrect syntax on JCL statements in cataloged procedures invoked by any scanned EXEC statements.

   You might still encounter JCL errors after using TYPRUN=SCAN, because this request checks the JCL only through the converter, not the interpreter. The difference is that the converter basically checks all expressions to the **left** of an equal sign plus **some** expressions to the right of an equal sign (and issues messages that start with IEFC), while the interpreter checks all expressions to the **right** of an equal sign (and issues messages that start with IEF). For example, a data set name containing a qualifier that exceeds eight characters, such as `DSN=L9755TB.JCL.TEST19970103` would not be flagged by TYPRUN=SCAN but would be caught by the interpreter.

6. Required: Remove `TYPRUN=SCAN` from the JOB statement and submit the job. The system response is:

   ```
   JOB jobname(jobnumber) SUBMITTED
   ***
   ```

### Results

When the job ends, you will receive a message indicating one of three conditions: job successful, JCL error, or program abend. Use your installation's viewing facility (for example, SDSF) to view the output and determine whether the job completed successfully.

## Reusable JCL: Deleting some VSAM clusters

Modify this JCL sample to delete one or more VSAM clusters, using DFSMSdfp access method services (IDCAMS).

### Before you begin

- If you have not already done so, allocate a data set to contain your modified version of this JCL sample. Use the instructions in "JCL exercise: Creating and submitting a job" on page 47 to create this data set.
- Determine the information (account number, programmer name, and so on) your company requires for each job that you submit.

## About this task

The steps that follow provide line-by-line instructions for modifying this JCL sample:

```
//jobname  JOB  (start of JOB statement parameters)
//stepname EXEC PGM=IDCAMS
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
  DELETE 'entryname'
  DELETE 'entryname,entryname,entryname'
/*
```

In the JCL statements that appear in code examples, lowercase text indicates items that you need to modify. Except for a few cases, lowercase alphabetic characters cannot be used in JCL.

All jobs require JOB and EXEC statements, so this sample contains both:
- The JOB statement marks the beginning of a job, specifies the job name, and also might provide company-specific details or JCL parameters that apply to all job steps within the job.
- The EXEC statement marks the beginning of a job step. In this case, the job step is to run access method services, which is a DFSMSdfp utility for managing catalogs. IDCAMS is its program name.

When you use the access method services through JCL, you must define two specific DD statements in the job step:
- The `SYSPRINT DD` parameter tells the system where to print IDCAMS messages.
- The `SYSIN DD` statement identifies an in-stream data set containing input for access method services to process.

If you modify this sample to complete more complex tasks, or if you encounter JCL errors, see *z/OS MVS JCL Reference* (SA22-7597), which is the comprehensive source of syntax rules and statement parameter descriptions.

Another useful reference is *z/OS DFSMS Access Method Services for Catalogs* (SC26-7394), which contains background information about VSAM clusters and details about using IDCAMS for other operations.

## Procedure

1. Required: Modify the JOB statement to uniquely identify your job and to provide additional company-specific information.

   ```
   //jobname  JOB  (start of JOB statement parameters)
   ```

   a. Replace `jobname` with a unique name to identify this job. A common convention is to use your TSO logon ID followed by a number (for example: `ZUSER031`).

      Syntax rules for the name field are:
      - The name must begin in column 3 of the JCL statement.
      - The name can be one through eight characters in length.
      - The first character in the name must be an alphabetic character (the letters A through Z) or a special character (the symbols #, @, and $).
      - The remaining characters can be alphanumeric (the letters A through Z and numbers 0 through 9) or special characters.
      - Blank spaces cannot be included in a name.

   b. Replace `(start of JOB statement parameters)` with parameters and values that conform to guidelines set at your company.

2. Required: Replace `stepname` with a unique name to identify this step.

   ```
   //stepname EXEC PGM=IDCAMS
   ```

   Syntax rules for `stepname` are identical to those listed for `jobname`. Aside from changing the step name, no further changes are required.

3. Required: Use the `SYSPRINT DD` parameter to tell the system where to print IDCAMS messages.

   ```
   //SYSPRINT DD  SYSOUT=*
   ```

   The `SYSPRINT DD` statement with `SYSOUT=*` tells the system to print the informational or error messages from IDCAMS in the job log. Although you may use other parameter values for `SYSPRINT`, no changes are required for this DD statement.

4. Required: Use the `SYSIN DD` statement to identify an in-stream data set as the source of input for access method services to process.

   ```
   //SYSIN    DD  *
     DELETE 'entryname'
     DELETE 'entryname,entryname,entryname'
   /*
   ```

   `DD *` or `DD DATA` marks the beginning of the in-stream data set; the delimiter `/*` marks the end of data.

   Replace `entryname` with the names of clusters that you want to delete. As coded in this sample, the `DELETE` commands illustrate two of several possible formats for specifying entries on this access methods services command:

   - The first DELETE command specifies only one entry to be deleted; for example: `DELETE 'DB8YU.DSNDBC.CC390'`
   - The second DELETE command lists several entries to be deleted. Multiple entries must be enclosed in quotes (as shown above), and separated by a comma.

   If your changes make the DELETE statement exceed 71 characters in length, note that continuation rules are slightly different than those for continuing other JCL statements on subsequent lines. To continue a command on several lines, type either a hyphen or a plus sign as the last nonblank character before, or in, column 72:

   - A hyphen continues the command after a completely specified value:

     ```
     DELETE entryname-
            entryname
     ```

   - A plus sign continues both the command and a value within the command:

     ```
     DELETE entry+
            name
     ```

5. Optional: Check for JCL syntax errors by submitting the job with `TYPRUN=SCAN` on the JOB statement.

   ```
   //jobname  JOB  (start of JOB statement parameters),TYPRUN=SCAN
   ```

   Using `TYPRUN=SCAN` does not catch all possible JCL errors, but it’s a good start to ensuring that your job will run. `TYPRUN=SCAN` requests that the system scan this job’s JCL for syntax errors, without executing the job or allocating devices. This parameter asks the system to check for:

   - Spelling of parameters and some subparameters that is not correct.
   - Characters that are not correct.
   - Unbalanced parentheses.
   - Misplaced positional parameters on some statements.
   - In a JES3 system only, parameter value errors or excessive parameters.

- Incorrect syntax on JCL statements in cataloged procedures invoked by any scanned EXEC statements.

  You might still encounter JCL errors after using TYPRUN=SCAN, because this request checks the JCL only through the converter, not the interpreter. The difference is that the converter basically checks all expressions to the **left** of an equal sign plus **some** expressions to the right of an equal sign (and issues messages that start with IEFC), while the interpreter checks all expressions to the **right** of an equal sign (and issues messages that start with IEF). For example, a data set name containing a qualifier that exceeds eight characters, such as `DSN=L9755TB.JCL.TEST19970103` would not be flagged by TYPRUN=SCAN but would be caught by the interpreter.

6. Required: Remove `TYPRUN=SCAN` from the JOB statement and submit the job. The system response is:

   ```
   JOB jobname(jobnumber) SUBMITTED
   ***
   ```

## Results

When the job ends, you will receive a message indicating one of three conditions: job successful, JCL error, or program abend. Use your installation's viewing facility (for example, SDSF) to view the output and determine whether the job completed successfully.

# Chapter 3. Basic JCL concepts

*Job control language* (JCL) is a set of statements that you code to tell the z/OS operating system about the work you want it to perform. Although this set of statements is quite large, most jobs can be run using a very small subset. Learn about essential and most frequently used JCL statements and parameters, as well as coding techniques.

JCL statements tell z/OS where to find the appropriate input, how to process that input (that is, what program or programs to run), and what to do with the resulting output.

All jobs use three main types of JCL statements:
- One *JOB statement* to identify the unit of work the operating system is to perform
- One or more *EXEC statements*, depending on the number of job steps within the job
- One or more *DD statements* to identify the input and output data sets

## JCL statements: What does the JOB statement do?

The JOB statement is the first control statement in a job. It marks the beginning of a job and also specifies the name of the job.

The JOB statement also might provide details and parameters that apply to all job steps within the job, such as accounting information and conditions for job termination. It also may contain any comments that help describe the statement.

This JCL example contains one JOB statement:

```
//JOBNUM1 JOB  504,SMITH  PAYROLL
//STEP1   EXEC PGM=PROGRAM1
//DD1     DD   DSN=HLQ.OUTPUT
//
```

- The name field contains the job name “JOBNUM1”. In every JOB statement, the name field contains a one- through eight-character name that identifies the job so that other JCL statements or the operating system can refer to it. Be sure to assign a unique name for each job.
- The parameter field defines information that applies to the entire job, contains an accounting number (504) and the programmer’s name (SMITH). These parameters are positional and must appear in the order shown.
- The comment field contains PAYROLL.

The end of a job is indicated by a null statement, which consists of only two forward slashes (//), or is marked by the beginning of another JOB statement. In this sample, JOBNUM1 ends with a null statement.

### JCL JOB statements: Positional and frequently used parameters

In addition to the two positional parameters (job accounting information and programmer name), the JOB statement also may contain over 20 keyword parameters. But you’ll most often use only this handful.

## Positional parameters

JOB statements have two positional parameters that apply to the entire job:

**Job accounting information**
The value that you code for job accounting information depends on the guidelines set at your company. The value is usually a number that identifies a department or person to whom processor time is billed.

Job accounting information may consist of multiple pieces of information, not just a single value as shown in this example.

**Programmer name**
The programmer name identifies the person or group responsible for a job. The programmer's name is not a mandatory part of the JOB statement unless your company has made it so.

## Keyword parameters

As with the positional parameters for the JOB statement, keyword parameter values apply for the entire job. The JOB statement has over twenty different keyword parameters, but you are most likely to use only these few:

**CLASS**
Use the CLASS parameter if your company uses classes to group jobs. Grouping jobs helps to:

- Achieve a balance between different types of jobs. A good balance of job class assignments helps to make the most efficient use possible of the system.
- Avoid contention between jobs that use the same resources.

Because jobs classes are site-specific, you have to check with your operations department to determine which job classes are available for use.

**TIME** Use the TIME parameter to specify the maximum amount of time that a job may use the processor or to find out through messages how much processor time a job used. Using the TIME parameter prevents an error in your program from causing it to run longer than necessary.

You can use the TIME parameter on a JOB statement to decrease the amount of processor time available to a job or job step below the default value. You cannot use the TIME parameter on a JOB statement to increase the amount of time available.

**MSGLEVEL**
The MSGLEVEL parameter controls how the JCL, allocation messages, and termination messages are printed in the job's output listing (SYSOUT).

The MSGLEVEL parameter value consists of two subparameters:

**statement**
The statement subparameter indicates which job control statements the system is to print on the job log.

**messages**
The messages subparameter indicates which messages the system is to print on the job log.

**MSGCLASS**
You can use the MSGCLASS keyword parameter to assign an output class

for your output listing (SYSOUT). Output classes are defined by the installation to designate unit record devices, such as printers.

Both the MSGLEVEL and MSGCLASS parameters have default settings, depending on your company's guidelines. The operating system uses the default setting if you omit one or both of the keyword parameters from the JOB statement. In this case, you would code these parameters only if you want to have a different message level or message class than the preset values.

The MSGLEVEL subparameters have only IBM-supplied values that you may specify, but output class assignments and the default settings for both parameters depend on values your company chooses to use.

## JCL statements: What does the EXEC statement do?

The EXEC statement marks the beginning of a step within a job, and specifies the name of a program or cataloged procedure to be run.

Procedures are named collections of partial JCL, usually one or more EXEC statements and data definition (DD) statements, that perform frequently used functions such as sorting data. Procedures are often called *procs*.

Programs and cataloged procedures are stored in specific data sets, which are called program or procedure libraries, respectively.

This JCL example contains only one EXEC statement (and therefore, only one job step).

```
//JOBNUM1 JOB  504,SMITH  PAYROLL
//STEP1   EXEC PGM=PROGRAM1
//DD1     DD   DSN=HLQ.INPUT
//
```

In this EXEC statement:

- The name field contains the step name "STEP1". A step name is a one- through eight-character name that identifies the job step so that other JCL statements or the operating system can refer to it.
- The parameter field contains the positional parameter PGM, which identifies the program to be run (PROGRAM1).

Also, the sample includes a DD statement that identifies the input data set, HLQ.INPUT, for the program. The JCL for a job step often contains several associated DD statements that define the program or procedure uses for input or output.

The end of a job step is indicated by a null statement, which consists of only two forward slashes (//); by another EXEC statement; or by another JOB statement. In this sample, STEP1 ends with a null statement that immediately follows the DD statement DD1.

### JCL EXEC statements: Positional and frequently used parameters

In addition to the two positional parameters (PGM and PROC), the EXEC statement also may contain about a dozen keyword parameters. But you'll most often use only this handful.

## Positional parameters

An EXEC statement must contain one of these positional parameters: PGM, PROC, or procedure name.

**PGM** The PGM parameter identifies the program the system is to run.

z/OS includes a number of programs, called *utilities*, which are useful in batch processing. These programs provide many small, obvious, and useful functions. For example, z/OS has a utility program named IEBGENER to copy data.

Customer sites often add their own customer-written utility programs (although most users refrain from naming them utilities) and many of these are widely shared by the user community. Independent software vendors also provide many similar products (for a fee).

**PROC or** *procedure name*

The PROC parameter or *procedure name* identifies the cataloged or in-stream procedure the system is to run.

If you omit the PGM or PROC parameter, z/OS automatically assumes that you are specifying a procedure that you want to run.

The following code illustrates the three ways to correctly code this positional parameter.

```
//STEP1 EXEC PGM=program-name
//STEP1 EXEC PROC=procedure-name
//STEP1 EXEC procedure-name
```

Lowercase text is variable text that you provide. You may code only one of these formats on a single EXEC statement.

## Keyword parameters

In addition to the positional parameter indicating the program or procedure to run, the EXEC statement also may contain keyword parameters. If you code one of these keyword parameters on the EXEC statement, the keyword parameter value will apply only to that step. You are most likely to use only these few EXEC keywords:

**COND**

In a multi-step job, use the COND parameter to specify the conditions that allow the system to bypass a step by testing return codes from any or all previous steps. You can code up to eight comparisons. If any comparison is true, the system bypasses the step.

As an alternative, you may use the IF/THEN/ELSE statement, which you might find easier to code than COND parameter conditions.

**PARM** 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.

**REGION**

Use the REGION parameter to override the default amount of storage space (in kilobytes or megabytes) that the system allocates to a particular job or job step.

You may code the REGION parameter on the JOB statement and the EXEC statement. If REGION appears on both statements, the value on the JOB statement overrides that on the EXEC statement.

## JCL EXEC statements: What are JCL procedures?

Some programs and tasks require a larger amount of JCL than a user can easily enter. JCL for these functions can be kept in procedure libraries.

A procedure library member contains only part of the JCL for a given task--usually the fixed, unchanging part of JCL. The user of the procedure supplies the variable part of the JCL for a specific job. In other words, a JCL procedure is like a *macro*.

Such a procedure is sometimes known as a *cataloged procedure*. A cataloged procedure is not related to the system catalog; rather, the name is a carryover from another operating system.

The following code shows an example of a JCL procedure (commonly called a *proc*).

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

In this example procedure:

- The **PROC** and **PEND** parameters identify the beginning and end of the JCL procedure. **PROC** is preceded by a label or name; in this example, the name is `MYPROC`. The **PROC** and **PEND** parameters are unique to procedures.
- The first (and only) step in the procedure is the EXEC statement named `MYSORT`, which identifies the program SORT as the program to be run.
- The `SORTIN DD` statement identifies the input data set containing data to be sorted. The variable `&SORTDSN` represents the input data set; its actual value will be determined by the JCL that calls in and uses this procedure.
- The `SORTOUT DD` statement identifies the output data set where SORT is to place the sorted input; the `SYSOUT=*` parameter identifies a default data set.
- The `SYSOUT DD` statement identifies a default data set in which the system is to place messages issued during the SORT program's processing.

If you coded JCL to use this example proc, your JCL might look like this:

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

In this example job:

- Your JCL begins with a JOB statement that names your job `MYJOB`.
- The first step in `MYJOB`, named `STEP1`, does two things:

  - Identifies MYPROC as the JCL procedure to be run.
  - Specifies the value SORTDSN=ZPROF.AREA.CODES as the value for the variable &SORTDSN.
- The SYSIN DD statement provides instream instructions for the SORT program to use on the input data set, ZPROF.AREA.CODES.

The previous example shows how to change the value of one part of a JCL procedure; in this case, replacing the variable &SORTDSN with a real data set name ZPROF.AREA.CODES. In some cases, you might need to override an entire statement within a JCL procedure. To do so, you code a JCL PROC override statement, which is shown in this example:

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

The MYSORT.SORTOUT DD statement is a procedure override that redefines the output data set as a newly created data set rather than a default data set. The *stepname.ddname* format tells the system to override the corresponding DD statement in the named step, within the JCL procedure. Using an override allows you to tailor the procedure for your needs, without changing its function for other users.

## JCL EXEC statements: How z/OS finds the program or procedure

To successfully run the program or procedure that you specify on a JCL EXEC statement, z/OS has to search for and find that program or procedure. Using JOBLIB, STEPLIB, or JCLLIB statements can reduce search time.

Where z/OS searches depends on what you specify in your JCL:

- When you code the PGM parameter, z/OS looks for an application program, and will automatically search standard system program libraries, such as SYS1.LINKLIB, which contains IBM-supplied programs. If the program you want to run resides in a private program library, you must specify either a JOBLIB DD statement or a STEPLIB DD statement for z/OS to successfully locate the program.
  - When you use a JOBLIB DD statement, insert the JOBLIB DD statement in the job before the first EXEC statement in the job. When you submit the job, z/OS will search any private libraries specified on that JOBLIB DD statement before searching system libraries. z/OS repeats that search order for any programs called within the job.

    For a multi-step job, using the JOBLIB DD statement is most efficient when most of the programs reside in private libraries.

  – When you use a STEPLIB DD statement, you may place it anywhere within a job step but it typically appears after the EXEC statement. When you submit the job, z/OS will search the private libraries specified on that STEPLIB DD statement, but only for the one step.

    For a multi-step job, using the STEPLIB DD statement is most efficient when most of the programs reside in system, rather than private, libraries.

- When you code the PROC parameter or you omit the PGM or PROC parameter, z/OS looks for a procedure and will automatically search standard system procedure libraries, such as SYS1.PROCLIB. If the procedure you want to run resides in a private library, you must specify the JCLLIB statement for z/OS to successfully locate the procedure.

  If a job does not specify a procedure library, the system retrieves all cataloged procedures called by EXEC statements from the procedure libraries defined by the installation for the job's job class. To direct the system to search another procedure library, or to limit the procedure libraries it searches, code a JCLLIB statement that identifies one or more libraries. On a JCLLIB statement, you may list system libraries, installation-defined libraries, or private libraries. The system searches the libraries in the order in which they are specified on JCLLIB.

## What are the standard system libraries?

z/OS has many standard system libraries; here are a few that you will come across often.

z/OS standard system libraries include:

**SYS1.PROCLIB**
This library contains JCL procedures distributed with z/OS. In practice, there are many other JCL procedure libraries (supplied with various program products) concatenated with it.

**SYS1.PARMLIB**
This library contains control parameters for z/OS and for some program products. In practice, there may be other libraries concatenated with it.

**SYS1.LINKLIB**
This library contains many of the basic execution modules of the system. In practice, it is one of a large number of execution libraries that are concatenated.

**SYS1.LPALIB**
This library contains system execution modules that are loaded into the link pack area when the system is initialized. There may be several other libraries concatenated with it. Programs stored here are available to other address spaces.

**SYS1.NUCLEUS**
This library contains the basic supervisor ("kernel") modules of z/OS.

**SYS1.SVCLIB**
This library contains operating system routines known as supervisor calls (SVCs).

These libraries are in standard PDS format and are found on the system disk volumes.

## JCL statements: What does the DD statement do?

Data definition (DD) statements define the data sets that a program or procedure uses when it runs. You must code one DD statement for each data set that is used or created within a job step.

The order of DD statements within a job step is not usually significant.

This JCL example illustrates the format of a DD statement:

```
//PAY  DD  DSN=HLQ.PAYDS,DISP=NEW  VENDOR PAYROLL
```

- The name field contains a one- through eight-character name, known as a *ddname*, that identifies the DD statement so that other JCL statements, programs, procedures, or the operating system can refer to it. The ddname of this DD statement is `PAY`.
- The parameter field contains only two keyword parameters:
  - **DSN**, which is an accepted abbreviation for the parameter **DSNAME**, which identifies the real name of a data set.
  - **DISP**, which identifies the data set HLQ.PAYDS as a new data set; that is, one the system is to create when this job is submitted for processing.
- The comment field contains the phrase `VENDOR PAYROLL`.

A DD statement describes a data set extensively, and can include the following information:

- The name that the program uses to refer to the data set, known as the *ddname*
- The actual name of the data set and its location
- Physical characteristics of the data set, such as record format
- The initial and final status of the data set, known as its *disposition*

You also can use DD statements to request I/O devices or specify storage allocation for new data sets.

### JCL DD statements: Advantages of using symbolic names

When you use symbolic names to identify individual DD statements in JCL, you can improve the reusability of programs. Here’s how it works.

When you use symbolic names for DD statements in JCL, you can improve the reusability of programs by changing data set information without having to recompile the programs that access the data set.

For example, suppose that your company uses a payroll program to record its employees’ pay. Each month, you have to run the program to update pay records, using a different data set each time. In the program code, the program uses the name “PAY” to refer to the data set. Instead of changing the program each month to use a different data set name, you can use “PAY” as the label in the name field (or *ddname*) for the JCL DD statement that identifies the payroll data set to be used. So instead of changing and recompiling the payroll program, you make only minor changes to your JCL.

The first time you run the program, your JCL looks like the JCL in Figure 1 on page 41.

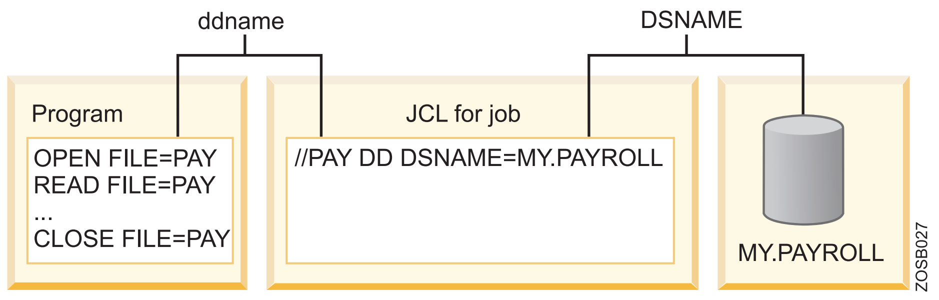


*Figure 1. Symbolic file name: Payroll program and MY.PAYROLL data set*

The next time, when the payroll data set changes to DIV1.PAYROLL, you change only the DSN parameter value on the DD statement, and the payroll program can continue to use the same ddname (PAY). Your JCL now looks like the JCL in Figure 2.

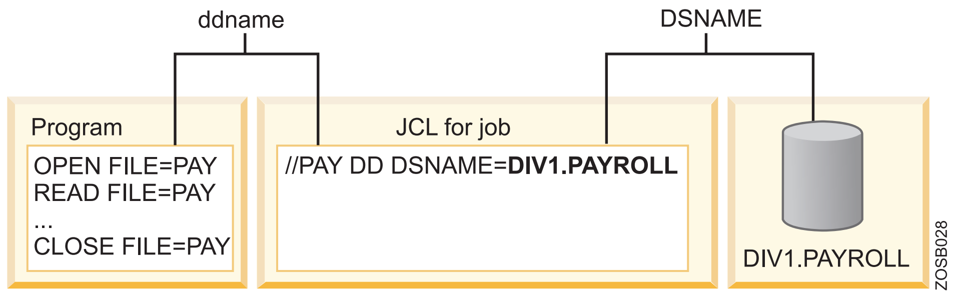


*Figure 2. Symbolic file name: Payroll program and new DIV1.PAYROLL data set*

The ddname connects a program reference to a data set to the data set description (that is, the DD statement) in the JCL. So the ddname "PAY" is a symbolic name that the payroll program uses, and the DSNAME parameter in the JCL identifies the real name of the data set.

The use of symbolic file names is another defining characteristic of the z/OS operating system. It applies a naming redirection between a data set-related name used in a program and the actual data set used during execution of that program. This redirection allows the program to be used to process different input data sets simply by changing the DSNAME in the JCL. This ability becomes significant for large commercial applications that might use dozens of data sets in a single execution of the program.

The format for coding program references depends on the language in which the program is written. For example:

**In a COBOL program**

The ASSIGN clause in the Environment Division identifies the DDNAME that must be used in the DD statement.

```
SELECT FILEIN ASSIGN TO PAY...
```

**In an Assembler program**

The DDNAME is indicated as a keyword parameter of the DCB macro.

```
FILEIN DCB DDNAME=PAY,...
```

IBM reserves the following list of DD names for optional, special-function DD statements:

**JOBLIB**
: A JOBLIB DD statement, placed just after a JOB statement, specifies a library that should be searched first for the programs executed by this job.

**STEPLIB**
: A STEPLIB DD statement, placed just after an EXEC statement, specifies a library that should be searched first for the program executed by the EXEC statement. A STEPLIB overrides a JOBLIB if both are used.

**JOBCAT and STEPCAT**
: JOBCAT and STEPCAT are used to specify private catalogs, but these are rarely used (the most recent z/OS releases no longer support private catalogs). Nevertheless, these DD names should be treated as reserved names.

**SYSABEND, SYSUDUMP, SYSMDUMP and CEEDUMP**
: The SYSABEND, SYSUDUMP, SYSMDUMP, and CEEDUMP DD statements are used for various types of memory dumps that are generated when a program abnormally ends.

## JCL DD statement: ddnames that are reserved for specific uses

JCL programmers can use almost any name as a label for a DD statements, but IBM reserves this list of ddnames for special-function DD statements.

Use the following special ddnames only when you want to use the facilities these names represent to the system:

**JOBLIB**
: A JOBLIB DD statement, placed just after a JOB statement, specifies a library that should be searched first for the programs executed by this job.

**STEPLIB**
: A STEPLIB DD statement, placed just after an EXEC statement, specifies a library that should be searched first for the program executed by the EXEC statement. A STEPLIB overrides a JOBLIB if both are used.

**SYSCHK**
: The SYSCHK DD statement defines a checkpoint data set that the system is to write during execution of a processing program.

**SYSCKEOV**
: The SYSCKEOV DD statement defines a checkpoint data set for checkpoint records from the checkpoint at end-of-volume (EOV) facility. The checkpoint at EOV facility is invoked by a DD CHKPT parameter.

**SYSIN**
: By convention, people often use a SYSIN DD statement to begin an in-stream data set.

**SYSABEND, SYSUDUMP, SYSMDUMP and CEEDUMP**
: The SYSABEND, SYSUDUMP, SYSMDUMP, and CEEDUMP DD statements are used for various types of memory dumps that are generated when a program abnormally ends.

Another set of reserved ddnames, JOBCAT and STEPCAT, are used to specify private catalogs, but these are rarely used (the most recent z/OS releases no longer support private catalogs). Nevertheless, these DD names should be treated as reserved names.

The following ddnames have special meaning to JES2; do not use them on a DD statement in a JES2 system.

JESJCL
JESJCLIN
JESMSGLG
JESYSMSG

The following ddnames have special meaning to JES3; do not use them on a DD statement in a JES3 system.

J3JBINFO
J3SCINFO
J3STINFO
JCBIN
JCBLOCK
JCBTAB
JESInnnn
JESJCL
JESJCLIN
JESMSGLG
JESYSMSG
JOURNAL
JS3CATLG
JST
STCINRDR
TSOINRDR

## JCL DD statements: Positional and frequently used parameters

In addition to its one positional parameter, the DD statement has well over 50 keyword parameters. But you'll most often use or encounter only this subset of these parameters.

A DD statement may contain only one positional parameter that must precede all keyword parameters. The following list describes the positional parameter values you may code for the DD statement.

*** (an asterisk)**
The * parameter value begins an in-stream data set.

**DATA** The DATA parameter value begins an in-stream data set that may contain statements with // in columns 1 and 2.

**DUMMY**
The DUMMY parameter value tells z/OS not to perform any input, output, or disposition processing on the data set. Use the DUMMY parameter value when you are not providing input or do not want the output for a data set, or when testing a program.

**DYNAM**
The DYNAM parameter value is supported only to provide compatibility with previous versions of the z/OS operating system.

Even when you become a JCL expert, you probably will use only a handful of DD statement keywords frequently. Which keyword parameters you use depends on several factors, including whether you want to use an existing or create a new data set, what type of data set you are using or creating, and whether your company uses SMS to manage data sets. Here are the keyword parameters that you are most likely to use or see in existing JCL:

**DCB** The DCB parameter defines the format type, length of records, and block size for a new data set.

**DISP** The data set disposition parameter, DISP, indicates:
- The current status of the data set, and whether the job requires exclusive use of it
- How z/OS is to handle the data set after the job step ends either normally or abnormally.

The DISP parameters of DD statements help to prevent unwanted simultaneous access to data sets. In other words, the DISP parameter helps manage the integrity of data sets.

**DSNAME or DSN**
The DSNAME parameter, or its abbreviation DSN, specifies the actual name of the data set. z/OS uses this name to locate the data set in storage. The DSNAME or DSN keyword must be specified for an existing data set.

**LABEL**
The LABEL parameter specifies specific information about a tape or direct access data set, including:
- The type and contents of the label or labels for the data set.
- If a password is required to access the data set.
- If the system is to open the data set only for input or output.
- The expiration date or retention period for the data set.

**SPACE**
The SPACE parameter allocates storage for a new data set on a direct access storage device. The allocation of a data set means either or both of two things:
- To set aside (or create) space for a new data set on a disk.
- To establish a logical link between a job step and any data set.

**SYSOUT**
The SYSOUT parameter specifies a system output data set and its output class. A system output (SYSOUT) data set contains the job output that is to be printed. This job output is also known is the output stream. Unlike a permanent data set, a sysout data set is disk space that z/OS uses as buffer storage for processing output.

**UNIT** When you are defining a new data set, you may use the UNIT parameter to tell z/OS to place the data set on:
- A specific device, by specifying a hardware address.
- A certain type or group of devices; examples of device types are 3390 for a disk or 3590 for tape.
- The same device as another data set.

**VOLUME or VOL**
When you are defining a new data set, you may use the VOLUME parameter, or its abbreviation, VOL, to tell z/OS to place the data set on a specific volume. You can request:

A private volume
A specific volume by serial number
The same volume that another data set uses

## JCL DD statements: Use different parameters for SMS data sets

The Storage Management Subsystem (SMS) automates the use of storage for data sets. The use of SMS, which is optional, affects the JCL you code on DD statements for SMS-managed data sets.

In a z/OS system, data management involves tasks that include the allocation, placement, backup, recall and deletion of data sets. These activities can be done either manually or through the use of automated processes. Several of these activities can be done manually through the use of JCL.

Part of the z/OS storage management product, the Storage Management Subsystem (SMS), automates the use of storage for data sets. With SMS, the z/OS system programmer or storage administrator may, for example, create model data definitions for typical data sets, so that SMS automatically assigns attributes to data sets when they are created. The data sets allocated through SMS are called *system-managed* data sets or *SMS-managed* data sets.

Before you start using JCL to work with data sets, you should know whether your company uses SMS for the data sets you will be creating or using. If your company does use SMS, you do not need to code certain DD statement keywords in your JCL.

For example, suppose you want to create a new data set named DATA.LIST. If SMS is active, you could use JCL like this:

```
//NEWDS DD  DSN=HLQ.DATA.LIST,
//          DISP=(NEW,KEEP),
//          DATACLAS=DSCLAS01,
//          STORCLAS=STRCLS20
```

In this case, z/OS can use characteristics from predefined data and storage classes when it creates the DATA.LIST data set for you.

If SMS is not active or not in use, you need to manually specify the space requirements and storage location for the new data set, and your JCL would look like this or something even more complicated:

```
//NEWDS DD  DSN=HLQ.DATA.LIST,
//          DISP=(NEW,KEEP),
//          SPACE=(CYL,(1,1)),
//          UNIT=SYSDA,
//          VOL=SER=SHARED
```

## JCL DD statements: Identify program libraries with JOBLIB or STEPLIB

To successfully run the program that you specify on a JCL EXEC statement, z/OS has to search for and find that program. Using JOBLIB or STEPLIB DD statements can reduce search time.

When you code the PGM parameter, z/OS looks for a program, and will automatically search standard system program libraries, such as SYS1.LINKLIB, which contains IBM-supplied programs. If the program you want to run resides in a private program library, you must specify either a JOBLIB DD statement or a STEPLIB DD statement for z/OS to successfully locate the program.

Although both the JOBLIB DD statement and the STEPLIB DD statement identify one or more private libraries as the location of a specified program, they dictate different search behaviors for z/OS:

- JOBLIB tells z/OS to search the private libraries for each step in the job
- STEPLIB tells z/OS to search the private libraries only for one step

In both cases, z/OS searches system libraries only if it does not find the program first in the private libraries on the JOBLIB or STEPLIB DD statement.

To decide which DD statement is most efficient for you to use, you need to know the location of the programs to be run. If most are in the same private library, for example, the JOBLIB DD statement is probably the best choice. If only a few programs are in private libraries, the STEPLIB DD statement is probably the most efficient.

In this example, a JOBLIB DD statement names the private library in which PROGRAM1 resides:

```
//JOBNUM1 JOB  504,SMITH  PAYROLL
//JOBLIB  DD   DSN=MY.LIBRARY,DISP=SHR//STEP1   EXEC PGM=PROGRAM1
//DD1     DD   DSN=HLQ.INPUT
//
```

In this example, a STEPLIB DD statement names the private library in which PROGRAM1 resides:

```
//JOBNUM1 JOB  504,SMITH  PAYROLL
//STEP1   EXEC PGM=PROGRAM1
//STEPLIB  DD   DSN=MY.LIBRARY,DISP=SHR//DD1     DD   DSN=HLQ.INPUT
//
```

If you code both a JOBLIB DD and STEPLIB DD statement in the same job, the STEPLIB DD statement overrides the JOBLIB statement only for the one step. For that step only, the system ignores JOBLIB and first searches the private libraries specified on the STEPLIB DD statement. If the system does not find the program in the private libraries, it then searches the system libraries. Then, if z/OS still has not found the program, the system abnormally ends the job step.

# Chapter 4. Coding your own JCL

The easiest way to learn JCL is to use some that's already been written, which is why we started the reusable JCL collection. If you need to complete tasks that are not yet represented in the reusable collection, you can borrow someone else's JCL and modify it to suit your task. Understanding general syntax rules and learning about frequently used parameters will help you learn to correctly modify or code your own JCL.

## JCL exercise: Creating and submitting a job

This exercise takes you through the process of creating a data set member for JCL, coding JCL (using a predefined sample), submitting the job, and viewing the job output. Even if you do not have access to a z/OS system to accomplish these tasks, reading through the instructions in this exercise will help you understand how to use JCL to create jobs, submit those jobs and interpret the output.

### Before you begin

**Before you begin:**

Before creating any job, you need to know the following:

- ***Installation conventions.*** Every job must include special accounting and identifying information, which varies from one z/OS installation to another. To submit your JCL successfully, you need to find out the conventions that are followed at your company.

  Use the worksheet in "Coding JCL: Collecting company-specific information" on page 54 for documenting this information. You may need to ask a mentor or co-worker to help you identify the conventions indicated in the worksheet.
- ***How to allocate and edit a data set.*** During the exercise, you will be entering JCL statements into a data set so that you can subsequently modify and reuse them as required. Therefore, you must know how to use ISPF panels (or an equivalent technique) to allocate and edit the data set according to the specific requirements of your z/OS system.

  **Note:**
  1. A common programming practice is to use JCL as the last qualifier in the name of any data set that is to contain JCL; for example, `*userid*.SORT.JCL`
  2. A data set that contains JCL must have a fixed-block format (RECFM=FB) with a logical record length of 80 (LRECL=80).
- ***The job to be done and the resources needed.*** You need to determine what work you plan to have z/OS perform:
  - What inputs (resources) you will need and where they are located
  - What program you plan to use.
  - Where the output, if any, should go. (When the job completes, you will either dispose of the output or hold it for later printing or for viewing.)

  The job for this exercise is to sort a simple file and list the contents alphabetically. Decisions about inputs, outputs, and processing have already been made for you, so all you will have to do is to copy the example code provided.

- ***How to view and understand held output.*** Running your job will produce three types of held output:
  - System messages from JES and z/OS
  - Your JCL code with procedures expanded, overrides applied, and symbolics resolved.
  - Output as requested by the JCL code

  Held output may be viewed, printed, or purged. Sample output shows you how the output from the exercise should look and explains what each part of the output means.

## Procedure

1. Allocate a data set to contain your JCL. Use ISPF (or equivalent function) to allocate a data set named *userid*.SORT.JCL (where *userid* is your TSO user ID) with a fixed-block format (RECFM=FB) and a logical record length of 80 (LRECL=80).
2. Edit the JCL data set and add the necessary JCL. Use ISPF (or equivalent function) to edit the data set that you just allocated.
   a. Enter the following JCL statements into the data set. Note that all JCL statements start with the special identifier //.

```
//SORT JOB 'accounting_data',  1
//  'user_name',  2
//  NOTIFY=&SYSUID,  3
//  MSGCLASS=message_class,  4
//  MSGLEVEL=(1,1),  5
//  CLASS=n,  6
//STEP1    EXEC  PGM=IEFBR14  7
//SORTIN   DD  *  8
NEPTUNE  9
PLUTO
EARTH
VENUS
MERCURY
MARS
URANUS
SATURN
JUPITER
/*  10
//SORTOUT  DD SYSOUT=*  11
/*  12
```

   In the JCL code above:

   **1** Replace *accounting_data* with the appropriate security classification and identification information, according to the information you filled in on "Coding JCL: Collecting company-specific information" on page 54.

   **2** Replace *user_name* with your name.

   **3** The NOTIFY parameter tells the system where to send "job complete" information. &SYSUID tells the system to automatically insert your user ID here, so the information will be sent to you.

   **4** The MSGCLASS parameter tells the system what to do with messages the system sends you as it processes your job; for example, use a held output class to allow reviewing the messages later. Replace *message_class* with the appropriate message class value.

   **5** MSGLEVEL=(1,1) tells the system to reproduce this JCL code in the output, and to include allocation messages.

**6** CLASS=*n* indicates the system resource requirements for the job.

**7** The EXEC statement invokes the program IEFBR14 and identifies the first (and only) job step in this job. You are arbitrarily naming it STEP1. All of the control statements that follow the EXEC statement are part of this job step.

IEFBR14 is the name of a program within your z/OS system. It does not actually process any data, but it enables you to run this job as a test to verify the JCL statements, and to create the input data. Later in the exercise you will replace IEFBR14 with the name of another program that sorts data.

**8** SORTIN is the name you have given the DD statement that describes the input data.

**9** NEPTUNE through JUPITER are the items to be sorted. This method of providing data to the program is referred to as *in-stream* data, an alternative to providing the input in a separate allocated data set.

**10** The symbols /* indicates the end of the input data stream.

**11** SORTOUT is the name you have given the DD statement that describes where the output from running the job will be placed. In this example, SYSOUT=* specifies that the output data will be directed to the SYSOUT device defined in the MSGCLASS statement.

**12** The symbols /* (optional) denote the end of the job.

3. Submit the JCL to the system as a job. When you have finished entering the JCL into the data set, submit the job by entering the SUBMIT command from the ISPF EDIT command line, the TSO/E command line, or following a READY mode message. Each of these methods is shown below.
   - ***ISPF EDIT command line:***

```
 EDIT ---- userid.SORT.JCL -------------------------- LINE 00000000 COL 001 080
 COMMAND ===> SUBMIT                                            SCROLL ===> CSR
******************************** TOP OF DATA  ********************************
//userid   JOB 'accounting data',
                      .
                      .
                      .
```

   - ***TSO/E command line:***

```
------------------------  TSO COMMAND PROCESSOR  ----------------------------
ENTER TSO COMMAND OR CLIST BELOW:


===> SUBMIT 'userid.SORT.JCL'



ENTER SESSION MANAGER MODE ===> NO     (YES or NO)
```

   - ***After READY mode message:***

```
    .
    .
    .
 READY
SUBMIT 'userid.SORT.JCL'
```

**Note:** When entering the command from the TSO command line or after a READY message, you must surround the data set name with single quotation marks if you include your user ID. However, you can also enter the command without specifying your user ID and without using single quotation marks, as shown below:

```
SUBMIT SORT.JCL
```

When you do not specify the user ID and do not include single quotes, the system automatically inserts your user ID before the data set name. (The insertion of the user ID is for the duration of the current job; it is not a permanent change to the data set name.)

After entering the command, you should receive the following message indicating that your job was submitted successfully:

- ***When submitted from the ISPF EDIT command line:***

```
 EDIT ---- userid.SORT.JCL -------------------------- LINE 00000000 COL 001 080
 COMMAND ===> SUBMIT                                            SCROLL ===> CSR
********************************* TOP OF DATA  ********************************
//userid   JOB 'accounting data',
                       .
                       .
                       .
JOB jobname(jobnumber) SUBMITTED
***
```

- ***When submitted from the TSO command line:***

```
-------------------------  TSO COMMAND PROCESSOR  -----------
ENTER TSO COMMAND OR CLIST BELOW:

===> SUBMIT 'userid.SORT.JCL'






ENTER SESSION MANAGER MODE ===> NO     (YES or NO)
JOB jobname(jobnumber) SUBMITTED
***
```

- ***When submitted after READY mode message:***

```
     .
     .
     .
    READY
   SUBMIT 'userid.SORT.JCL'
     .
     .
     .
JOB jobname(jobnumber) SUBMITTED
***
     .
     .
    READY
```

When the job ends, you will receive a message indicating one of three conditions: job successful, JCL error, or program abend. If the message indicates the error or abend condition, review steps 2 and 3 of this exercise to make sure that you followed the instructions exactly, then resubmit the job.

If the job fails again, consult the appropriate messages as indicated below:

- ***If the message begins with HASP***, the job was failed by JES2.
- ***If the message begins with IAT***, the job was failed by JES3.

4. View and understand the output from the job. Use your installation's viewing facility (for example, SDSF) to view the output and determine whether the job completed successfully.

   If the job is on hold in the held queue, consider printing it for a record of the job activity.

```
1                    J E S 2  J O B  L O G  --  S Y S T E M  A Q T S  --  N O D E  P L P S C
0
 15.21.28 JOB17653  IRR010I  USERID userid   IS ASSIGNED TO THIS JOB.
 15.21.28 JOB17653  ICH70001I userid   LAST ACCESS AT 15:21:28 ON WEDNESDAY, OCTOBER 13, 2006
 15.21.28 JOB17653  $HASP373 SORT  STARTED - INIT    9 - CLASS 5 - SYS AQTS
 15.21.28 JOB17653  IEF403I SORT - STARTED - TIME=15.21.28                                         1
 15.21.28 JOB17653  - ==========================================================================================================
 15.21.28 JOB17653  -                                    REGION        --- STEP TIMINGS ---                  ----PAGING COUNTS----
 15.21.28 JOB17653  - STEPNAME PROCSTEP PGMNAME     CC     USED      CPU TIME  ELAPSED TIME    EXCP     SERV  PAGE  SWAP   VIO SWAPS
 15.21.28 JOB17653  - STEP1             IEFBR14     00       4K   00:00:00.01   00:00:00.03       1      211     0     0     0     0
 15.21.28 JOB17653  IEF404I SORT - ENDED - TIME=15.21.28
 15.21.28 JOB17653  - ==========================================================================================================
 15.21.28 JOB17653  - NAME-user_name        TOTALS: CPU TIME=  00:00:00.01  ELAPSED TIME=  00:00:00.05 SERVICE UNITS=    21
 15.21.28 JOB17653  - ==========================================================================================================
 15.21.28 JOB17653  $HASP395 SORT  ENDED
0------ JES2 JOB STATISTICS ------
-  13 OCT 1996 JOB EXECUTION DATE
-          20 CARDS READ
-          45 SYSOUT PRINT RECORDS                                                                2
-           0 SYSOUT PUNCH RECORDS
-           3 SYSOUT SPOOL KBYTES
-        0.00 MINUTES EXECUTION TIME
         1 //SORT JOB '662282,D58,9211064,S=C',                               JOB17653
           //  'user_name',
           //  NOTIFY=userid,
           //  MSGCLASS=H,                                                       00280009
           //  MSGLEVEL=(1,1),                                                   00430010             3
           //  CLASS=5                                                           00430010
         2 //STEP1    EXEC  PGM=IEFBR14
         3 //SORTIN  DD *
         4 //SORTOUT  DD SYSOUT=*
         5 //SYSIN     DD *              GENERATED STATEMENT
 ICH70001I userid   LAST ACCESS AT 15:21:28 ON WEDNESDAY, OCTOBER 13, 2006  IEF236I ALLOC. FOR SORT STEP1
 IEF237I JES2 ALLOCATED TO DATAIN
 IEF237I JES2 ALLOCATED TO SYSIN
 IEF142I SORT STEP1 - STEP WAS EXECUTED - COND CODE 0000         5
 IEF285I   userid.SORT.JOB17653.D0000101.?          SYSIN                   4
 IEF285I   userid.SORT.JOB17653.D0000103.?          SYSOUT
 IEF285I   userid.SORT.JOB17653.D0000102.?          SYSIN
 IEF373I STEP /STEP1   / START 1996286.1521
 IEF374I STEP /STEP1   / STOP  1996286.1521 CPU    0MIN 00.01SEC SRB    0MIN 00.00SEC VIRT     4K SYS   180K EXT       4K SYS    9424K
 IEF375I  JOB /SORT / START 1996286.1521
 IEF376I  JOB /SORT / STOP  1996286.1521 CPU    0MIN 00.01SEC SRB    0MIN 00.00SEC
```

*Figure 3. Output from Job Invoking IEFBR14 Program*

Figure 3 on page 51 contains an example of the held output for this exercise. Each part of this output is explained below:

- The first block of information ( **1** ), ending with the line `$HASP395 SORT ENDED`, is installation-specific and may differ on your system.
- The JES2 job statistics section ( **2** ), ending with the amount of execution time (`0.00 MINUTES EXECUTION TIME`), contains JES messages about the job.
- The next section ( **3** ) contains the JCL statements, or JCL listing, that resulted from the job.
- The final section ( **4** ), beginning with message ICH70001I, contains the system output messages resulting from processing the job.
- Within the final section, the condition code 0000 ( **5** ) tells you that the program ran successfully. You receive one condition code for each step in the job. If a condition code is non-zero, see the documentation for the specific program you invoked.

5. Make changes to your JCL. When your job has run successfully, edit the data set containing the JCL and change or add control statements as indicated below:

```
//SORT JOB 'accounting_data',
//  'user_name',
//  NOTIFY=&SYSUID,
//  MSGCLASS=H,
//  MSGLEVEL=(1,1),
//  CLASS=5
//STEP1    EXEC  PGM=SORT 1
//SYSIN   DD *    SORT     FIELDS=(1,75,CH,A) 2
/*
//SYSOUT   DD  SYSOUT=*    3
//SORTIN   DD  *
NEPTUNE
PLUTO
EARTH
VENUS
MERCURY
MARS
URANUS
SATURN
JUPITER
/*
//SORTOUT  DD SYSOUT=*
/*
```

**1** Replace the program name with the name of your sort program. In this job, SORT will sort the input data identified by the SORTIN DD statement.

**2** Add the SYSIN control statement. SYSIN specifies how you want the sort to be done. In this case, you are indicating that you want to sort the fields from column 1 to column 75 as characters in ascending sequence.

**3** Add the SYSOUT control statement. SYSOUT specifies the data set to which SORT will write its messages. A SYSOUT data set is a system-handled output data set. This data set is placed temporarily on direct access storage. Later, the system prints it or sends it to a specified location.

When you have finished entering the JCL into the data set, submit the job.

6. View and understand your final output. Figure 4 on page 53 shows an example of the held output for the completed exercise. Each part of this output is

explained below:

```
1                   J E S 2  J O B  L O G  --  S Y S T E M  A Q T S  --  N O D E  P L P S C
0
 13.40.27 JOB06572  IRR010I  USERID 'userid' IS ASSIGNED TO THIS JOB.
 13.40.27 JOB06572  ICH70001I 'userid' LAST ACCESS AT 13:39:20 ON MONDAY, NOVEMBER 15, 2006
 13.40.27 JOB06572  $HASP373 SORT     STARTED - INIT    9 - CLASS 5 - SYS AQTS
 13.40.27 JOB06572  IEF403I SORT - STARTED - TIME=13.40.27                                                 1
 13.40.28 JOB06572  - ==================================================================================================================
 13.40.28 JOB06572  -                             REGION        --- STEP TIMINGS ---                   ----PAGING COUNTS----
 13.40.28 JOB06572  - STEPNAME PROCSTEP PGMNAME     CC     USED      CPU TIME  ELAPSED TIME    EXCP     SERV  PAGE  SWAP   VIO SWAPS
 13.40.28 JOB06572  - STEP1             SORT        00     576K   00:00:00.03   00:00:00.15      20     1614     0     0     0     0
 13.40.28 JOB06572  IEF404I SORT - ENDED - TIME=13.40.28
 13.40.28 JOB06572  - ==================================================================================================================
 13.40.28 JOB06572  - NAME-'user name'          TOTALS: CPU TIME=  00:00:00.03  ELAPSED TIME=  00:00:00.16 SERVICE UNITS=   1614
 13.40.28 JOB06572  - ==================================================================================================================
 13.40.28 JOB06572  $HASP395 SORT     ENDED
0------ JES2 JOB STATISTICS ------
-  15 NOV 1996 JOB EXECUTION DATE
-           25 CARDS READ
-           81 SYSOUT PRINT RECORDS                                                                      2
-            0 SYSOUT PUNCH RECORDS
-            4 SYSOUT SPOOL KBYTES
-         0.00 MINUTES EXECUTION TIME
         1 //SORT JOB 'accounting data',                                      JOB06572
           //  'user_name',
           //  NOTIFY=userid,
           //  MSGCLASS=H,
           //  MSGLEVEL=(1,1),
           //  CLASS=5                                                                                   3
         2 //STEP1    EXEC  PGM=SORT
         3 //SYSIN   DD *
         4 //SYSOUT   DD  SYSOUT=*
         5 //SORTIN   DD  *
         6 //SORTOUT  DD SYSOUT=*
           /*
 ICH70001I 'userid' LAST ACCESS AT 13:39:20 ON MONDAY, NOVEMBER 15, 2006
 IEF236I ALLOC. FOR SORT STEP1
 IEF237I JES2 ALLOCATED TO SYSIN
 IEF237I JES2 ALLOCATED TO SYSOUT
 IEF237I JES2 ALLOCATED TO SORTIN
 IEF237I JES2 ALLOCATED TO SORTOUT
 IEF142I SORT STEP1 - STEP WAS EXECUTED - COND CODE 0000          5
 IEF285I   userid.SORT.JOB06572.D0000101.?             SYSIN
 IEF285I   userid.SORT.JOB06572.D0000103.?             SYSOUT
 IEF285I   userid.SORT.JOB06572.D0000102.?             SYSIN
 IEF285I   userid.SORT.JOB06572.D0000104.?             SYSOUT
 IEF373I STEP /STEP1   / START 1996319.1340
 IEF374I STEP /STEP1   / STOP  1996319.1340 CPU    0MIN 00.03SEC SRB    0MIN 00.00SEC VIRT   576K SYS   188K EXT    4096K SYS    9444K
 IEF375I  JOB /SORT    / START 1996319.1340
 IEF376I  JOB /SORT    / STOP  1996319.1340 CPU    0MIN 00.03SEC SRB    0MIN 00.00SEC
1ICE143I 0 BLOCKSET     SORT  TECHNIQUE SELECTED                                                         4
 ICE000I 1 --- CONTROL STATEMENTS/MESSAGES ---- 5740-SM1 REL 12.0 ---- 13.40.28 NOV 15, 2006 --
0             SORT     FIELDS=(1,75,CH,A)
 ICE088I 1 SORT    .STEP1   .        , INPUT LRECL = 80, BLKSIZE = 80, TYPE = F
 ICE093I 0 MAIN STORAGE = (MAX,4194304,4194304)
 ICE156I 0 MAIN STORAGE ABOVE 16MB = (3624960,3624960)
 ICE128I 0 OPTIONS: SIZE=4194304,MAXLIM=1048576,MINLIM=450560,EQUALS=N,LIST=Y,ERET=RC16 ,MSGDDN=SYSOUT
 ICE129I 0 OPTIONS: VIO=N,RESDNT=ALL ,SMF=NO   ,WRKSEC=Y,OUTSEC=Y,VERIFY=N,CHALT=N,DYNALOC=N            ,ABCODE=MSG
 ICE130I 0 OPTIONS: RESALL=4096,RESINV=0,SVC=109 ,CHECK=Y,WRKREL=Y,OUTREL=Y,CKPT=N,STIMER=Y,COBEXIT=COB1
 ICE131I 0 OPTIONS: TMAXLIM=4194304,ARESALL=0,ARESINV=0,OVERRGN=65536,EXCPVR=NONE ,CINV=Y,CFW=Y
 ICE132I 0 OPTIONS: VLSHRT=N,ZDPRINT=N,IEXIT=N,TEXIT=N,LISTX=N,EFS=NONE    ,EXITCK=S,PARMDDN=DFSPARM ,FSZEST=N
 ICE133I 0 OPTIONS: HIPRMAX=OPTIMAL ,DSPSIZE=MAX
 ICE084I 0 BSAM ACCESS METHOD USED FOR SORTOUT
 ICE084I 0 BSAM ACCESS METHOD USED FOR SORTIN
 ICE090I 0 OUTPUT LRECL = 80, BLKSIZE = 80, TYPE = F
 ICE080I 0 IN MAIN STORAGE SORT
 ICE055I 0 INSERT 0, DELETE 0
 ICE054I 0 RECORDS - IN: 9, OUT: 9
 ICE134I 0 NUMBER OF BYTES SORTED: 720
 ICE180I 0 HIPERSPACE STORAGE USED = 0K BYTES
 ICE188I 0 DATA SPACE STORAGE USED = 0K BYTES
 ICE052I 0 END OF DFSORT
 EARTH
 JUPITER
 MARS
 MERCURY
 NEPTUNE                                                                                                 6
 PLUTO
 SATURN
 URANUS
 VENUS
```

*Figure 4. Output from Job Invoking SORT Program*

- The first block of information ( **1** ), ending with the line `$HASP395 SORT ENDED`, is installation-specific and may differ on your system.

- The JES2 job statistics section ( **2** ), ending with the amount of execution time, contains JES messages about the job.
- The next section ( **3** ) contains the JCL statements, or JCL listing, that resulted from the job.
- The next section ( **4** ), beginning with message ICH70001I and ending with message ICE052I, contains the system output messages resulting from processing the job.
- Within the system output messages, the condition code 0000 ( **5** ) tells you that the program ran successfully. You receive one condition code for each step in the job. If a condition code is non-zero, see the documentation for the specific program you invoked (in this case, SORT).
- The final section ( **6** ), from `EARTH` to `VENUS`, contains the output produced by the SORT program.

## Coding JCL: Collecting company-specific information

To correctly code JCL statements and parameter values, you need to collect certain company-specific information. Use this checklist and worksheet to document what you need to know about your company's IT environment.

### Procedure

Ask your system programmer or mentor to help you complete the following list, which identifies elements of your work environment that might affect the JCL that you code. Use Table 3 on page 55 to record your answers. When this worksheet is complete, you should have the company-specific information you need for most of the jobs you will run on z/OS.

- Determine which job entry subsystem (JES2 or JES3) is installed on the z/OS system you will use. For many jobs, the type of JES does not affect JCL parameters; for certain jobs, however, the JES in use does dictate which JCL parameters, values, or job entry control (JECL) statements you may code.
- Determine which access methods your company uses for its data sets. An access method defines the technique that is used to store and retrieve data. Access methods have their own data set structures to organize data, system-provided programs (or macros) to define data sets, and utility programs to process data sets. Access methods, therefore, determine which JCL parameters and parameter values that you need to code.
- For direct-access storage devices (DASD), determine which naming conventions are used, as well as default or recommended values for data set attributes.
- For storing or backing up data on tape, determine which tape device volume numbers and types are available for your use.
- Determine whether your company uses the Storage Management Subsystem (SMS) to automate the use of storage for data sets. The JCL parameters for SMS-managed (also called system-managed) data sets are different from some parameters used for non-SMS data sets.
- Determine the information (account number, programmer name, and so on) your company requires for each job that you submit.

*Table 3. JCL worksheet*

| **Company convention or z/OS environment specifics** | | **Notes / Values to code on JCL statements** |
|---|---|---|
| **Job entry subsystem**<br>JES2<br>or<br>JES3 | | |
| **Access methods**<br>Queued Sequential (QSAM)<br>Basic Partitioned (BPAM)<br>Virtual Sequential (VSAM)<br>Basic Sequential (BSAM)<br>Basic Direct (BDAM) | | |
| **Direct-access storage devices (DASD)** | | DSN=<br>UNIT=<br>VOL=SER= |
| **Magnetic tape devices** | | LABEL=<br>UNIT=<br>VOL=SER= |
| **Data management system** | | |
| Conventions for SMS-managed data sets | | |
| | Average record | AVGREC= |
| | Data classes | DATACLAS= |
| | Management classes | MGMTCLAS= |
| | RACF profile names | SECMODEL= |
| | Storage classes | STORCLAS= |
| Conventions for non-SMS-managed data sets | | |
| | Data set attributes or requirements | BLKSIZE |
| | | LRECL= |
| | | RECFM |
| | | SPACE= |
| | | SYSOUT= |
| **Conventions for the JOB statement** | | |
| | Account number | |
| | Other accounting information | |
| | Programmer name | |
| | Class | CLASS= |
| | Message class | MSGCLASS= |
| | Message level | MSGLEVEL= |
| | Region size | REGION= |
| | Time limit | TIME= |

## Coding JCL: Syntax rules for the name field

The name field begins in column 3 and may extend through column 10 in any JOB, EXEC, or DD statement. Using unique labels in the name field is useful for identifying, referring to, and diagnosing problems with specific JCL statements.

Syntax rules for the name field are:
- The name must begin in column 3 of the JCL statement.
- The name can be one through eight characters in length.
- The first character in the name must be an alphabetic character (the letters A through Z) or a special character (the symbols #, @, and $).
- The remaining characters can be alphanumeric (the letters A through Z and numbers 0 through 9) or special characters.
- Blank spaces cannot be included in a name.

## Coding JCL: Data set types and name syntax

Naming and syntax rules for coding data set names vary depending on the type of data set you are identifying. Use this summary to determine the type of data set and its corresponding syntax for the DSNAME parameter value.

Table 4 on page 57 lists the different types and examples of correctly coded names. Unless another resource is noted in the table, *z/OS MVS JCL Reference* (SA22-7597) is the definitive source for complete details about the data set types and permissible names, along with syntax rules.

*Table 4. Summary of data set types and correctly coded DSNAME (DSN) parameter values*

| Type of data set | DSNAME (DSN) parameter value formats and examples |
|---|---|
| **Permanent** | **Unqualified names:** One through 8 alphanumeric or special ($, #, @) characters, a hyphen, or a character X'C0'. The first character must be alphabetic or special ($, #, @).<br><br>**Example of an unqualified name:**<br>`DSNAME=ALPHA` |
| | **Qualified names:** Multiple unqualified names joined by periods. Each qualifier is coded like an unqualified name; therefore, the name must contain a period after every 8 characters or fewer. The maximum length of a qualified data set name is:<br>• 44 characters, including periods.<br>• For a generation data group, 35 characters, including periods.<br>• For an output tape data set, 17 characters, including periods.<br><br>**Example of a qualified name:**<br>`DSNAME=ALPHA.PGM` |
| | **RACF-protected data sets:** Use the same format as for a qualified name, and make sure the high-level qualifier of the name is defined to RACF. Further details are documented in *z/OS Security Server RACF Security Administrator's Guide* (SA22-7683). |
| | **Formats for names of cataloged data sets:**<br>*dsname*<br>*dsname*(*member*)<br>*dsname*(*gen_data_group*)<br>*dsname*(INDEX \| PRIME \| OVFLOW)<br><br>**Example for a cataloged data set:**<br>`DSNAME=LIB1(PROG12)`<br><br>Further details are documented in *z/OS DFSMS Access Method Services for Catalogs* (SC26-7394). |
| **Temporary** | When you define a temporary data set, you can code the **DSNAME** parameter or omit it; in either case, the system generates a qualified name for the temporary data set.<br><br>When you use the **DSNAME** parameter for a temporary data set, code the name as two ampersands (&&) followed by a character string 1 to 8 characters in length:<br>• The first character following the ampersands must be alphabetic or special ($, #, @).<br>• The remaining characters must be alphanumeric or special ($, #, @).<br><br>**Formats for temporary data set names:**<br>&&*dsname*<br>&&*dsname*(*member*)<br>&&*dsname*(INDEX \| PRIME \| OVFLOW)<br><br>**Example for a temporary data set:**<br>`//DD3 DD DSNAME=&&WORK,UNIT=3420` |

*Table 4. Summary of data set types and correctly coded DSNAME (DSN) parameter values (continued)*

| Type of data set | DSNAME (DSN) parameter value formats and examples |
|---|---|
| **In-stream or system output (sysout)** | When defining an in-stream or sysout data set, you can code the **DSNAME** parameter or omit it; if omitted, the system generates a name for the data set.<br><br>The data set name for in-stream and sysout data sets consists of two ampersands (&&) followed by one through eight 8 alphanumeric or special ($, #, @) characters, a hyphen, or a character X'C0'. The first character following the ampersands must be alphabetic or special ($, #, @).**Example for an in-stream data set:**<br>`//DDIN DD DATA,DSNAME=&&PAYIN1`<br><br>**Example for a sysout data set:**<br>`//DDOUT DD DSNAME=&&PAYOUT1,SYSOUT=P` |
| **Backward reference** | A *backward reference* is a reference to an earlier statement in the job or in a cataloged or in-stream procedure called by this or an earlier job step. A backward reference can be coded in the **DSNAME** parameter to copy a data set name from an earlier DD statement.<br><br>**Formats for backward references:**<br>`*.`*`ddname`*<br>`*.`*`stepname.ddname`*<br>`*.`*`stepname.procstepname.ddname`*<br><br>**Example of a backward reference in DD5 statement in STEP2:**<br>`//STEP1 EXEC PGM=CREATE`<br>`//DD4   DD DSNAME=&&ISDATA(PRIME),DISP=(,PASS),`<br>`//         UNIT=(3350,2),VOLUME=SER=334859,`<br>`//         SPACE=(CYL,(10,,2),,CONTIG),DCB=DSORG=IS`<br>`//STEP2 EXEC PGM=OPER`<br>`//DD5   DD DSNAME=*.STEP1.DD4,DISP=(OLD,DELETE)` |
| **Dummy data set** | The parameter **NULLFILE** specifies a dummy data set. **NULLFILE** has the same effect as coding the DD **DUMMY** parameter. |

# Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

## Programming interface information

This book documents information that is NOT intended to be used as Programming Interfaces of z/OS.

## Trademarks

Printed in USA