

Instructor Notes:



Job Control Language (JCL)

Lesson 00 Course Overview

©2016 Capgemini. All rights reserved.
The information contained in this document is proprietary and confidential.
For Capgemini only.

Instructor Notes:**Document History**

Date	Course Version No.	Software Version No.	Developer / SME	Change Record Remarks
16-July-2009	4.0	JCL	Arjun Singh	Content Creation
26-Oct-2009	4.0	JCL	Documentation Team	Quality Review
30th-May-2011	5.0	JCL	Vaishali Kasture	Content creation and Revamping
10 th -Feb-2015	6.0	JCL	Veena Keshavalu	Revamping
26 th -June-2016	6.0	JCL	Veena Keshavalu	Revamping post the integration

Instructor Notes:**Course Goals and Non Goals****Course Goals**

- Compile, Link and Run a Cobol Program using the JCL codes.
- Execution of Jobs in batch mode.

Course Non Goals

- Application and use of JCL and its utility with VSAM, DB2, IDMS and ADSO.



Instructor Notes:

Pre-requisites

Knowledge of MULTIPLE VIRTUAL STORAGE (MVS).

Instructor Notes:

Add instructor notes here.

Intended Audience

Programmers



Instructor Notes:**Day Wise Schedule****Day 1**

- Lesson 1: Introduction
- Lesson 2: JCL Syntax
- Lesson 3: Job Statement

Day 2

- Lesson 4: THE EXEC STATEMENT
- Lesson 5: THE DD STATEMENT

Day 3

- Lesson 6: PROCEDURE

Day 4

- Lesson 7: UTILITY

Day 5

- Lesson 7: UTILITY

Instructor Notes:**Table of Contents**

Lesson 1: Introduction

- 1.1: What is JCL?
- 1.2: What is JOB?
- 1.3: Processing of JCL

Lesson 2: JCL Syntax

- 2.1: Syntax of JCL
- 2.2: JCL Rules

Instructor Notes:**Table of Contents****Lesson 3: Job Statement**

- 3.1: Job Statement
- 3.2: Parameter Field
- 3.3: Accounting Information
- 3.4: Programmer's Parameter Name
- 3.5: MSGLEVEL Parameter
- 3.6: MSGCLASS Parameter
- 3.6: CLASS Parameter
- 3.8: PRTY Parameter
- 3.9: TIME Parameter
- 3.10: REGION Parameter
- 3.11: ADDRSPC Parameter
- 3.12: NOTIFY Parameter
- 3.13: RESTART Parameter
- 3.14: TYPRUN Parameter

Instructor Notes:**Table of Contents****Lesson 4: THE EXEC STATEMENT**

- 4.1: THE EXEC STATEMENT
- 4.2: THE PGM Parameter
- 4.3: Use of Other Parameters in EXEC
- 4.4: THE ADDRSPC PARAMETER
- 4.5: THE ACCT PARAMETER
- 4.6: THE PARM PARAMETER
- 4.6: THE COND PARAMETER
- 4.8: Using the COND, JOB, EXEC and PGM Parameters
- 4.9: Using IF/Else/EndIf

Instructor Notes:**Table of Contents****Lesson 5: THE DD STATEMENT**

- 5.1: Function of the DD Statement
- 5.2: The DSNAMES Parameter
- 5.3: The DISP Parameter
- 5.4: The UNIT and VOLUME Parameters
- 5.5: The VOL Parameter
- 5.6: The SPACE Parameter
- 5.6: The LABEL Parameter
- 5.8: The DCB Parameter
- 5.9: Models
- 5.10: In-Stream Data Format
- 5.11: The SYSOUT, SYSIN, SYSPRINT Parameter
- 5.12: The DD Statement – Concatenation
- 5.13: The DUMMY Parameter
- 5.14: The JOBLIB DD Statement
- 5.15: The STEPLIB DD Statement
- 5.16: Job Cat and Step Cat
- 5.16: Storage Dump

Instructor Notes:**Table of Contents****Lesson 6: PROCEDURE**

- 6.1: PROCEDURES
 - 6.1.1: Catalogued Procedures
 - 6.1.2: IN-STREAM Procedures
- 6.2: Rules to Override JCL Procedures
 - 6.2.1: Procedure LAM (S1)
 - 6.2.2: Required in Step 1
 - 6.2.3: Procedure LAM (S2)
 - 6.2.4: Required in Step 2
 - 6.2.5: Procedure LAM (S3)
 - 6.2.6: Examples
- 6.3: Symbolic Parameters & Symbolic Overrides
 - 6.3.1: Symbolic Parameter Examples
 - 6.3.2: Symbolic Overriding
- 6.4: The PROC Statement
 - 6.4.1: SET Statement
 - 6.4.2: INCLUDE Statement

Instructor Notes:**Table of Contents****Lesson 7: UTILITY**

- 7.1: IEFBR14 UTILITY
- 7.2: IEBGENER UTILITY
- 7.3: IEBCOPY UTILITY
- 7.4: IEHLIST UTILITY
- 7.5: SORT UTILITY
 - 7.5.1: DFSORT
 - 7.5.1: Sorting by Multiple Fields
 - 7.5.2: Copying Data Sets
 - 7.5.3: Tailoring the Input Data Set With INCLUDE and OMIT
 - 7.5.4: MERGE Fields
 - 7.5.5: Summing Records – SUM Statement
 - 7.5.6: Suppress Records With Duplicate Control Fields
 - 7.5.7: Reformatting Records – INREC & OUTREC
 - 7.6: ICETOOL
 - 7.6.1: Copy
 - 7.6.2: Sort

Instructor Notes:

Add instructor notes here.

[References](#)

Expert MVS/XA JCL - Mani Carathanassis
MVS/JCL - Doug Lowe



Instructor Notes:

Add instructor notes here.

Next Step Courses (if applicable)

COBOL
VSAM
CICS
DB2



Instructor Notes:



Job Control Language (JCL)

Lesson 01 Introduction

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential.
For Capgemini only.

Instructor Notes:

Lesson Objectives

Introduction to JCL



Instructor Notes:

1.1: What is JCL?
Introduction

Work that a user can perform under MVS/XA Topic:

- Time Sharing
- Online Processing
- Batch Processing

Instructor Notes:**1.1: What is JCL?**
JCL Description

Language that allows users to communicate with the operating system.

Step tells the operating system the following:

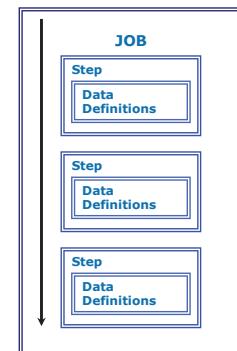
- Application Program details:
 - name, location, components , OS components etc. to be processed.
- Data file details:
 - The name and location of files the application program needs.
- Hardware devices:
 - Devices the application program needs to achieve its function.

Instructor Notes:**1.2: What is JOB?**
Description

JCL control statements are organized into groups of work called Jobs.
A JOB is simply the execution of a program or a set of related programs
General structure of a simple JOB:

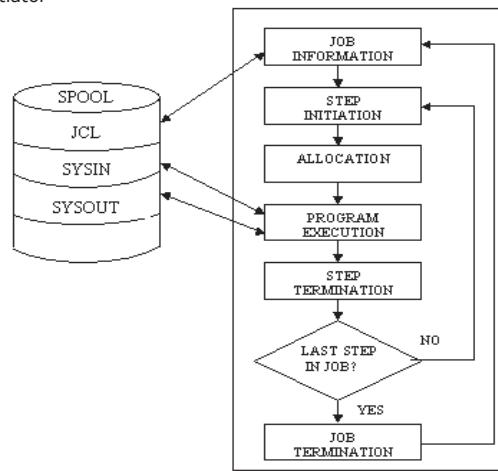
```
//Job statement
//Step 1      EXEC program
//           DD statements
//
//Step 2      EXEC program
//           DD statements
//
//Step 3      EXEC program
//           DD statements
//
//           DD statements
```

Each program executed by a job is called a JOB step



Instructor Notes:1.2: What is JOB?
Description (contd..) – SAMPLE JCL

```
//DA0001TA JOB    LA2719,CG,MSGCLASS=A,  
//  MSGLEVEL=(1,1),NOTIFY=DA0001T  
//SORT1   EXEC  PGM=SORT  
//SYSOUT  DD      SYSOUT = *  
//SORT    DD      DSN=DA0001T.TRANFILE,  
//  DISP=SHR  
//SORTOUT DD      DSN=DA0001T.TRANSORT,  
//  DISP=(NEW,KEEP),UNIT=SYSDA,  
//  SPACE=(TRK,(1,1)),  
//  DCB=(DSORG=PS,RECFM=FB,  
//  LRECL=400,BLKSIZE=3200)  
//SORTWK01  DD      UNIT=SYSDA,  
//  SPACE=(CYL,(1,1))  
//SYSIN   DD      *  
          SORT FIELDS=(16,5,CH,A)  
/*  
//
```

Instructor Notes:**1.3: Processing of JCL
JOB Execution****JOB Execution by Initiator**

Instructor Notes:**1.3: Processing of JCL**
JOB Execution (contd..)

If an appropriate job exists in job queue, initiator takes its JCL and goes through the following processes:

- Job initiation
- Step initiation
- Allocation
- Program execution
- Step Termination
- Job Termination

Instructor Notes:**Summary**

Job Control Language (JCL):

- Means of communication with IBM 3090 MVS Operating System.

Initiator

- Set of routines with sole function to select a job from job queue and execute it under its control.
- Selects appropriate jobs and goes through following processes:
 - Job initiation, Step initiation, Allocation, Program execution, Step termination and job termination.



Instructor Notes:**Review Question**

Question 1: Which of the following picks a job from the job queue and processes it?

- Option 1: Initiator
- Option 2: Online
- Option 3: Batch

Question 2: Which of the process will check the COND parameter on the JOB statement?

- Option 1: Job initiation
- Option 2: Step initiation



Instructor Notes:



Job Control Language (JCL)

Lesson 02 JCL Syntax

©2016 Capgemini. All rights reserved.
The information contained in this document is proprietary and confidential.
For Capgemini only.

Instructor Notes:

Lesson Objectives

Syntax for writing JCL
Rules to be followed



Instructor Notes:**2.1: Syntax of JCL**
Description

JCL statements are coded in 80-byte records.

Only 72 characters are used to code JCL statements.

Within 72 characters, JCL statements can be coded in a relatively free-form manner, with just few restrictions.

Each JCL statement is divided into many fields.

All JCL statements begin with two slashes (//) in the first two positions (except /*). All positions of a line, from 1 to 71 (included), can be used for coding a JCL statement. Position 72 is used (rarely) for imbedded comment continuation, and positions 73 through 80 are used for numbering purposes.

With the exception of the null (//), delimiter (*), comment (//*), all other statements follow the same general format:

//name operation parameter1,parameter2 [comment]

Instructor Notes:**2.1: Syntax of JCL
Description (contd..)**

Different types of JCL statements

- JOB
- EXEC
- DD
- DELIMITER
- Null
- COMMENTS
- PROC
- PEND
- OUTPUT

JOB- Marks the beginning of a job and assign a name to the job.
EXEC- Marks the beginning of a job and names the job step. Identifies the program or procedure to be executed
DD- Identifies an input or output file within a job step and defines all the resources required
DELIMITER(/*)- Indicates the end of data placed in a JCL
NULL(//)-Marks the end of a job
COMMENTS//(/*) - Contains comments
PROC - Marks the beginning of a procedure definition
PEND - Indicates the end of a procedure definition
OUTPUT - Supplies options for SYSOUT processing

Instructor Notes:**2.1: Identifier Field Format**

```
//NAME    Operation    param1,param2 ...
```

Identifier Field

- Identifies a record as a JCL statement.
- For most JCL statements, this field occupies the first two character positions and contains two slashes (//).
- Two exceptions are:
- Delimiter statement that contains a slash and the start(/*)
- Comment statement with identifier field as (//*).

Name: Every JCL statement can or must have a name. The name should not exceed 8 characters.

Operation: The operation field follows the name field and specifies the statement's function. Delimiter, comment and null statements do not have an operation field. E.g. JOB, EXEC, DD, PROC, PEND, or OUTPUT. One or more blanks must follow the operation.

Parameter: The parameter field consists of one or more parameters, separated by commas. No imbedded blanks between parameters are permitted. Parameters are broadly classified into 2 categories viz. Positional and Keyword.

A positional parameter is identified by its position relative to other parameters in the operand field.

Note: THE COMMENT STATEMENT CAN NOT BE CONTINUED!

Instructor Notes:**2.1: Name Field Format****Name Field**

- Associates a name with a JCL statement.
- Always required on a JOB statement.
- Must start from column three.
- Can be up to 8 characters in length.
- Comprises letters, numbers or national characters (#, @ and \$).
- First character must be a letter or national character.

Instructor Notes:**2.1: Operation Field Format****Operation Field**

- Specifies the statement's function.
- Can be coded anywhere on the line, but should be separated from the name field by at least one space.
- Statements that do not have operation field are:
 - /* ----- Delimiter
 - // ----- Null
 - ///* ----- Comment

Instructor Notes:**2.1: Parameter Field Description**

In JCL parameters are of two types:

- **Positional**

- must be first
- must be in a specific order
- absence must be indicated by a comma
- if the last one is absent, no comma necessary
- if all are absent, no comma is necessary.
- E.g..
 - TIME=(5,0) = TIME=(5) = TIME=5
 - TIME=(0,30) = TIME=(,30)

- **Keyword**

- must be after any positional parameters
- can be any order with respect to one another
- Identified by an equal sign (=) and variable information
- DISP=OLD,DISP=(NEW,KEEP,KEEP),DISP=(,KEEP,KEEP)
- DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)

Instructor Notes:**2.1: Parameter Field Format****Parameter Field**

- Begins at least one position after the end of operation field and can be extended up to column 71.
- One or more component parameters are coded, each separated by commas.
 - Parameters supply information that influence statement processing.
- Comments field begins in the position after the space that marks the end of parameters field.

Rule 1: All positional parameters are coded first in the operand field and in their proper sequence.

E.g.: p1,p2,p3

p1,p3,p2 illegal because they are not in sequence.

Rule 2: A keyword parameter is identified by a keyword followed by an equal sign (=) and variable information. A keyword parameter follows positional parameter and can be coded in any order.

E.g. : p1,p2,p3,k1=,k2=,k3=

p1,p2,p2,k3=,k2=,k1=

Both are valid.

Rule 3: The absence of positional parameter is denoted by a comma (,) coded in its place, except when the last or remainder of the positional parameter is not present. The placeholder commas do not need to be coded in this case.

E.g.: p1,,p3,k1=,k2=,k3= place holder comma required.

p1,k1=,k2=,k3= place holder comma not required.

Instructor Notes:**2.1: Syntax of JCL
Example**

Syntax

```
//OUTPUT DD DSN=DA0001T.TRANS,DISP=(,CATLG),  
//    SPACE=(TRK,(2,1)),UNIT=SYSDA
```

Rule 4: Both positional parameters and variable information for keyword parameters may be composed of sub parameters. The sub-parameters may be either positional or keyword. Sub-parameters must be coded as a list. The list must be enclosed in parentheses unless only one sub-parameter is coded. When only one sub-parameter is coded the parentheses are optional.

e.g.: p1,(sp1,sp2),p3,k1=(sk1,sk2),k2=,k3=
p1,(sp1),p3,k1=(sk1),k2=,k3=
Or p1,sp1,p3,k1=sk1,k2=,k3=

Comments: are separated from the parameters by a blank. The comment field begins in the position after the space that marks the end of parameters field and ends in column 71. MVS ignores what you code here.

Instructor Notes:**2.2: JCL Rules**
Rules

Positional parameters are coded first in the operand field and in their proper sequence.

- Absence is denoted by a comma (,) coded in its place.

A keyword parameter is identified by a keyword followed by an equal sign (=) and variable information.

Both positional parameters and variable information for keyword parameters may comprise sub-parameters.

Instructor Notes:**2.2: JCL Rules**
Rules for Continuation

JCL statement can be continued into the next line.

- Code two slashes at the beginning of the line.
- Continue in the parameter field.
- Start anywhere between positions 4 and 16 (4 and 16 included).

• Note:

- A comma enclosed in an apostrophe cannot be used for continuation.

Rules for Continuation: The JCL statement can be continued in a simple way. The statement must be interrupted at a comma. This means that the last valid character of the line must be a comma followed by at least one blank. Then the statement can be continued into the next line by coding two slashes at the beginning of the line and continuing the parameter field starting anywhere between positions 4 and 16 (4 and 16 included). Note that, the comma that indicates continuation, is not an extraneous character but part of the statement.

Instructor Notes:

2.2: JCL Rules

Rules for Continuation: Example

```
//DD1    DD DSN=DA0001T.EMPFFILE,  
//      DISP=(NEW,CATALOG,DELETE),  
//      UNIT=SYSDA,SPACE=(TRK,(5,1)),  
//      DCB=(LRECL=80,RECFM=FB,BLKSIZE=800)
```

Instructor Notes:**Summary**

Every JCL statement must have a name.

A keyword parameter is identified by a keyword followed by an equal sign (=) and variable information.

Absence of positional parameter is denoted by a comma (,).



Instructor Notes:**Review Question**

Question 1: All JCL statements begin with:

- Option 1: //
- Option 2: **
- Option 3: --

Question 2: Which of the parameters are coded first in a JCL statement?

- Option 1: Positional
- Option 2: Keyword



Instructor Notes:



Job Control Language (JCL)

Lesson 03 Job Statement

©2016 Capgemini. All rights reserved.
The information contained in this document is proprietary and confidential.
For Capgemini only.

Instructor Notes:

Lesson Objectives

Job statement

Accounting Information, Programmer's Name parameter

MSGLEVEL, MSGCLASS, CLASS, PRTY, TIME, REGION , ADDRSPC, NOTIFY, RESTART parameter



Instructor Notes:**3.1: Job Statement
Functions****JOB statement:**

- First statement to be coded for a JCL.
- It has three basic functions :
 - Identifies a job to MVS and supplies a job name for MVS to refer to the job.
 - Supplies accounting information to MVS.
 - Supplies various options that influence or limit job processing.

A JOB statement must be at the beginning of every job submitted to the system for execution. It must have a name. The absence of a job name will result in a JCL error. JOB statement identifies a job to the operating system with the job name operand. There are three possible delimiters for a job during a reading process:

Another job statement in the input stream. It signals the end of (reading) one JOB and the beginning of (reading) another

A null statement. Following a null statement all JCL statements except a JOB statement will be ignored

End-of-file on the reading device, meaning there are no more statements to be read in

Instructor Notes:**3.1: Job Statement Format**

```
//jobname JOB ([account-no] [,additional, accounting-info.]),  
//      "Programmer's name", MSGCLASS=class,  
//      MSGLEVEL=(JCL [,MSG]), NOTIFY=user-id,  
//      USER=user-id, PASSWORD=password,  
//      CLASS=jobclass,  
//      ADDRSPC={VIRT|REAL},  
//      TIME=([minutes][,seconds]|[1440]),  
//      REGION=Value{K|M},  
//      PRTY=priority,  
//      RESTART={stepname|procexec.stepname|*},  
//      TYPRUN={HOLD|SCAN},  
//      COND=(test[,test].....)
```

Instructor Notes:

3.1: Job Statement
Example

```
// DA0001TA JOB (LA2719,TRG),CG,  
//      MSGCLASS=A,  
//      MSGLEVEL=(1,1),  
//      NOTIFY=DA0001T,  
//      TIME=(2, 3),  
//      REGION=500K,  
//      TYPRUN=SCAN,  
//      RESTART=STEPNAME
```

Instructor Notes:**3.1: Job Statement
Example**

A job statement must have a name. The absence of job name will result in a JCL error:
• Format:

//job name JOB parameters

//DA0001TA JOB parameters

The job name cannot exceed 8 characters and is usually the user id (login id). Login id is generally 7 characters, so you need to add a suffix else the system prompts you to enter the character when a job is submitted to the system for execution.

When a job is submitted to the system, a job number is also assigned so that the job can be further identified. This way jobs with the same name can be uniquely identified. Jobs with the same name cannot execute simultaneously. If several jobs with the same name are submitted they execute sequentially even if additional jobs could be executing. Jobs waiting to run because of this time conflict are shown in hold status.

The rest of the JOB statement contains positional parameters followed by keyword parameter.

Instructor Notes:**3.2: Accounting Information**
Description**Accounting Information:**

- Positional parameter.
- If present, it must be the first in the parameter field.
- Consists of several positional parameters, the first of which is account number.
- The account number is an alphanumeric field, 1 to 4 characters long (more than 4 is also permitted).
- Maximum size is 142 characters.
- It is normally used to determine how billing is to be done.

If a positional parameter is present (it normally is), it must be the first in the parameter field. It can have a maximum of 142 characters (including parentheses and commas but not apostrophes). It is used to tie the resources used by the job to the appropriate account.

The account-number is an alphanumeric field from 1 to 4 characters long (many installations permit the use of more than 4 characters).

Additional-accounting-information is installation dependent. Many of the fields are not very important and are not often used.

Instructor Notes:**3.2: Accounting Information
Format and Example**

Format :

([account-number] [,additional. accounting-info.])

Example:

//DA0001TA JOB (LA2719,TRG)

Example:

//DA0001TA JOB LA2719, parameters..

LA2719 is the account number for the training dept. This varies from one project to another.

Remark: An installation has the option of making the account number mandatory and most installations do. If so, its absence causes a JCL error.

If the account number is incorrectly specified, in this case its not JCL error. However when job is submitted to the system for execution, we get the message "JOB NOT RUN" in the SYSOUT.

Instructor Notes:**3.3: Programmer's Name Parameter**
Description

It is a positional parameter.

Follows accounting information.

Installation-dependent.

Maximum size is 20 characters

- Note: If it contains any special characters other than a hyphen, and a period in the middle or the beginning but not at the end of the name, the name must be enclosed in apostrophes.

Following the accounting information parameter, another positional parameter, the programmer's name can be coded. The installation determines if this parameter is required or not. If required, it must be coded immediately after the accounting information, and its omission will cause a JCL error. The programmer's name cannot exceed 20 characters. If it contains any special characters other than a hyphen and a period in the middle of the beginning (but not at the end) of the name, the name must be enclosed in apostrophes. The apostrophes are not added to the length of the name.

Instructor Notes:

3.3: Programmer's Name Parameter
Example

```
//DA0001TA JOB LA2719,'O"Kelly',....
```

If a name contains an apostrophe (e.g., D'COSTA), two apostrophes must be coded. They count as one character in the length of the name.

Example:

```
//DA0001TA JOB LA2719,Sheela,parameters  
//DA0001TA JOB LA2719,'D"COSTA',parameters
```

Instructor Notes:**3.4: MSGLEVEL Parameter**
Description

Keyword parameter

Optional

Installation-dependent

Allows you to control:

- Contents of JCL print
- System messages
 - MSGLEVEL lets us specify the type of messages you wish to include in your output
 - Determines the amount of JCL and SMS allocation messages to be assigned to the device indicated vide MsgClass

This parameter specifies whether the submitted JCL or JCL-related messages should be shown on the job's output.

Instructor Notes:**3.4: MSGLEVEL Parameter Format****MSGLEVEL=([JCL Statements][,Message])**

JCL Statements : 0,1 or 2

Message : 0 or 1

General syntax:

MSGLEVEL=([jcl][,messages]) keyword parameter

jcl - 0, 1, or 2

- 0 - Only the JOB statement will be shown
- 1 - All JCL is shown

Instream

Expanded cataloged procedures

Symbolic parameter substitutions

- All JCL is shown, but not expanded procedure listing.

messages – 0 or 1

- 0 - No messages will be shown i.e. information about step completion.
- All messages will be shown viz. allocation and termination messages.

The messages sub-parameter can be thought of as On (1) or Off (0).

The default in majority of the installations is (1,1).

Instructor Notes:**3.4: MSGLEVEL Parameter
JCL Values****JCL Statements**

0
1
PROCEDURES
2

Definition

Print the job statement
Print all the JCL statements along with the
Print only JCL statements

Message

0
terminates
1
terminates

Allocation/termination messages, if the job
abnormally
Allocation/termination messages, if the job
abnormally/normally

Remark:

1. If the entire parameter or either of the two fields is omitted, an installation-defined default is assumed.
MSGLEVEL=1 ----- MSGLEVEL=(1,default)
MSGLEVEL=(,1) ----- MSGLEVEL=(default,1)
Parameter omitted ----- MSGLEVEL=(default,default)
2. If the job encounters an ABEND failure, the second field always defaults to 1 even if coded as 0.

Instructor Notes:**3.4: MSGLEVEL Parameter
The JOB Statement – MSGLEVEL (0,0)**

- JESJCL: Shows only JOB statement details

The screenshot shows a terminal window titled "Session A - [24 x 80]". The menu bar includes "File", "Edit", "View", "Communication", "Actions", "Window", and "Help". The main display area shows the following JCL code:

```
SSDF OUTPUT DISPLAY P390B97R JOB07754 DSID      3 LINE 0      COLUMNS 01- 80
COMMAND INPUT ===> -
***** TOP OF DATA *****
1 //P390B97R JOB (RCCTH,&SYSUID),
//          MSGCLASS=R,CLASS=R,MSGLEVEL=(0,0),NOTIFY=&SYSUID,
//          REGION=1000K,TIME=1200
IEFC653I SUBSTITUTION JCL - (RCCTH,P390B97),MSGCLASS=R,CLASS=R,MSGLE
TIME=1200
***** BOTTOM OF DATA *****
```

At the bottom of the screen, function key definitions are listed:

```
PF 1=HELP      2=SPLIT      3=END       4=RETURN     5=IFIND      6=BOOK
PF 7=UP        8=DOWN       9=SWAP      10=LEFT      11=RIGHT     12=RETRIEVE
```

The status bar at the bottom right shows "04/021".

Instructor Notes:

3.4: MSGLEVEL Parameter
Demo

Job Parameters



Instructor Notes:**3.5: MSGCLASS Parameter**
Description

Keyword parameter

Optional

Installation-dependent

Format :

MSGCLASS= sysout class

- Sysout class is a one-character code.
- viz : A-Z, 0-9 max. 36 output classes.
- MSGCLASS specifies an output class that's associated with the jobs message output.
- MVS assigns a default, if the parameter is omitted

This parameter assigns a sysout class to the Job log. The job log consists of what is known as system or JES datasets:

JES2 or JES3 log
JCL and its associated messages
Allocation and Termination messages

MSGCLASS - indicates the format of output
 - Specifies output class for
 Job log (collection of all operations)
 List (collection of all printed output like compiled
class - A character from A to Z or a number from 0 to 9 (in all 36 classes)
MSGLEVEL parameter indicates whether or not one wishes to print the JCL
statements and allocation messages. The MSGLEVEL can save paper. After a
job is debugged, there may be no need to print all the JCL and allocation
messages each time it runs. To reduce printing to a minimum, one may wish to
MSGLEVEL=(0,0).

Remark: If the MSGCLASS parameter is omitted, an installation-defined
default will be used.

Instructor Notes:**3.5: CLASS Parameter**
Description

Nature of job (Short Running (cpu time), Long Running, Large resources (tape,disk))

Keyword parameter

Optional

Installation-dependent

This parameter assigns a class to a job.

General Syntax:

CLASS=jobclass Keyword parameter

jobclass – A letter from A to Z or a number from 0 to 9 (in all 36 classes).

The job class affects job's processing in these ways:

When job is submitted, it is placed in an input queue where it waits to be executed. Queues can be thought of as waiting lines for jobs. Each job class has its own input queue

Job waits in the input queue until it is selected by an initiator to be processed. Each initiator is set to a list of job classes that it can select from.

Simply put, Jobclass identifies the nature of the job:

- Short running or long running
- Resource utilization

Instructor Notes:**3.5: CLASS Parameter**
The JOB Statement - CLASS

THE CLASS PARAMETER : Queues jobs for execution based on the importance of the job. The importance in turn depends on the type of the job:

- A job involving complex statistical calculations
 - Is a type of job taking a lot of CPU time and to be run in the night only
- A job updating a database
 - Is a type of job which is I/O bound and to be run immediately
- A job generating reports
 - Is a type of job to be run in the evenings
- A job running system programs
 - Is a type of job to be run continuously

Instructor Notes:**3.5: CLASS Parameter Format**

The CLASS parameter assigns an input class to a job.

A job class is a one character code.

Values assigned can be A-Z, 0-9

CLASS parameters specifies class in which the job is to scheduled.:

- Note : Each job class has its own input queue and CLASS gives jobs a preference for execution by the OS.

CLASS=job class

Each installation group jobs that have like characteristics into classes. By segregating jobs with similar characteristics, an installation can maintain a good mix of the jobs running at a given moment. This maintains system throughput and efficient use of resources.

For example, suppose the default CLASS is A:

Job statement:

//DA0001TA JOB LA2719,CG,MSGCLASS=A,MSGLEVEL=(1,1)

is equivalent to:

//DA0001TA JOB
// LA2719,CG,MSGCLASS=A,MSGLEVEL=(1,1),CLASS=A

Instructor Notes:**3.6: PRTY Parameter**
Description

Keyword parameter

Optional

Installation-dependent

PRTY specifies the priority of the job within the job class.

- Example:

CLASS=A, PRTY=3

This parameter determines the scheduling priority of a job in relation to other jobs in the job input queue of the same class.

The PRTY parameter is used to define the job's input class selection priority. The higher the number, the better (greater) the priority. The PRTY parameter simply controls the job's position in the input queue. It has no affect on the job's performance. Jobs with higher priorities will be selected before job's with lower priority. A job's priority does not affect its performance. Once the job is selected for execution, the priority function is finished. Two jobs having same job class and same priority will be executed in sequence. It is meaningless to compare the PRTY parameter of two jobs belonging to different classes.

Format :

PRTY=priority-no.

For JES2, it is a no. between 0 & 15

For JES3, it is a no. between 0 & 14

0 - lowest priority

15 - highest priority for JES2)

Instructor Notes:**3.7: TIME Parameter**
Description

Keyword parameter

Optional

- Format:

TIME=([minutes][,seconds]|1440)

TIME parameter:

- Specifies the amount of CPU time a job may use.
 - For example: All steps in a job can use it collectively.

This parameter specifies the total amount of CPU time that all steps in a job can use collectively.

minutes - a number from 1 to 357912 (248.55 days)

seconds - a number from 1 to 59

The job is not timed for CPU. Note that TIME=1440 is rarely used, and most installations disallow its use in a testing environment. TIME=1440 should be used by an on-line system like CICS OR ADS/O.

When the TIME parameter is omitted, an installation-defined default is used. This default is usually very high and unlikely to cause an S322 ABEND failure unless the program goes into an endless loop.

If the TIME parameter is also coded in the JOB statement and exec statement within job , both will be in effect and either can cause a S322 ABEND failure. It is not advisable to use them both.

CPU time is the amount of time that the computer devoted to the job after it was selected for processing. It is not the amount of time it was in the machine.

Instructor Notes:

3.7: TIME Parameter

Example

- TIME=1440 (no time-limit)
 - Job is not timed for CPU. It also does not time-out(S522 ABEND failure)
- TIME=(3,20)
 - All the steps in a job are allowed collectively 3 minutes and 20 seconds of CPU time
 - If this amount exceeds the result will be S322 ABEND failure
 - TIME=MAXIMUM
 - Permits maximum CPU time for a job for 357912 minutes

Reasons to code TIME

While testing a new program which may go into a loop, unless it is “timed out” automatically (system code 522)

Special jobs which need to by-pass any Time limit barrier and which must not be “timed out” too.

TIME parameter puts an upper limit on the amount of CPU time that a job may use.

Example: TIME=(3,20). All the steps in the job are allowed collectively 3 minutes and 20 seconds of CPU time. If this amount is exceeded, the result will be a S322 ABEND failure.

If the TIME parameter is coded using only minutes, seconds defaults to zero. For example, TIME=5 is the same as TIME=(5,0).

If the TIME Parameter is coded using only seconds, minutes defaults to zero. For example, TIME=(,6) is the same as TIME=(0,6).

The TIME parameter is intended almost exclusively for a testing environment and should be coded to preempt the program going into CPU loop.

The TIME parameter can also be supplied by the CLASS parameter. When the TIME parameter is omitted and the CLASS parameter does not supply it, the job will not be timed for CPU time. However each step will be individually timed (TIME parameter at EXEC statement or its installation-defined default), unless it contains TIME=1440.

Instructor Notes:**3.8: REGION Parameter**
Description

REGION specifies the amount of storage (central / virtual) a job is allocated (Max. storage requirement).

Maximum virtual memory available is 2GB.

- Keyword parameter
- Optional
- Installation-defined

This parameter specifies the limit of available storage for each of the steps in the job within the job's address space. i.e., the amount of storage the job is allocated. In other words, it specifies the amount of storage needed by the step (within the job) with the highest storage requirements.

Instructor Notes:

3.8: REGION Parameter

Format and Example

Value - 1 to 2096128 if K is used.

Value - 1 to 2047 if M is used (M is not available to MVS/SP)

REGION=value(K|M)

General SyntaxREGION=value{K|M}
keyword parameter

value – 1 to 2096128 if K (1024 bytes) is used. It should be an even number. If an odd number is used it will be rounded off to the next higher even number. value – 1 to 2047 if M (1024K or 1048576 bytes) is used. M is not available to MVS/SP, only to MVS/XA and MVS/ESA.

When a job is selected by an initiator for execution, it is given an address space of 16 MB (minus what MVS/SP uses). In case of MVS/XA, job is given an address space of 2GB. And all of it is available to the job's steps. However a step normally requires only a small fraction of this huge storage, below the 16M line. An ordinary COBOL or any other language program seldom needs more than 1000K. This is normally what the value in the REGION parameter represents in the installations. Few jobs like CICS, IMS, DB2 need storage above the 16M line. An ordinary batch job seldom has such high requirements and, as a result confined to storage below the 16M line. Storage availability below this line varies in different installations, but is generally around 8MB in MVS/SP and around 9 MB in MVS/XA. Storage above the 16M line can be acquired by coding a value higher than 16M. However, it may be restricted by the installation to only those jobs that need it.

Instructor Notes:**3.8: REGION Parameter****Example 1**

All the steps in the job are limited to this value. If more storage is needed the result is a S878 or S808 or S804 ABEND failure. If one of this failure's occurs the user must increase the value in the REGION parameter.

When the amount of storage requested in the REGION parameter is higher than the address space can provide, an S822 ABEND failure will result.

REGION=500K

Examples:**Example 1:**

Assume REGION=1000K were coded in the JOB statement. All the steps in the job are limited to this value. If more storage is needed, the usual result is S878 or S80A or S804 ABEND failure. If one of these failures occurs, the user must increase the value in the REGION parameter.

Example 2:

REGION=10M

When the amount of storage requested in the REGION parameter is higher than the address space can provide, an S822 ABEND failure occurs.

Instructor Notes:

[3.8: REGION Parameter](#)
[Example2](#)

REGION=0K (or 0M) is coded, the entire address space (except those areas used by MVS/SP or MVS/XA) is available.

Remark: If REGION parameter is omitted, an installation defined default will be used.

Instructor Notes:**3.9: ADDRSPC Parameter Description**

The ADDRSPC parameter specifies whether the job will use Real or Virtual storage

Keyword parameter

Optional

- Format:

- Note: ADDRSPC=REAL is a parameter disallowed in practically all installations because of performance problems.

ADDRSPC={VIRT|REAL}

General Syntax:

ADDRSPC={VIRT|REAL}

ADDRSPC=REAL the allocation is done in REAL storage and the program is not pageable

ADDRSPC=VIRT the allocation is done in VIRTUAL storage and the program is pageable

Remark: This is the rarely used parameter because of the default. Note that ADDRSPC=REAL is a parameter that is disallowed in practically all installation because it can cause serious performance problems for other jobs.

Instructor Notes:**3.10: NOTIFY Parameter**
Description

Keyword parameter
• Format:

NOTIFY=user-id

Requests that the OS sends a message to TSO user-id about:

- the Job's completion and
- its completion status – whether normal or abnormal
- Note: If the NOTIFY parameter is omitted no message will appear when the job terminates

General Syntax

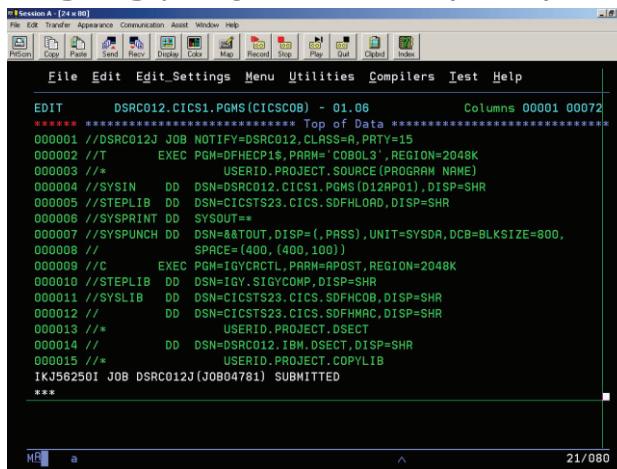
NOTIFY=userid keyword parameter

userid – A name from 1 to 7 characters, identifying a valid TSO user.
Example: NOTIFY=DA0001T

If coded, a message will appear on the user's TSO terminal indicating if the job abended or got a JCL error. If the job terminates while the user was logged off, the message will appear when the user logs on. If the NOTIFY parameter is omitted, no message will appear when the job terminates.

Instructor Notes:**3.10: NOTIFY Parameter
The JOB Statement – NOTIFY**

- The OS giving your job a number (JobID) when submitted.

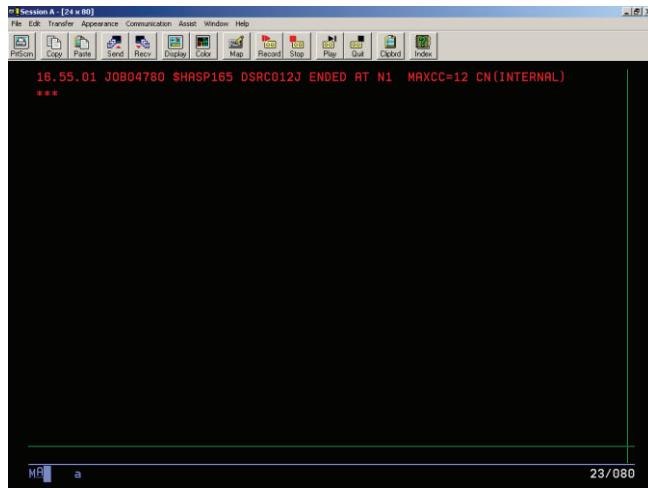


```
EDIT      DSRC012.CICS1.PGMS(CICSCOB) - 01.06          Columns 00001 00072
***** **** * ***** Top of Data **** **** ****
000001 //DSRC012J JOB NOTIFY=DSRC012,CLSS=A,PRTY=15
000002 //I      EXEC PGM=DPHCP1$,PRM= COBOL3',REGION=2048K
000003 /*           USERID.PROJECT.SOURCE(PROGRAM NAME)
000004 //SYSIN   DD DSN=DSRC012.CICS1.PGMS(D12RP01),DISP=SHR
000005 //STEPLIB  DD DSN=CICSTS23.CICS.SDFHLOAD,DISP=SHR
000006 //SYSPRINT DD SYSOUT=
000007 //SYSPUNCH DD DSN=&TOUT,DISP=(PRSS),UNIT=SYSDA,DCB=BLKSIZE=800,
000008 //           SPACE=(400,(400,100))
000009 //C      EXEC PGM=IGYCRCTL,PRM=PPOST,REGION=2048K
000010 //STEPLIB  DD DSN=IGY.SIGCOMP,DISP=SHR
000011 //SYSLIB   DD DSN=CICSTS23.CICS.SDFHC0B,DISP=SHR
000012 //           DD DSN=CICSTS23.CICS.SDFHMAC,DISP=SHR
000013 /*           USERID.PROJECT.DSECT
000014 //           DD DSN=DSRC012.IBM.DSECT,DISP=SHR
000015 /*           USERID.PROJECT.COPYLIB
IKJ56250I JOB DSRC012J(JOB004781) SUBMITTED
***
```

Instructor Notes:

3.10: NOTIFY Parameter
The JOB Statement – NOTIFY

- 'Notification' about completion and completion status.



Instructor Notes:

3.10: NOTIFY Parameter
Demo

Job Parameters



Instructor Notes:**3.11: RESTART Parameter**
Description

Keyword Parameter

Optional

- Format:

RESTART={stepname|procexec.stepname|*}

- The RESTART parameter requests that a job begin its execution with a step other than the first one

Example

**RESTART=step3
RESTART=(0,LE)****General Syntax:****RESTART={stepname|procexec.stepname| *}** keyword parameter

Stepname – The name of the step where execution is to begin.

procexec.stepname – The name of the EXEC statement invoking a procedure and the name of the step within the procedure where execution is to begin.

- Indicates that execution of the job is to begin with the first step and is the default.

Things to avoid:

Duplicate names for EXEC statements invoking procedure

If RESTART=procexec.stepname is used, the first procexec found is used.

Duplicate stepnames within procedure.

If RESTART=procexec.stepname is used, the first stepname within procexec found is used.

Duplicate stepnames.

If RESTART=stepname is used, the first stepname found is used.

EXEC statements (invoking procedures or any step) without names.

No restart is possible.

Instructor Notes:**3.12: TYPRUN Parameter Description**

Keyword Parameter

Optional

- Format

TYPRUN={HOLD|JCLHOLD|SCAN|COPY}

- The TYPRUN parameter requests special processing for the job.

Example

TYPRUN=SCAN

- Checks the JCL for syntactical JCL errors.

General Syntax

TYPRUN={HOLD|JCLHOLD|SCAN|COPY} keyword parameter

HOLD - Job is held held (and not executed temporarily) until the operator uses a command to release. A job is held in the input queue only if syntactically correct.

JCLHOLD (JES2 only) – Job is held (and not executed) until the operator uses a command to release it. Note the job is held in queue even if it is syntactically incorrect. This option is rarely used.

SCAN – Job is scanned for all syntactical JCL errors but will not execute.

COPY (JES2 only)- Job will be printed. No execution and no syntax checking takes place. This option is also rarely used.

The following JOB statement illustrates the use of the parameters relevant to the JOB statement:

```
//DA0001T JOB LA719,PAI,MSGCLASS=A,MSGLEVEL=(1,1),PRTY=5,  
//CLASS=B,REGION=0M,TIME=(0,1),NOTIFY=DA0001T
```

Instructor Notes:

3.12: TYPRUN Parameter
[Lab](#)



Instructor Notes:**Summary**

Accounting Information Parameter and programmer's name are positional parameters.

MSGLEVEL parameter specifies whether the submitted JCL or JCL-related messages should be shown on the job's output.

MSGCLASS parameter assigns a sysout class to the Job log.

CLASS parameter assigns a class to a job.

PRTY parameter determines the scheduling priority of a job in relation to other jobs in the job input queue of the same class.

TIME parameter specifies the total amount of CPU time that all steps in a job can use collectively.

**Summary**

Instructor Notes:**Summary**

REGION parameter specifies the limit of available storage for each of the steps.

ADDRSPC parameter specifies if the job uses real or virtual storage.

NOTIFY parameter informs a TSO user when his or his job terminates.

RESTART parameter requests that a job begin its execution with a step other than the first one.

TYPRUN requests special processing for the job.

**Summary**

Instructor Notes:



Job Control Language (JCL)

Lesson 04 The EXEC Statement

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential.
For Capgemini only.

Instructor Notes:**Lesson Objectives**

On completion of this lesson, you will be able to:

- Explain use of the EXEC statement
- Explain use of the PGM, REGION, TIME, ADDRSPC, ACCT COND parameters & IF/Else/EndIf



Instructor Notes:**4.1: The EXEC Statement**
Introduction

Function: An EXEC statement identifies the step during the reading process when a job is submitted to the system.

It defines the STEP and STEP level information to the system.

Code Snippet:

```
//[stepname] EXEC parameters
```

Remark:

- Stepname is optional. When the stepname is omitted, no reference can be made to that step. A job can contain a maximum of 255 EXEC statements.

Introduction to the EXEC Statement:

An EXEC statement identifies a step during the reading process when a job is submitted to the system. When an EXEC is found, the system accepts all the JCL statements that follow the step, until a delimiter is found. There are four possible delimiters for a step during the reading process:

Another EXEC statement in the input stream (it signals the end of [reading] one step and the beginning of [reading] of another)

A JOB statement

A null statement, that is, // (all JCL statements will be ignored except for a JOB statement)

End-of-file on the reading device; this means, there are no more statements to read.

General Syntax

```
//[stepname] EXEC parameters keyword parameter
```

Stepname: This is optional. When the **stepname** is omitted, no reference can be made. A job can contain not more than 255 steps.

Instructor Notes:**4.1: The EXEC Statement**
Step Name

Must start from column 3 following two //

Stepname is of 1 to 8 character alphanumeric or national character & UNIQUE

Optional but mandatory in

- Overriding
 - STEPNAME.DDNAME
- Restart=STEPNAME
- Referback
 - *.STEPNAME.DDNAME

Instructor Notes:**4.2: The PGM Parameter**
Using The PGM Parameter

Function: The PGM parameter specifies the name of the program to be executed in a step.

Code Snippet:

```
PGM=program-name(load-module)
```

Remark:

- The program specified in PGM is always a member of a PDS which is called a load library or an executable program library.

Using The PGM Parameter:

The PGM parameter identifies the program to be executed in a step.

General Syntax

```
PGM=pgmname           positional parameter
```

Pgmname: Name of the program to be fetched from the load library and executed.

The program specified in **PGM** is always a member of library (PDS). This library is commonly known as an executable program library or a load library. The **EXEC** statement can identify only the member. It has no parameter available to identify the library. If necessary, this must be done by using a **JOBLIB** or a **STEPLIB DD** statement.

Instructor Notes:**4.2: The PGM Parameter
Using The PGM Parameter in EXEC - Example**

Code Snippet

```
//Stepname    EXEC PGM=xyz  
//Steplib     DD DSN=DA0001T.LIB.LOAD,DISP=SHR
```

Or

```
//DA0001TA   JOB .....  
//joblib      DD      DSN=DA0001T.LIB.LOAD,DISP=SHR
```

Using The PGM Parameter in EXEC – Example:

```
//DA0001TA   JOB LA2719,CG,MSGLEVEL=(1,1),NOTIFY=&SYSUID  
//JOBLIB      DD DSN=DA0001T.LIB.LOADLIB,DISP=SHR  
//S1          EXEC PGM=ASSIGN1
```

OR

```
//DA0001TA   JOB LA2719,CG,MSGLEVEL=(1,1),NOTIFY=&SYSUID  
//S1          EXEC PGM=ASSIGN1  
//STEPLIB     DD DSN=DA0001T.LIB.LOADLIB,DISP=SHR
```

Instructor Notes:**4.3: Use of Other Parameters in EXEC
Using Other Parameters**

If neither JOBLIB nor STEPLIB is coded, system searches certain predefined libraries (that is, system default libraries). If the specified member is found, it is executed. If not found, the result is S806 ABEND failure.

The following keyword parameters can be specified at the EXEC statement. The parameters are REGION, ADDRSPC, TIME, COND, ACCT, and PARM.

Instructor Notes:**4.3: Use of Other Parameters in EXEC
Using Other Parameters (Contd.)**

When REGION and ADDRSPC parameter are coded in both the JOB and EXEC statements within a job, the value of the JOB statement will be used.

If TIME parameter is coded in both the JOB and EXEC statements within a job, both will be in effect. Either of them can cause a S322 ABEND failure.

These parameters will be used for that step only.

Example

```
//DA0001TA EXEC PGM = PAYPGM1, PARM = '0791'
```

Instructor Notes:

4.3: Use of Other Parameters in EXEC
Demo

EXEC Parameters



Instructor Notes:**The REGION Parameter:**

This parameter specifies the available storage limit for the step within address space of the job.

- **General Syntax**

REGION=value{K|M} - keyword parameter

Value : 1 to 2096128 if K (1024 bytes) is used. It should be an even number; it will be rounded to the next higher even number.

1 to 2047 if M (1024K or 1048576 bytes) is used. M is not available to **MVS/SP**, only to **MVS/XA** and **MVS/ESA**.

If the **REGION** parameter is omitted, the **REGION** parameter in the **EXEC** statements within the job is used. If the **REGION** parameter is coded in neither the **JOB** nor the **EXEC** statement, an installation-defined default is used. The default value of most installations is between 500K and 1000K.

If the **REGION** parameter is coded in both the **JOB** and an **EXEC** statement within the job, the value in the **JOB** statement is used.

The **REGION** parameter in the **JOB** statement is used much more often than the one in the **EXEC** statement. Coding the same value for all steps would have the same effect as the **REGION** parameter in the **JOB** statement.

- **Example :**

```
//DA0001TA JOB LA2719,CG,MSGLEVEL=(1,1),NOTIFY=&SYSUID  
//S1      EXEC PGM=ASSIGN1,REGION=500K  
//STEPLIB  DD DSN=DA0001T.LIB.LOADLIB,DISP=SHR
```

Instructor Notes:**The TIME Parameter:**

This parameter specifies the total amount of CPU time that the step is allowed to use.

- **General Syntax**

TIME=([minutes][,seconds] | [1440]) keyword parameter

Minutes: a number from 1 to 1439

Seconds: a number from 1 to 59

1440: The step is not timed for CPU. Note that **TIME=1440** is rarely used, and most installation disallow its use in a testing environment. **TIME=1440** should be used by an on-line system such as **CICS** OR **ADS/O**.

- **Use:**

- When the **TIME** parameter is omitted, an installation-defined default is used. This default is usually very high and can not cause an **S322 ABEND** failure.
- If the **TIME** parameter is also coded in the **JOB** statement, both will have effect and either can cause a **S322 ABEND** failure. It is not advisable to use them both.

- **Remark:**

It is possible for a step to get more CPU time than that is specified in the **TIME** parameter or the default by a maximum 10.5 seconds. This is due to the fact that the system checks for violations every 10.5 seconds.

- **Example:**

```
//DA0001TA JOB LA2719,CG,MSGLEVEL=(1,1),NOTIFY=&SYSUID  
//S1      EXEC PGM=ASSIGN1,REGION=500K,TIME=(,3)  
//STEPLIB  DD DSN=DA0001T.LIB.LOADLIB,DISP=SHR
```

Instructor Notes:**4.4: The ADDRSPC Parameter**
Using the ADDRSPC Parameter

The characteristics significant for use are as below:

- ADDRSPC is a keyword parameter.
- It is optional.
- It is installation-dependent.

Code Snippet:

```
ADDRSPC={VIRT|REAL}
```

Remarks:

- This is the rarely used parameter because of the default. Note that ADDRSPC=REAL is a parameter that is disallowed in practically all installation because it can cause serious performance problems for other jobs.

This parameter cannot override a specific ADDRSPC coded on the job statement, but may override the system default.

If ADDRSPC needs to be overridden for a given step of a called procedure this can be done by the inclusion of a procstepname parameter.

Syntax:

```
ADDRSPC [.procstepame] = {VIRT / REAL}
```

VIRT: Request virtual storage. The system can page the step

REAL: Request real storage. The system cannot page the job step and must place the job step in real storage

Instructor Notes:**4.5: The ACCT Parameter
Introduction**

The characteristics significant for use are as below:

- Keyword parameter
- Optional
- Installation-dependent

Code Snippet:

```
ACCT=(account-no,[additional account-info.])
```

Remarks:

The ACCT parameter specifies accounting information to be used for the step as opposed to the Accounting information in the JOB statement.

The ACCT Parameter:

The parameter specifies accounting information to be used for the step as opposed to the accounting information in the JOB statement.

General Syntax:

```
ACCT=(acctno [,additional-acct-info]) keyword parameter
```

Acctno: The account number to be used for the step

additional-acct-info: same as in the JOB statement but without any JES2 meaning

Use:

The ACCT parameter is seldom used, and when it is, generally only the account number is displayed. This is used to charge resource utilization for a step to a different account number other than the one coded in the JOB statement. If an account number is also coded in the JOB statement, the account number in the EXEC statement is used.

Using the ACCT Parameter (Contd.):

Example:

```
//DA0001TA JOB LA2719,CG,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//S1      EXEC PGM=IEFBR14,ACCT=('es0013,hr4200,iefbr14')
//DD1      DD DSN=DA0001T.SHEELA.EMPFILE,disp=(MOD,DELETE),
//                  SPACE=(TRK,0),UNIT=SYSDA
```

Instructor Notes:**4.6: The PARM Parameter**
Using the PARAM Parameter

The characteristics significant for use are as below:

- Keyword parameter
- Optional
- Installation-dependent

Code Snippet:

PARM=string (a max. of 100 characters)

Remarks: The PARM parameter provides a way to supply data of limited size to the executing program.

Note : If commas are parts of the string, the entire field must be enclosed in parenthesis or apostrophe's.

Using the PARM Parameter:

This parameter provides a way to supply data of limited size to the executing program

General Syntax

PARM=string keyword parameter

String: A string of characters up to 100. If commas are a part of the string, the entire field must be enclosed in parentheses (or apostrophes). If any portion of the string contains special characters (other than hyphen), that portion of the entire string must be enclosed in apostrophes.

Note: Any parentheses that are used are counted as characters, whereas apostrophes do not. All information after the “=” in the PARM parameter, excluding apostrophes, is saved by the system within the step’s own space. When the program begins execution by using the appropriate instructions, it can find the saved information in the storage space.

Instructor Notes:**4.6: The PARM Parameter**
Using the PARM Parameter - Examples

In COBOL the following must be coded:

1. PARM=(A,B,C,D) or 'A,B,C,D'
2. PARM=1005

```
LINKAGE SECTION.  
01 PARM.  
    05 plength    PIC S9(4) COMP.  
    05 string      PIC X(100).  
PROCEDURE DIVISION USING PARM.
```

- plength contains the length of the data passed to the program and data is placed in the variable string.

Instructor Notes:Using the PARM Parameter – Examples:

```
//DA0001TA JOB LA2719,CG,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//S1      EXEC PGM=ASSIGN2,PARM='G2 ',TIME=(,1)
//STEPLIB  DD   DSN=DA0001T.LIB.LOADLIB,DISP=SHR
//INFILE   DD   DSN=DA0001T.EMPFILE,DISP=SHR
```

-Access the data in a Cobol program:

Identification Division.
 Data Division.
 Linkage Section.
01 Parm-data-Area.
 05 Parm-len pic s9(4) comp.
 05 Parm-data1.
 10 Data1-in X(100).
Procedure Division using Parm-Data-Area.
 Display Data1-in.

Parm-len contains the length of the data passed to the program and data is placed in the variable string.

Using the PARM Parameter – Examples:
Rules for continuation

```
//DA0001TA JOB LA2719,CG,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//COB      EXEC PGM=IKFCBL00,REGION=1024K,
//           PARM=(notrunc,nodynam,lib,size=4096k,buf=116k',
//           'apost,nores,seq')
```

OR

```
//DA0001TA JOB LA2719,CG,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//COB      EXEC PGM=IKFCBL00,REGION=1024K,
//           PARM=(notrunc,nodynam,lib,'size=4096k','buf=116k',
//           'apost,nores,seq')
```

Note that an expression in quotes cannot be continued, we need to enclose the string in parentheses and field containing special characters in apostrophes.

Ex.1 PARM='29/06/00' or ('29/06/00')
 Ex.2 PARM=(A,B,C,D) or 'A,B,C,D'
PARM parameter (Contd ...)
 Pass data vide PARM in the JCL:

```
//Step010 EXEC PGM=Load Module, PARM='Hello World'
```

Coding Rules:

Use apostrophes around the passed data if the data consists of a space

PARM='001 JOHN'

If passed data comprises an apostrophe or an ampersand, code two such characters consecutively.

PARM="Hello World"
 PARM='ABC&&4'

To continue coding data on the next line, code a comma and continue. Also coding apostrophes around continued data is a must. Comma gets included in the byte count.

PARM='001,
 JOHN'

Page 04-16

To code till column 71 and then continue, leave column 72 blank and continue on the next line from column 16

Instructor Notes:**4.7: The COND Parameter**
Introduction

The characteristics significant for use are as below:

- The COND parameter causes conditional execution of steps within a job.
- It can be coded in the JOB or EXEC statement or both.
- Return code (or Condition code) is a number between 0 and 4095, issued by an executing program just before the execution is finished.

Introduction:

Characteristics of the COND parameter are as follows:

The COND parameter can be coded in the JOB as well as the EXEC statement. It is mostly used in the EXEC statement. The main tool for controlling the execution of steps within a job is the COND parameter.

A Return (or Condition) code:

A return code is a number between 0 and 4095, issued by an executing program just before its execution is finished.

It is intended to identify an important event found (or not found) during the execution. For example, a program may issue a return code of 21 to indicate that a problematic event (such as a record is out of sequence) was detected during the execution. Alternatively, a program may issue a return code of 0 to indicate that the execution was trouble-free.

The return code issued by a program is saved by the system for the duration of the job. Any subsequent step of the same job can interrogate this return code by using the COND parameter either in the JOB or EXEC statement. The result of this interrogation is to permit or bypass the execution of the step.

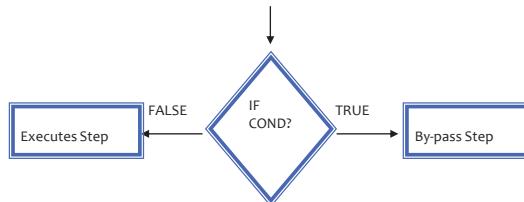
Note: The return code is never available to a job other than the one which issued it. In other words, the step that interrogates the return code must be in the job same as the step that issued it, and it should be subsequent to the step that issued the return code.

Instructor Notes:

4.7: The COND Parameter

The JOB Statement – COND

The COND parameter states that if a condition is true then the step in which this COND parameter is coded is to be bypassed. If the condition is false than the step is to be executed.



The JOB statement COND parameter performs the same return code tests for every step in a job. If the JOB statement's return code test is satisfied, the job terminates.

The JOB COND parameter performs its return code tests for every step in the job, even when EXEC statements also contain COND parameters.

If any step satisfies the return code test in the JOB statement, the job terminates. The job terminates regardless of whether or not any EXEC statements contain COND parameters and whether or not an EXEC return code test would be satisfied.

If the JOB statements return code test is not satisfied, the system then checks the COND parameter on the EXEC statement for the next step. If the EXEC statement return code test is satisfied, the system bypasses that step and begins processing of the following step, including return code testing.

Instructor Notes:**4.7: The COND Parameter
Using COND in the JOB Statement**

Code Snippet:

```
COND=((code, operator) [,....])
    code = number between 0 to 4095
    operator = LT, GT, LE, GE, NE, EQ
```

Rules:

- Condition is read from left to right.
- Maximum of 8 tests.
- If COND evaluates to TRUE the step is BYPASSED.
- If COND evaluates to FALSE the step is EXECUTED.

Using COND in the JOB Statement:

IBM-established conventions are as follows:

Return code of 0 indicates a complete success.

Return code of 4 indicates a warning. The warning is benign, so a return code will normally be treated as acceptable.

Return code of 8 indicates questionable results.

Return code of 12 indicates bad results.

Return code of 16 indicates a terminal condition.

Example

There can be a maximum of eight tests in the COND parameter. Condition is evaluated from left to right and if a test is satisfied, the job stops execution at that point.

```
//Step010 Exec Pgm=CBL1
//Step020 Exec Pgm=CBL2,Cond=(0,LT,Step010)
Pgm CBL2 is not executed if pgm CBL1 returns any CC >
0000
//Step030 Exec Pgm=CBL3,Cond=(0,Eq,Step010)
Pgm CBL3 is not executed if pgm CBL1 returns CC 0000
```

Instructor Notes:

4.7: The COND Parameter

Using COND in the JOB Statement - Example

Consider a job with five steps. Assume that none will ABEND.

```
//DA0001TA    JOB    LA2719,CG,COND=((12,LT),(8,EQ))
```

- STEP1 issues a return code of 0
- STEP2, if executed, issues a return code of 4
- STEP3, if executed, issues a return code of 16
- STEP4, if executed, issues a return code of 0
- STEP5, if executed, issues a return code of 4

Using COND in the JOB Statement:

STEP 1 is executed by default, since no previous return codes exist, the COND parameter in the JOB statement is ignored for the first step.

Before STEP2 begins execution, the system interrogates the existing return code (0), using the tests in the COND parameter and reading the test from left to right,

Is 12 less than 0? The answer is "no". The first test of the COND parameter is not satisfied. The second test is tested.

Is 8 equal to 0? . The answer is "no". Neither of the two tests is satisfied, and therefore, STEP2 is executed.

Before STEP3 begins execution, the system interrogates the existing return codes (0 and 4), using the tests in the same COND parameter. Since the result for return code 0 is already known, only 4 is tested:

Is 12 less than 4? The answer is "no". The first test of the COND parameter is not satisfied. The second test is tested.

Is 8 equal to 4 . The answer is "no". Neither of the two tests is satisfied, and therefore, STEP3 is executed.

Before STEP4 begins execution, the system interrogates the existing return codes (0 , 4 and 16), using the tests in the same COND parameter. Since the results for return code 0 and 4 are already known, only 16 is be tested:

Is 12 less than 16? The answer is "yes". The first test of the COND parameter was satisfied. There is no need for the second test.

Executions of the job stops. STEP 4 and the remaining steps do not get executed.

A message is displayed as the output: IEF2011 DA0001TA STEP4-JOB TERMINATED BECAUSE OF CONDITION CODES.

Instructor Notes:**4.7: The COND Parameter
Using COND in the EXEC Statement**

Code Snippet:

```
COND=((code,operator[,stepname])[,...][,EVEN/ONLY])
```

- EVEN requests that execution be permitted even though any previous step has ABENDED (If no step has ABENDED then the EVEN condition is ignored)
- ONLY requests that execution be permitted only if previous step has ABENDED (If no step ABENDs the COND parameter is ignored)

Using COND in the EXEC Statement:

The COND parameter can perform a test (or multiple tests) before a step begins execution against the return (condition) codes issued by previous steps. If a test is satisfied (reading from left to right), the step is not executed.

General Syntax

```
COND=((code,operator[,stepname])[,(code,operator[,stepname])]...  
[,EVEN|ONLY])
```

keyword parameter

Code : This is a number between 0 and 4095.

Operator: This provides a comparison between a return code and the code. There are six operators: LT, LE, NE, EQ, GT, GE

Stepname: This identifies the name of the preceding step whose return code is interrogated. It can also appear as two names proceexec.stepname where proceexec identifies the name of the EXEC statement invoking a procedure and "stepname" the stepname within the procedure.

EVEN : This requests that execution be permitted even though a previous (any previous) step has ABENDED.

ONLY : This requests that execution be permitted only if a previous (any previous) step has ABENDED.

Instructor Notes:**4.7: The COND Parameter
Using COND in the EXEC Statement (contd.)**

If the COND parameter is coded in both the JOB and EXEC statements, the COND parameter of the JOB statement is tested first and then the COND of the EXEC statement is tested.

Using COND in the EXEC Statement (Contd.):

There can be a maximum of eight tests in the COND parameter. EVEN or ONLY count toward eight. Condition is evaluated from left to right and if a test is satisfied, only that step is not executed.

Remark:

EVEN and ONLY cannot make reference to a particular step. They refer to any previous step that has ABENDED.

EVEN and ONLY are mutually exclusive.

EVEN and ONLY have no positional significance. Each can be coded anywhere in the COND parameter in relation to other tests.

Following an ABEND failure, a step cannot be executed unless it contains EVEN or ONLY in the COND parameter of its EXEC statement.

The first step is always executed unless COND=ONLY appears in the EXEC statement.

COND=ONLY causes the first step to be bypassed, since no previous ABEND failures have occurred. Any other COND parameter in the first EXEC statement is ignored (that is, COND=(4,LT) or COND=EVEN) or results in JCL error.

(that is, COND=(5,LT,stepname)) The reason is, there are no previous step.

A step that is not executed issues no return code because a program responsible for issuing the return code was not even loaded into the storage. As a result, no return code exists. An attempt to interrogate the return code of such a step in the COND parameter of a subsequent step is ignored.

Instructor Notes:**4.7: The COND Parameter**
Using the COND Parameter

- The following table will help you in solving queries related with the COND parameter.

COND	JOB	EXEC
TRUE	TERMINATE	BYPASSED
FALSE	CONTINUES	EXECUTED

Using The COND Parameter:

A step that ABENDs (carries out ABENDING) issues ‘no return code’ because a program always issues a return code (conditionally or by default) if it reaches the end of its execution and intentionally returns control to the system. When an ABEND occurs, the program loses control instantly, and is evicted from execution by the system. This results in a specific effect: when a step that ABENDs ‘no return code’ exists (a completion code exists), an attempt to interrogate the return code of such a step in the COND parameter of a step is ignored until it contains EVEN or ONLY.

Using COND, JOB, EXEC, and PGM:

If the COND parameter is coded neither at the JOB nor at the EXEC statement:

The step is executed regardless of previous return codes.
However, this does not happen if the previous step has ABENDED.

If the COND parameter is coded in both the JOB statement as well as in the EXEC statement within the JOB:

Both are tested.

The COND parameter of the JOB statement is tested first.

If none of its tests are satisfied, then the COND parameter of the EXEC statement is tested.

If a test is satisfied, none of the steps from that point onwards are executed.

Instructor Notes:

**4.7: The COND Parameter
IF/THEN/ELSE/ENDIF**

IF/THEN/ELSE/ENDIF statement construct determines for conditionally executing one or more steps.

Nesting is possible up to 15 levels.

If the coded condition is true, the following steps till else will be executed and if the condition is false then the steps coded on the else part will be executed.

Do not specify JOBLIB, JCLLIB,, JOBCAT, STEPCAT, JOB within the THEN or ELSE scope of IF statement.

Relational expression field consists of:

Comparison operators - GT, LT, NG, NL, EQ, NE, GE, LE
 > < \neg > \neg < = \neg = \geq \leq

Connect operators – AND(&), OR(|),NOT (\neg) operators

Keywords:

RC	Indicates Return code
ABEND	Indicates occurrence of Abend
\neg ABEND	Indicates non occurrence of Abend
ABENDCC	Indicates a specific system or user abend code
RUN	Indicates execution of the specified Step
\neg RUN	Indicates non execution of the specified step

Example 1:

This example tests the return code for a step.

```
//RCTEST IF (STEP1.RC GT 20|STEP2.RC = 60) THEN
//STEP3 EXEC PGM=U
//ENDTEST ENDIF
//NEXTSTEP EXEC
```

The system executes STEP3 if:

The return code from STEP1 is greater than 20, or the return code from STEP2 equals 60.

If the evaluation of the relational expression is false, the system bypasses STEP3 and continues processing with step NEXTSTEP.

Example 2:

```
//
// DSRP039A JOB (),NOTIFY=&SYSUID
//STEP01 EXEC PGM=IEBGENER
//XYZ IF (STEP01.RC=0) THEN
//STEP02 EXEC PGM=IEBCOPY
//XYZ ELSE
//STEP03 EXEC PGM=IEFBR14
//XYZ ENDIF
//
```

Instructor Notes:**4.7: The COND Parameter
IF/THEN/ELSE/ENDIF (contd..)**

Syntax

```
//Name IF (Relational expression) THEN
// Steps to be processed when relational expression is true
//Name ELSE
// Step to be processed when relational expression is false
//Name ENDIF
```

- Name field is optional.
- Operation fields are IF,THEN, ELSE and ENDIF.
 - ELSE is an optional clause.
 - ENDIF marks the end of the statement

Instructor Notes:4.8: Using the COND, JOB, EXEC, and PGM Parameters
Example1

```
//DA0001TA      JOB .....  
//S1    EXEC   PGM=P1           (4)  
//S2    EXEC   PGM=P2,  
//          COND=(0,LT,S1),EVEN) (12)  
//S3    EXEC   PGM=P3,COND=(8,LT,S2) (0)  
//S4    EXEC   PGM=P4,COND=(4,LT)     (8)  
//S5    EXEC   PGM=P5,  
//          COND=((4,LT,S1),(0,LT,S3)) ABEND  
//S6    EXEC   PGM=P6,  
//          COND=((EVEN,(0,LE,S5)) (16)  
//S7    EXEC   PGM=P7,  
//          COND=((0,LT,S1),(12,LT,S3)) (0)  
//S8    EXEC   PGM=P8,  
//          COND=(16,EQ,S6),ONLY) (0)  
//S9    EXEC   PGM=P9, COND=EVEN (4)  
//S10   EXEC   PGM=P10, COND=ONLY (0)
```

Instructor Notes:4.8: Using the COND, JOB, EXEC, and PGM Parameters
Example2

```
//DA0001TA      JOB      LA2719,CG,COND=(10,LT)
//STEP1  EXEC    PGM=AAA          (6)
//STEP2  EXEC    PGM=BBB,
                           COND=((2,EQ),(4,EQ))          (2)
//STEP3  EXEC    PGM=CCC, COND=ONLY          (4)
//STEP4  EXEC    PGM=DDD,
                           COND=(5,GT,STEP1),(2,EQ))          (6)
//STEP5  EXEC    PGM=EEE          (9)
//STEP6  EXEC    PGM=FFF,          (10)
                           COND=((8,GT,STEP5),EVEN)
//STEP7  EXEC    PGM=GGG,
                           COND=(4,GT,STEP4)          (12)
//STEP8  EXEC    PGM=HHH
//STEP9  EXEC    PGM=III, COND=ONLY          --
```

Instructor Notes:**Summary**

The PGM parameter identifies the program to be executed in a step.

The REGION parameter specifies the limit of available storage for the step.

The TIME parameter specifies the total amount of CPU time that the step is allowed to use.

The ADDRSPC parameter specifies if the step will use real or virtual storage.

The ACCT parameter specifies accounting information to be used for the step.

The PARM parameter provides a way to supply data of limited size to the executing program.

Controlling the execution of steps within a job is done by specifying the COND parameter.

**Summary**

Instructor Notes:



Job Control Language (JCL)

Lesson 05 The DD Statement

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential.
For Capgemini only.

Instructor Notes:**Lesson Objectives**

On completion of this lesson, you will be able to:

- Use DD statement
- Use DSN, DISP, UNIT, VOL, SPACE, LABEL, DCB, SYSOUT, SYSIN,SYSPRINT, DUMMY parameter
- Use JOBLIB and STEPLIB statement
- Use of JobCat and StepCat
- Request storage dump



Instructor Notes:

[5.1: Function of The DD Statement](#)
Introduction

Data Definition (DD) statement describes the datasets that must be allocated.

At least one DD statement for each dataset is necessary in the program to read from/write to dataset.

DD statements are coded after the EXEC statement that identifies the job step.

Introduction:

A DD (Data Definition) statement must appear in a step when the executing program expects to read from or write to a dataset. In other words, DD statement describes the dataset.

The maximum number of DD statements in a step is 3273. The DD statement can be coded in any order and always appears after the EXEC statement with the exception of JOBLIB, JOBCAT, and PROCLIB DD statement.

Instructor Notes:**5.1: Function of The DD Statement**
DD Statement - DASD Dataset Format

```
//ddname DD DSNAME=data-set-name,  
//           DISP=(status, normal-disp, abnormal-disp),  
//           UNIT=unit,  
//           VOL=SER=Vol-Ser,  
//           SPACE=(unit,(primary,secondary),dir),  
//           DCB=(option, option,...)
```

As job executes the system performs device and space allocation for each ddname specified.

Each ddname must be unique & 1 to 8 characters long comprising of alphanumeric or national character & first character must be alphabetic or national character. If there are duplicate ddnames, even though the system will make the allocation specified, it will direct all related messages to the first ddname.

AVOID using ddnames that begin with 'SYS', 'JOB', 'STEP' are considered as reserve words for IBM products. JOBLIB, JOBCAT, SYSOUT, SYSIN, STEPLIB, STEPCAT, SYSUDUMP, STEPLIB, SYSABEND, SYSDBOUT
As job executes the system performs device and space allocation for each ddname specified.

Each ddname must be unique & 1 to 8 characters long comprising of alphanumeric or national character & first character must be alphabetic or national character. If there are duplicate ddnames, even though the system will make the allocation specified, it will direct all related messages to the first ddname.

AVOID using ddnames that begin with 'SYS', 'JOB', 'STEP' are considered as reserve words for IBM products. JOBLIB, JOBCAT, SYSOUT, SYSIN, STEPLIB, STEPCAT, SYSUDUMP, STEPLIB, SYSABEND, SYSDBOUT

Instructor Notes:**5.1: Function of The DD Statement****DD Statement**

Classified into Permanent and Temporary datasets

- Permanent datasets are used as a data-store on disks and tapes.
- Temporary datasets are used as work areas within a job; they last only for the duration of the job.
- A Temporary dataset name starts with && followed by 1 to 6 characters - DSN=&&TEMPRY
 - Absence of DSN coding also denotes a temporary dataset.
- A DSN coding without && indicates a Permanent dataset.

If the dataset (DSN) is not found in the system catalog, the system throws a "JCL error".

Dataset names (DSN) cannot be duplicated in the VTOC. If duplicated, the system returns a "CC of 8".

Instructor Notes:**5.2: The DSNAMe Parameter
Characteristics**

It is optional.

It is always required for a permanent dataset.

DSNAME is a keyword parameter.

It supplies the dataset name because it is stored in the datasets label or the file catalog's entry.

DSN (or DSNAME) parameter identifies the name of the dataset to be created.

Characteristics:

The DSN (or DSNAME) parameter identifies the name of the dataset to be created or retrieved.

General Syntax**DSN=name| NULLFILE | referback**

- keyword parameter

Name: This could be a qualified name. This name consists of two or more simple name separated by periods for a maximum of 44 characters.

For example (i) **DSN=DA0001T.CG.EMPFILE**For example (ii) **DSN=DA0001T.CG.COBOL(ASS1)**

This describes a sequential dataset, that means, ASS1 is a member of PDS/library DA0001T.CG.COBOL

For example (iii) **DSN=&&name**

A simple name preceded by two ampersands identifies a temporary dataset. It is considered temporary because it is not retained beyond job termination.

Instructor Notes:**5.2: The DSNAME Parameter
Format and Example**

Format:

DSN=name

Example:

DSN=DA0001T.CG.PAYROLL.MASTER

To refer to a member of a PDS:

//TRANFILE DD DSN=DA0001T.CG.TRANFILE(mem1)

Instructor Notes:**5.2: The DSNAME Parameter
Format and Example (Contd.)****Format****DSN=&&name**

- Identifies a temporary dataset.
- Temporary means that it cannot be retained beyond job termination.

Format and Example (Contd.):

The system generates a name with the following format:

SYSyyddd.Thhmmss.RV001.jobname.name

yyddd – date as per Julian calendar;
hhmmss – uses 24-hour clock. It is the time of JOB initiation (beginning of JOB execution)

RV001 –system provided information in reference to the reader;
jobname – as it appears in the JOB statement;
name – whatever is coded after &&.

For e.g. DSN=&&temp. The system generates the following name:

SYS03173.T090000.RV001.DA0001TA TEMP

Instructor Notes:**5.2: The DSNAME Parameter
Example**

This also creates a temporary dataset.

```
//DD1 DD DSN=&&TEMP,UNIT=SYSDA,  
// SPACE=(TRK,(2,1),RLSE),  
// DISP=(,PASS,DELETE)
```

```
//SORTWK1 DD UNIT=SYSDA,SPACE=(TRK,(2,1),RLSE),  
// DISP=(,PASS,DELETE)
```

Example:

If the DSN name is omitted from a DD statement (except DD *, SYSOUT and DUMMY), it also indicates a temporary dataset. However, the system generates a name with the following format:

```
SYSyyddd.Thhmmss.RV001.jobname.R0000001  
//SORTWK1 DD UNIT=SYSDA,SPACE=(TRK,(1,2),RLSE)  
SYS00173.T100000.RV001.DA0001TA.R0000001
```

This form is basically used when a step requires a work dataset (a dataset created at the beginning of the step's execution and deleted at the end). Mostly used in utilities.

Instructor Notes:**5.2: The DSNAMES Parameter Formats**

Referback formats

- *.**STEPNAME.DDNAME** : This requests that the dataset name is to be copied from DD statement 'ddname' found in a previous step 'stepname'.

DSN=*.STEP2.OUT

- *.**DDNAME** : This requests that the dataset name is to be copied from a previous DD statement 'ddname' found in the same step. This is seldom used.

DSN=*.DD1

REFERBACK: Backward reference (Referback) is used to copy information from a previous DD statement (within the same job) thus simplifying JCL code. PGM can be Referback. On the DD statement, this can be applied to three parameters:

DSN
VOL
DCB

Referback on a DD statement

```
//IGTRN01A   JOB    NOTIFY=DSRC012
//STEP010    EXEC    PGM=PGM1
//DD1        DD      DSN=EMP.PDS,DISP=(NEW,CATLG),
//                  DCB=(LRECL=80,RECFM=FB,BLKSIZE=800),
//
//DD2        DD      DSN=INP.PDS,DCB=*.DD1, .....
//STEP2    EXEC    PGM=PGM2
//DD2        DD      DSN=STU.PDS, DCB=*.STEP010.DD1, .....
```

Instructor Notes:

5.2: The DSNAME Parameter
Formats (contd..)

Referback on a PGM statement

```
//LKED EXEC PGM=HEWL,REGION=1024K,PARM='XREF,LIST'  
//SYSLIB DD DSNAME=CEE.SCEELKED,DISP=SHR  
//SYSPRINT DD SYSOUT=A  
//SYSLIN DD DSNAME=DSRC746.SATYA.OBJ(ADD1),DISP=  
//SYSLMOD DD DSNAME=DSRC746.SATYA.LOAD(ADD1),DISP  
//SYSUT1 DD UNIT=SYSDA,SPACE=(TRK,(10,10))  
/*  
//GO EXEC PGM=*.LKED.SYSLMOD  
//SYSPRINT DD SYSOUT=A  
//SYSOUT DD SYSOUT=A  
//SYSIN DD *  
10  
/*
```

Instructor Notes:**5.2: The DSNAME Parameter
Formats (contd.)**

- *.PROCEXEC.STEPNAME.DDNAME : Requests that the dataset name be copied from DD statement 'ddname' found in a previous step 'stepname' found within procedure 'procexec'.

```
DSN=*.PS4.STEP2.OUT
```

Instructor Notes:**5.3: The DISP Parameter**
Characteristics of the DISP Parameter

DISP is a keyword parameter.

Its use is optional.

Format :

DISP=(status-fld,normal-disp-fld,abnormal-disp-fld)

NEW ,DELETE ,DELETE
OLD ,KEEP ,KEEP
DISP=(SHR ,CATLG ,CATLG)
MOD ,UNCATLG ,UNCATLG
,PASS

Characteristics of the DISP Parameter:

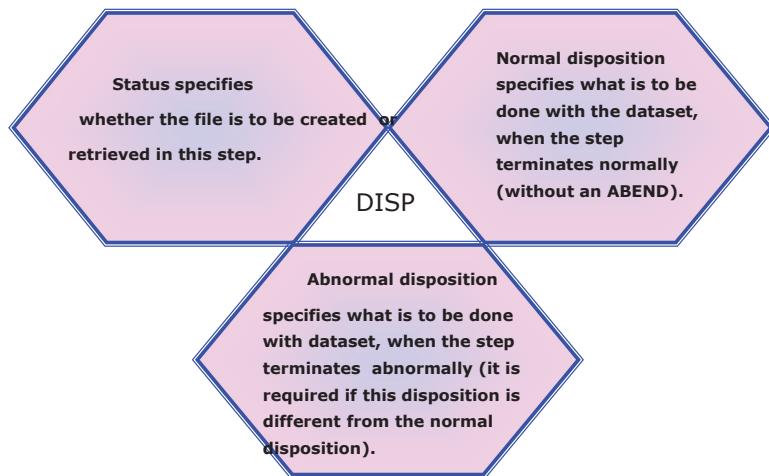
The DISP parameter specifies the following:

It specifies if the dataset is to be created or retrieved.

It indicates how to dispose off the dataset when the step terminates (normally or abnormally).

Instructor Notes:**5.3: The DISP Parameter**
Characteristics of the DISP Parameter

The DISP parameter has three positional sub-parameters:



Instructor Notes:**5.3: The DISP Parameter
Values of the Status Sub-parameter**

NEW indicates that the dataset does not exist and should be created in this step

OLD indicates that the dataset exists and should be allocated for exclusive use

STATUS

MOD indicates that an existing dataset will be retrieved or will be created.

SHR indicates that the dataset exists and should be allocated for shared use.

The status-field: This field tells the system whether the dataset is to be created or retrieved.

NEW – Indicates that the dataset will be created in this step

OLD - Indicates that an existing dataset will be retrieved and demands exclusive control

SHR - Indicates that an existing dataset will be retrieved. It also indicates that this dataset, if on disk, can be shared with one or more other users

(Mnemonic hint: NOMS; N:New, O: OLD, M: MOD, S: SHR)

Instructor Notes:**5.3: The DISP Parameter****The MOD Subparameter**

MOD - This subparameter has two possible meanings:

- It indicates that an existing dataset will be retrieved. This will be true if:
 - The dataset is either cataloged or passed OR
 - The DD statement contains either VOL=SER or VOL=REF
- It indicates that the dataset will be created. This will be true if:
 - The DD statement contains neither VOL=SER nor VOL=REF and it describes a dataset which is neither cataloged nor passed.
 - The DD statement contains VOL=REF referring to a dataset, which is a non-specific request for a new dataset.

The MOD Subparameter:

MOD - This sub parameter has two possible meanings:

Indicates that an existing dataset will be retrieved. This will be true if:

The dataset is either cataloged or passed.

The DD statement contains either VOL=SER or VOL=REF (a VOL VOL=REF referring to a DD statement, which is a nonspecific request for a new dataset, is not included).

Indicates that the dataset will be created. This is true if:

The DD statement contains neither VOL=SER nor VOL=REF and it describes a dataset which is neither cataloged nor passed.

The DD statement contains VOL=REF referring to a DD statement, which is nonspecific, a request for a new dataset.

Instructor Notes:**Example 1:**

```
//DD1 DD DSN=DA0001T.EMPFILE,DISP=(MOD,CATLG),  
//           UNIT=TAPE
```

Explanation:

- The system assumes DA0001T.EMPFILE to be an existing dataset. Since the DD statement contains neither VOL=SER or VOL=REF, the system searches the catalog and gets volume information from the catalog entry. The volume having been found, the dataset will be treated as existing dataset.
- Had the dataset been neither cataloged nor passed, the system would have been unable to find the volume information and MOD will default to new.

Example 2:

```
//DD1 DD DSN=DA0001T.EMPFILE,DISP=(MOD,CATLG),  
//           UNIT=SYSDA,VOL=SER=BS3003,SPACE=(TRK,(1,2))
```

Explanation: Since VOL=SER is specified; the fate of MOD is sealed, whether or not it exists. It will be treated as OLD (with appropriate positioning). If the dataset exists on that volume no problem, however, if it does not exist the result will be S213-04 ABEND failure (i.e. dataset does not exist)

Note: When UNIT and VOL=SER is specified the system does not search the catalog to locate the dataset.

Instructor Notes:**5.3: The DISP Parameter**
Normal and Abnormal DISP-Parameters

These are the values of Normal and Abnormal DISP-parameters:

- DELETE: The dataset is deleted and uncataloged when the step terminates normally (or abnormally).
- KEEP: The dataset is to be retained when the step terminates normally (or abnormally).
- CATLG: The dataset is retained and an entry is made in the catalog when the step terminates normally (or abnormally).
- UNCATLG: The dataset is retained but entry is removed from the catalog when the step terminates normally.

The normal disposition field: This field is used to tell the system how to dispose of the dataset when the step terminates normally (without an ABEND).

Normal and Abnormal DISP-Parameters:

Values of Normal and Abnormal DISP-parameters are as follows: (Mnemonic Hint: DUCK)

DELETE: This indicates that the dataset is to be deleted when the step terminates. For an existing dataset, OLD, SHR or MOD (not defaulting to NEW), the dataset is also uncataloged, if the catalog is used while retrieving the dataset. It only deletes the dataset if the catalog is not used during the retrieval. This means that for a cataloged dataset, if you specify UNIT and VOL=SER, the system does not search the catalog.

Note:

(i) When a tape dataset is deleted, nothing happens. A tape dataset cannot be deleted through the DISP parameter. It is effectively deleted when the dataset is written over.

(ii) A VSAM cluster cannot be deleted by coding DISP=(OLD, DELETE) as it defaults to DISP=(OLD,KEEP).

(iii) A member of PDS cannot be deleted because DISP applies to the entire PDS, and as a result, it deletes the entire PDS. Use either TSO or IEHPROGM utility.

The system always issues a message indicating "DELETED" or "NOT DELETED N", where N indicates the reason for failing.

Instructor Notes:**5.3: The DISP Parameter
Normal and Abnormal DISP-Parameters (Contd.)**

- PASS - Normal disposition only, the dataset is retained for use by a later step.

NOTE

- PASS is not permitted in the abnormal disposition field.

Remark

- If the abnormal disposition field is omitted, the default is the normal disposition field.
- When a dataset specified in DSN does not exist, then
- S213-04 ABEND failure occurs, that is, dataset does not exist.

Normal and Abnormal DISP-Parameters (contd.):

KEEP: This Indicates that the dataset is to be kept when the step terminates. The system takes no action and issues a message indicating the dataset is kept. Also, the system issues a message "KEPT". Note that "NOT KEPT" message does not exist.

Note: KEEP does not imply CATLG. As a result, DISP=(NEW,KEEP) should be rarely used because next time you retrieve the dataset, you need to specify UNIT and VOL=SER.

CATLG: This indicates that the dataset is to be kept and an entry for it placed in the catalog when the step terminates.

PASS: This indicates that an entry for the dataset (containing DSN, VOL=SER and UNIT information) be placed on a table in storage (Passed Dataset Queue). This entry is to be used in a subsequent step to "receive the passed dataset". A message is displayed "PASSED".

Instructor Notes:**The Abnormal (or Conditional) Disposition Field:**

This field is used to notify the system how to dispose off the dataset when the step terminates abnormally (ABENDs). It is required only if this disposition is different from the normal disposition.

DELETE, KEEP, CATLG, and UNCATLG have the same meaning as they have in the normal disposition. Note that **PASS** is not permitted in the abnormal disposition field.

The best example of using the abnormal disposition field is,

DISP=(NEW,CATLG,DELETE)

If there is ABEND, the dataset is to be deleted. This eliminates future manual intervention to delete and uncatalog the dataset in order to restart.

Defaults: Some defaults in the DISP parameter are fixed and others are variable.

- If the DISP parameter is omitted, the default is always **(NEW,DELETE)**.

//SYSUT1 DD UNIT=SYSDA,SPACE=(TRK,(1,2))
/* (NEW,DELETE) IS THE DEFAULT

- If the status is omitted, the default is always **NEW**.
DISP=(,CATLG) is same as DISP=(NEW,CATLG)
- If the normal disposition field is omitted:
 - If the status field is **NEW**, the default is **DELETE**.
 - If the status field is **OLD** or **SHR** and the dataset name is non-temporary:
 - If the DD statement is not receiving a passed dataset, the default is **KEEP**.

//DD1 DD DSN=DA0001T.EMPFILE,DISP=SHR

- If the DD statement is receiving a passed dataset, which was created during the execution of the job and was never given a permanent disposition, the default is **DELETE**.

```
//S1      EXEC      PGM=PR
//DD1      DD        DISP=(, PASS), DSN=USER1.PDST,
//                  UNIT=SYSDA,
//                  DCB=(BLKSIZE=23440, LRECL=80, RECFM=FB),
//                  SPACE=(TRK, (50,10),RLSE)
//S2      EXEC      PGM=PC
//DD2      DD        DISP=OLD, DSN=USER1.PD
```

In DD2, **DISP=OLD** defaults to **DISP=(OLD, DELETE)**.

Instructor Notes:**The Abnormal (or Conditional) Disposition Field (Contd.):**

- If the normal disposition field is omitted (Contd.):
 - If the status field is **OLD** or **SHR** and the dataset name is non-temporary (Contd.):
 - If the DD statement is receiving a passed dataset, which was created during the execution of the job but was given permanent disposition since being created, the default is **KEEP**.

```
//S1      EXEC    PGM=PR
//DD1     DD       DISP=(, PASS),
             DSN=USER1.PDST,UNIT=SYSDA,
             // DCB=(BLKSIZE=23440, LRECL=80, RECFM=FB),
             // SPACE=(TRK, (50,10), RLSE)
//S2      EXEC    PGM=PC, COND=(4, LT)
//DD2     DD       DISP=(OLD, CATLG), DSN=USER1.PDST
//S3      EXEC    PGM=PF, COND=(4,LT)
//DD3     DD       DISP=(OLD,PASS), DSN=USER1.PDST
//S4      EXEC    PGM=PK, COND=(4, LT)
//DD4     DD       DISP=OLD, DSN=USER1.PDST
```

In **DD4**, **DISP=OLD** defaults to **DISP=(OLD,KEEP)**

- If the DD statement is receiving a passed dataset, which existed before the job began execution, the default is **KEEP**.

```
//S1      EXEC    PGM=PR
//DD1     DD       DISP=(SHR,PASS), DSN=USER1.LONE
//S2      EXEC    PGM=PK, COND=(4,LT)
//DD2     DD       DISP=SHR,DSN=USER1.LONE
```

In **DD2**, **DISP=SHR** defaults to **DISP=(SHR,KEEP)**

Despite the several possible defaults for **DISP=OLD** or **DISP=SHR** their use is extremely common. When not receiving a passed data set, they always safely default to **DISP=(OLD,KEEP)** and **DISP=(SHR,KEEP)**, respectively.

- If the status field is **OLD** or **SHR** and the dataset name is temporary, the default is **pass**.

```
//DD1     DD DISP=OLD,DSN=&&TEMP
```

DISP=OLD defaults to **DISP=(OLD,PASS)** and the message is displayed as the output – “INVALID DISP FIELD – PASS SUBSTITUTED”

- If the status field is **MOD**, which defaults to an existing data set, **MOD** works the same as **OLD** and **SHR**. Page 05-21
- If the status field is **MOD**, which defaults to **NEW**, the default of the second field is **DELETE**.

Instructor Notes:**The Abnormal (or Conditional) Disposition Field (Contd.):**

DISP=MOD can default to **(MOD,KEEP)**, **(MOD,DELETE)**, **(MOD,PASS)**, and **(NEW,DELETE)**. In view of all these possibilities, it is recommended that defaults are not to be practiced with **MOD**.

Remark: The various fields of the DISP parameter stand for the PDS and not the member.

```
//DD1 DD DSN=USER1 LIB2(Z32), DISP=(OLD,DELETE,DELETE)
```

In the above case, the PDS and the member both are deleted. Both the PDS and the member should exist.

How a member is handled depends on whether or not it exists and whether the program opens for input or output. A summary of all possibilities is presented below.

//S1	EXEC	PGM=P1
//D1	DD	DSN=DA0001T LIB(M12),DISP=SHR

M12 EXISTS

- In P1, if M12 is opened in I/P mode, for reading, M12 is read.
- In P1, if M12 is opened in O/P mode for writing, M12 is replaced (not in place).

M12 DOES NOT EXIST

- In P1, if M12 is opened in I/P mode for reading, ABEND (S013-18).
- In P1, if M12 is opened in O/P mode for writing; M12 will be created and written into.

Instructor Notes:**5.3: The DISP Parameter
DD STATEMENT****PDS and DISP:**

- To create a new member of an existing PDS, code DISP=OLD or SHR, not NEW, since DISP refers to the partitioned dataset as a whole not to the member.
- The PDS directory is updated to know about the new member.

If DISP parameters are not coded, the following defaults:

DISP Coded as :Defaults to :

If not coded(NEW,DELETE,DELETE)
(OLD).....(OLD,KEEP,KEEP)
(SHR).....(SHR,KEEP,KEEP)
(,CATLG).....(NEW,CATLG,CATLG)
(MOD).....If dataset does not exist :

(NEW,DELETE,DELETE)

If dataset exists

:

(OLD,KEEP,KEEP)
(MOD,CATLG,).....(NEW/OLD,CATLG,)
 DELETE) DELETE)

Disposition of OLD deletes existing records from the dataset. So, to append records, code MOD.

Instructor Notes:

5.3: The DISP Parameter
Normal and Abnormal DISP-Parameters (Contd.)

If M12 EXISTS

```
//S1 EXEC PGM=P1
//D1 DD DSN=DA0001T.LIB(M12),
//                               DISP=SHR
```

- P1 opened in I/P mode for reading, M12 is read.
- P1 opened in O/P mode for writing, M12 will be replaced (not in place)

If M12 DOES NOT EXIST

- P1 opened in I/P mode for reading, ABEND (S013-18).
- P1 opened in O/P mode for writing, M12 will be created and written into.

Instructor Notes:

[5.4: The UNIT and VOLUME Parameters](#)
Characteristics

UNIT and VOLUME parameters work together to specify the location of the dataset.

UNIT indicates the following:

- The device type or device address on which the dataset resides on a volume
- The number of devices to be allocated to the dataset
- The instance when the mount message is to be shown to the operator

There is no default for UNIT.

Instructor Notes:**5.4: The UNIT and VOLUME Parameters**
Format and Example**Format**

```
device address  
UNIT=( generic device name ,device-count [, DEFER]  
)  
generated device name
```

Example

```
UNIT=3380 ; UNIT=3400-6 ; UNIT=(3390,2)  
UNIT=SYSDA ; UNIT=DISK ; UNIT=TAPE
```

Format:

device address: Identifies the exact device address. This notation is almost never used.

generic device name: Identifies the device type using a universal system-supplied name.

For example: UNIT=3390; UNIT=3400-5; UNIT=3480

generated device name: Identifies the device type using an installation-defined name.

For example: UNIT=SYSDA; UNIT=DISK; UNIT=TAPE

The generated names can be made to mean whatever an installation requires them to mean. For example, UNIT=SYSDA can mean all 3380 devices of any density, or single density only, or a subset of double density devices or a combination of 3380 and 3390 device. Their definition can vary from installation to installation.

device count: Specifies the number of devices to be allocated for the dataset. The limit is 59 devices. If omitted, default is 1 except when DD statement describes a disk multi-volume dataset. In such case, device count=number of volumes.

Of the four, the generated device name is used most commonly.

Instructor Notes:

5.4: The UNIT and VOLUME Parameters
Use of the UNIT Parameter

The UNIT parameter can be coded in any one of the following three ways :

- By specifying the device address

• Example :

UNIT=301

- By specifying the generic name that identifies a particular type of device

• Example :

UNIT=3380

- By specifying a Group name that specifies devices that belong to categories set up by the installation

• Example:

UNIT=TAPE

Use of the UNIT Parameter:

(i) For example, UNIT=(SYSDA,5) , UNIT=(TAPE,2)

(ii) For example, 2 UNIT=SYSDA is same as UNIT=(SYSDA,1) because of default.

(iii) For example, 3

```
//DD1 DD DSN=DA0001T.EMPFILE,DISP=(CATLG,DELETE),
//      UNIT=SYSDA,VOL=SER=(BS3001,BS3002,BS3003),
//      SPACE=(TRK,(1,2)),DCB=(LRECL=80,RECFM=FB,
//      BLKSIZE=800)
```

In this example, UNIT =SYSDA defaults to UNIT=(SYSDA,3)

Note: UNIT=(,2) can also be used if the device is being supplied by the catalog.

Default:

There is no default for device name. If it is not coded in the UNIT parameter and it is also not supplied by the catalog, by the Passed dataset Queue, or by the VOL=REF, the result is a JCL error. The message is “IEF2101 JOBNAME STEPNAME DDANAME – UNIT FIELD SPECIFIES INCORRECT DEVICE NAME”, which is misleading. It means that the device name was needed but not coded.

Instructor Notes:**5.5: The VOL Parameter**
Use of the VOLUME (VOL) Parameter

VOLUME indicates Vol-Ser of the dataset's volume.

UNIT and VOLUME parameters are not needed for cataloged datasets.

Format:

```
VOL = { SER=(VOL1[,VOL2], ...) }  
      REF=referback  
      REF=dsname
```

Remark :

- The maximum no. of volumes is 255 and there is no default for VOL=SER or VOL=REF

Use of the VOLUME (VOL) Parameter:

The main function of the VOL (or VOLUME) is to identify the volume(s) by serial number where an existing dataset resides or where a new dataset will reside.

SER=(vol1,vol2,...) – Specifies the serial number(s) of the volume(s) to be used. The maximum number of volumes is 255.

A volume serial is a combination of alphabetic, numeric, and national characters (\$ @ #) up to 6. A hyphen is also permitted. In a real (or production) environment, the number of characters is almost never less than 6.
For example.

```
VOL=SER=BS3001 or  
VOLUME=SER=BS3001  
VOL=SER=(BS3013,BS3014)
```

The VOL parameter must be coded:

- While retrieving a dataset, which is neither cataloged nor passed.
- While retrieving a dataset, which is cataloged, but the catalog must not be used.
- While creating a dataset, which must reside on a particular volume.

Instructor Notes:**Use of the VOLUME (VOL) Parameter:**

- **REF=referback**

Referback can have three formats:

- ***.stepname.ddname**: This requests that the volume is to be the same as the volume for DD statement **ddname** found in the previous step **stepname**

VOL=REF=*.STEP2.DD1

- ***.ddname**: This requests that the volume is to be the same as the volume for previous DD statement **ddname** found in the same step **stepname**.

VOL=REF=*.DD1

- ***.procexec.stepname.ddname**: This requests that the volume is to be the same as the volume for DD statement **ddname** found in the previous step **stepname** found within a procedure **procexec** (name of the **EXEC** statement invoking the procedure

VOL=REF=*.PR1.STEP2.DD1

Referbacks are not encouraged. They are to be used only when they are necessary. A **referback** with a **stepname** causes a JCL error if the referenced step does not execute. Such **referbacks** must be avoided where restart is required.

- **REF=dsname**: This requests that the volume is to be the same as the one where dataset **dsname** resides. The dataset must be cataloged or passed. The dataset does not even have to exist, as long as it is cataloged or passed. The name of the referenced dataset need not appear anywhere else in the job.

For example, VOL=REF=DA0001T.EMPFILE

Remark: When **VOL=REF** (**referback** or **dsname**) is used, the system supplies the volume as well as the unit information. Therefore, the **UNIT** parameter is usually unnecessary.

Default:

There is no default for **VOL=SER** or **VOL=REF**. However, if both are omitted, no JCL error results. Instead, the meaning of DD statement changes. For example, while retrieving when **VOL=SER** or **VOL=REF** is coded, the catalog is not used. If neither is coded, the catalog is used.

Instructor Notes:**VOLUME=(v1,v2,v3,v4,v5)**

To specify the volume where a dataset is stored

V1=PRIVATE, requests for private volume for the dataset to reside
V2=RETAIN, the volume must not be dismounted after dataset is closed
at the end of the job step

V3=VOLUMN SEQ NO, In multi volume dataset it specifies the volume
to begin processing with. Volume sequence number values ranges
from 1 thru 255

V4=VOLUMN COUNT, refers to the maximum number of volumes
required for output data sets. This is used with tape data sets and
maximum volumes allowed for tape data sets is 8

V5=SER indicates the serial numbers of the tape or disk volume the
dataset is on. It takes maximum of 6 characters.

UNIT and VOLUME parameters not needed for cataloged dataset.

VOL=SER=(VOL1[,VOL2],]

While creating New datasets, if UNIT is not coded:

Defaults to installation-defined devices based on purpose:

Unit	Volume		
For Development datasets		3390	ZTSO01
For Testing datasets		3391	ZRSO06

For SMS managed datasets, defaults to STORAGE CLASS:

Storage Class	Unit	Volume
SCTSO	3390	ZTSO03

Common problems with UNIT

If you forget to code DISP at the dataset that you are reading,

//DDRd DD DSN=IGTRN12.OLD.DS

you will get an error message SPACE NOT SPECIFIED FOR
DATASET or INCORRECT DEVICE TYPE SPECIFIED.

**Correct this problem by coding DISP=SHR that you forgot, not
UNIT.**

You will get this same error message if you code an incorrect UNIT with
a Cataloged dataset.

Instructor Notes:**5.6: The SPACE Parameter**
Characteristics of SPACE

The SPACE parameter is coded for Non-VSAM datasets, to specify how much space is to be allocated to the dataset.

It has two positional sub-parameters:

- The first sub-parameter indicates the unit of measure used for space allocation.
- Second sub-parameter of the SPACE parameter indicates how much space to allocate to the dataset.

A diagram showing the syntax of the SPACE parameter. It is enclosed in a rounded rectangle. On the left, 'SPACE=' is followed by a left brace [and three options: 'TRK,' (with a comma), 'CYL,' (with a comma), and 'BLK,' (with a comma). To the right of the options is another left brace [and a closing right brace]. Inside the inner braces, there is a list: '(prim-alloc[,sec-alloc][,directory])'.

Characteristics of the SPACE Parameter:

The SPACE parameter must be included in a DD statement at the following instances:

A new disk dataset is created.

An old dataset needs to alter its entitlement to additional space. That is, it needs to request additional disk space for an old dataset when available space is exhausted.

An old disk dataset must make all unused space available.

TRK: This requests that space be allocated in tracks.

CYL: This requests that space be allocated in cylinders.

Blksize: This specifies the average blocksize of the dataset. The system translates it to tracks.

Instructor Notes:**5.6: The SPACE Parameter**
Characteristics of SPACE (contd.)

The second sub-parameter has three positional parameters.

The three positional sub-parameters specify:

- Primary
- Secondary
- Directory space

Directory allocation is made only for PDS.

Directory blocks(256 bytes each) are the units of measure for directory.

Characteristics of the SPACE Parameter:

Directory: This specifies the number of directory blocks (256 bytes each) to be assigned to the directory of a PDS.

The directory quantity, if not coded, defaults to zero; therefore, the directory quantity must be specified for a new PDS. If it is so, S013-14 ABEND failure does not occur if an attempt is made to add the first member to a PDS.

This identifies the number of tracks (if TRK is coded) or cylinders (if CYL is coded) or the number of blocks (if blksize is coded) that must be allocated during the allocation process for a new dataset before the step begins execution. The system allocates the requested space in one extent. If this is not possible (and CONTIG is not coded), two extents are used, then three and so on up to five extents. If as many as five extents cannot satisfy the request, the result is the following allocation JCL error:

IEF257I jobname stepname ddname -SPACE REQUESTED NOT AVAILABLE.

If the request is nonspecific (no VOL=SER or VOL=REF), needing a storage volume, the JCL error message will be different:

IEF257I jobname stepname ddname -INSUFFICIENT SPACE ON STORAGE VOLUMES.

Remark: The system always allocates the primary quantity in the least number of extents possible on a single volume. The primary quantity cannot be split over multiple volumes. The primary allocation cannot be omitted (coding 0 is allowed). It is ignored if the dataset is old.

For example, SPACE=(TRK,3)

For example, SPACE=(CYL,4)

For example, SPACE=(23440,100)

For example, SPACE=(TRK,0)

The primary allocation cannot be omitted (coding 0 is allowed). It is ignored if the dataset is old.

Instructor Notes:**Characteristics of the SPACE Parameter (Contd.):**

- **sec-alloc - Secondary allocation or secondary quantity:**

This identifies the number of tracks (if **TRK** is coded) or cylinders (if **CYL** is coded) or the number of blocks (if **blksize** is coded) that are to be allocated when all the available space is exhausted while writing to a dataset. The system allocates the secondary quantity in the least number of extents possible, and just like the primary quantity; it can be given in as many as five extents, if necessary.

The system always supplies the specified secondary allocation when one is needed unless one of the two events occurs:

- The allocated volume does not have enough space to satisfy the secondary allocation and no other volumes are allocated.
- The needed secondary allocation, if granted, causes the dataset to exceed 16 extents on the volumes and no other volumes are allocated.

If either of these two conditions arises, the result is a **SB37-04** ABEND failure (normally for a sequential dataset). For a PDS, the ABEND can also be **SE37-04**.

Please note that a PDS is confined to a single volume, while a sequential dataset can extend into a maximum of 59 volumes. The 16-extent-per-volume limit for a dataset is system-supplied and cannot be altered.

The secondary allocation is optional. If omitted, defaults to 0. When no secondary allocation is coded and the primary allocation is exhausted, the result is an SD37-04 ABEND failure.

Instructor Notes:**Characteristics of the SPACE Parameter (Contd.):**

- **sec-alloc - Secondary allocation or secondary quantity (contd.):**
Remark: The directory quantity is taken away from the beginning of the primary allocation if **TRK** or **CYL** is coded in the **SPACE** parameter. When **blksize** is coded, the system adds the directory blocks to the data blocks and then computes the amount of primary space.

For example, **SPACE=(TRK,(20,5,5)) OR SPACE=(TRK,(20,,5))**

If no secondary

For example, **SPACE =(CYL,(20,5,5)) OR SPACE =(CYL,(20,,5))**
if no secondary

For example,
SPACE =(23440,(200,50,5)) OR SPACE =(23440,(200,,5))
if no secondary

- **RLSE:** This requests that any unused space is to be freed when the dataset is closed. This works for both new and old datasets, provided they were opened for output. Space is released on the boundary used in the **SPACE** parameter. If tracks (or cylinders) are allocated, unused tracks (or cylinders), are released.

Remark: Using **RLSE** is highly recommended for datasets not intended for future expansions. Temporary datasets are ideal candidates. For datasets that expand in future runs, **RLSE** can result in a larger number of extents, and, possibly, a premature SB37-04 ABEND failure. **RLSE** will be ignored if the dataset is opened by another user (or shared by another job) or the step ABEND's.

For example, **SPACE=(TRK,(5,1),RLSE)**

Instructor Notes:

5.6: The SPACE Parameter Example

For PS

SPACE=(CYL,(10,1))

SPACE=(800,(500,100))

For PDS

SPACE=(23440,(200,50,2))

SPACE PARAMETER

SPACEPARAMETER Specifies how much space to allocate for a new disk (DASD) dataset.

Syntax:

SPACE=(S1,(S2,S3,S4),S5)

S1 - the Unit of space in Cylinders (CYL), Tracks (TRK), Blocks of records (a number representing block size), Record length

S2 – Primary (initial) quantity of units

S3 - Secondary quantity of units if primary is exceeded; but allocated only when the dataset expands.

(S3 X 15 or 16 or 123 secondary allocations - called Extents)

S4 - Number of Directory blocks (applicable for PDS only)

S5 - Releases the unused space requested

EXAMPLE:

SPACE=(TRK,(5,2)) You get 5 tracks + (2 X 15) tracks = 35 tracks

Disk surfaces are divided into Tracks of recording space

Group of tracks make a Cylinder

Many cylinders make a disk Volume

Different models of IBM Mainframe disks have different track capacities, number of cylinders and device capacities.

Disk Volume Model	Bytes/Track	Tracks/Cylinder	Total Cylinders	Total Bytes
3390-1	56664	15	1113	0.946 Gb
3390-2	56664	15	2226	1.892 Gb

Instructor Notes:**Space (EXTENTS)**

The smallest unit of space on mainframe disks is one whole track. The OS cannot split tracks for use by other datasets

One or more contiguous tracks or cylinders allocated to a dataset are called an EXTENT

An EXTENT is a piece of disk space.

Your primary space allocation is 1 extent.

When you request SPACE=(TRK,(5,2)) you get:

1 primary extent and 2*15 secondary extents
tracks

Space (Block size)

You can specify space request by Block size instead of Tracks or Cylinders.

EXAMPLE:

SPACE=(3840,200) You get 200 blocks each holding
3840 bytes = 768,000 bytes on the disk

If you code a secondary allocation:

SPACE=(3840,(200,60)) ...plus (60 X 3840 bytes X 15
Extents) as secondary allocation

Since one track is the minimum unit of space, if block size requested works out smaller than one track, you still get one track of disk space.

SPACE (Record Length)

Applicable only for SMS managed datasets

Code AVGREC parameter along with Space parameter

Do not code BLKSIZE in the DCB keyword parameter\

```
//DDN      DD      DSN=DSRC012.DS,
//                      DISP=(NEW,CATLG,DELETE),
//                      UNIT=SYSDA,
//                      AVGREC=U,
//                      SPACE=(133,10000)
//                      RECFM=FB,
//                      LRECL=133
```

With AVGREC=U:

The 133 in Space parameter is the average Record length in bytes
=U means that the 10000 is a "unit" estimate of the quantity of records to be written.

Instructor Notes:**Example**

```
//DDN      DD      DSN=DSRC012.DS,
//                                DISP=(NEW,CATLG,DELETE),
//                                UNIT=SYSDA,
//                                AVGREC=U,
//                                SPACE=(133,10)
```

You get (10 records X 133 bytes) of space.

For Secondary allocation request: SPACE=(133,(10,2))

AVGREC can support record quantities by thousands (AVGREC=K) and by millions (AVGREC=M)

```
//          AVGREC=M,
//          SPACE=(133,10)
```

You get (10,000,000 records X 133 bytes) of space.

SPACE (Directory Blocks)

While creating a PDS, the SPACE parameter is to include a request for Directory Blocks

Space=(TRK,(1,2,7))

You get $1 + (2 \times 15) = 31$ tracks and create 7 Directory blocks. Each directory block can store information about 5 members; implies 35 members can be allocated.

Space=(TRK,(1,,7))

When you do not want secondary space for the PDS.

EXAMPLES:

Space=(TRK,(5,,1)) Allocates a PDS with no secondary space and 1 directory block.

Space=(3840,(200,60)) Allocates $(200 \times 3840) + (60 \times 3840 \times 15)$ of disk space.

```
//          //DDN DD DSN=.....,
//          DISP=.....,
//          UNIT=3390,
//          Space=(TRK,10)
```

Instructor Notes:**5.6: The SPACE Parameter
Characteristics**

When the directory is exhausted, the result is an S013-14 ABEND failure.

The directory quantity, if not coded defaults to zero, that is, a sequential dataset.

When the secondary space is exhausted, the result is SB37-04 ABEND failure for a sequential dataset and SE37-04 ABEND failure for a PDS.

There is no default for primary allocation (coding 0 is allowed) and secondary and directory space if omitted, defaults to zero.

Instructor Notes:

You receive the secondary space requested for your dataset only as the dataset expands as you use it. You can get up to 16 extents of secondary space per dataset per disk volume for a PS. If you use more than one disk volume for a PS dataset, it can spread up to 16 secondary extents on each volume. A PDS cannot span disk volumes. The limit of 16 extents per dataset per disk volume does not apply to VSAM datasets, which can exist with up to 123 extents.

SPACE (RLSE)

Causes all of the whole unused tracks to be freed for use by other datasets.

Syntax:

SPACE=(6200,300,RLSE)	Releases unused tracks.
SPACE=(6200,(300,60),RLSE)	Releases unused tracks in the last active secondary.
SPACE=(80,1,RLSE)	No space is released since space is less than a track for any disk volume model.

RLSE does not work if a step abends

Does not work if the dataset is not closed normally

Does not work when the dataset is shared

Instructor Notes:**5.7: The LABEL Parameter Characteristics**

The LABEL parameter can specify the following:

- The sequence of a tape dataset on a volume
- The type of label of the dataset

Format :

LABEL=([seq-no][, type])

- seq-no: This identifies the sequence number of the dataset on a tape volume. It can have 1 to 4 digits. If omitted, it defaults to 1. If 0 is coded, it defaults to 1. maximum: 9999
- seq-no (Contd.): The sequence number is ignored in the following situations:
 - Retrieving a tape dataset through the catalog
 - Receiving a passed tape dataset

Instructor Notes:**5.7: The LABEL Parameter
Characteristics (contd.)**

Type: This identifies the type of label for the dataset. There are many types of labels :

- SL indicates IBM standard label. If the sub-parameter is omitted, SL is the default. NL indicates that no labels are used. NL is not commonly used. Normally NL is used for a tape coming from or going to another installation which has no SL capabilities.

Characteristics of LABEL:

There are many types of labels. To name a few, which are important from project perspective.

SL: This indicates IBM standard label. If the sub parameter is omitted, SL is the default.

NL: This indicates no labels are used. NL is not commonly used. Normally, NL is used for a tape coming from or going to another installation, which has no SL capabilities.

BLP or Bypass Label Processing: This indicates that labels are not to be recognized and are to be treated as ordinary files. BLP is used as a last resort when neither SL nor NL can accomplish what is required.

Label Verification: While retrieving an SL tape dataset, both the volume serial and the dataset name are verified. While creating an SL tape dataset with VOL=SER or VOL=REF, only the volume serial is verified.

While retrieving an NL tape dataset, neither the volume serial nor dataset name can be verified. However, only an NL tape volume can be mounted. An SL volume is rejected.

Instructor Notes:**5.7: The LABEL Parameter
Examples**`LABEL=(2, SL)``LABEL=(, NL)`

Defaults : If omitted, the LABEL parameter defaults to (1, SL).

There are four ways to supply the same information:

- Omit the label parameter
- Code LABEL=(1, SL)
- Code LABEL=(, SL) 1 is the default
- Code LABEL=1 SL is the default

Instructor Notes:**5.7: The LABEL Parameter
Examples (contd.)**

Creating labeled tape dataset:

```
//OUT DD DSN=DA0001T.TAPE,  
// DISP=(, CATLG), UNIT=TAPE,  
// DCB=(BLKSIZE=32720, LRECL=80,  
// RECFM=FB)  
/* LABEL not supplied - the default is (1, SL)
```

Creating non-labeled tape dataset:

```
//OUT DD DSN=DA0001T.TAPE,  
// DISP=(, KEEP), UNIT=TAPE, LABEL=(, NL),  
// DCB=(BLKSIZE=32720, LRECL=80,  
// RECFM=FB)  
/* LABELs are not used
```

Instructor Notes:**5.8: The DCB Parameter Characteristics**

The DCB parameter specifies characteristics that are stored in the Data Control Block of the file.

No DCB parameter is required, if the application program supplies the required information.

Format:

DCB=([referback] | [model] [, subparm] ,)

Characteristics of DCB:

The DCB parameter specifies values to be used to complete the Data Control Block (DCB) when a dataset is opened. A DCB is constructed by the language processor (compiler or assembler), based on the appropriate instructions of the language being used, and resides inside the code of the program. The compiler collects this information and defaults from various parts of the program (For example, in COBOL, RECORD CONTAINS 80 CHARACTERS; BLOCK CONTAINS 10 RECORDS and so on) and constructs the DCB. Note that the DCB exists only for non-VSAM datasets and is checked by the OPEN routines (for input or output). Certain values must be "hard-coded" in the DCB by the program. Others can be left out, giving the user the option of supplying these values via the DCB parameter (as well as other means).

There are three suppliers of DCB information:

Values supplied by the program, referred to as hard-coded. When a value is hard-coded, it cannot be changed unless the program is changed.

Values coded in the DCB parameter of the DD statement. These values are ignored if they are already hard-coded.

Values from the standard label of the dataset. The values supplied by the label are limited to: BLKSIZE, LRECL, RECFM, DSORG, and so on. Values from the label are not used if they are hard-coded inside the program or coded in the DCB parameter of the DD statement.

Instructor Notes:**5.8: The DCB Parameter Characteristics**

Format for referback:

- *.*stepname.ddname*
- *.*ddname*
- *.*procexec.stepname.ddname*

Model specifies the name of a dataset which must be cataloged and resides on disk.

- This dataset is called a model DSCB.
- Example :

Sub Parameters: BLKSIZE, LRECL, RECFM, DEN, BUFNO and DSORG

DCB=PROD.MODEL

Characteristics of DCB:

DCB=([referback] | [model][,subparameter],..... keyword parameter

Referback: This can have three formats:

*.*stepname.ddname*: This requests that the DCB parameter is to be copied from the DD statement "ddname" found in the previous step "stepname".

DCB=*.STEP2.DD1

*.*ddname*: This requests that the DCB parameter is to be copied from a previous DD statement "ddname" found in the same step "stepname".

DCB=*.DD1

*.*procexec.stepname.ddname*: This requests that the DCB parameter is to be copied from DD statement "ddname" found in the previous step "stepname" found within a procedure "procexec" (name of EXEC statement invoking the procedure).

DCB=*.PR1.STEP2.DD1

Remark: The DCB referback copies the DCB parameter as opposed to the DSN and VOL=REF referbacks which acquire the dataset name and the VOL=SER respectively, whether or not the DSN and VOL parameters are present in the referenced DD statement. If the DCB referback refers to a DD statement, which contains no DCB, nothing is copied and no message appears.

Sub-parameters: Mnemonic Hint:BLaBbeReDD (B: BLKSIZE, L: LRECL, B: BUFNO, R:RECFM, D:DEN, and D:DSORG).

Instructor Notes:**Models:**

This specifies the name of the dataset which has following characteristics:

- It must be cataloged. If it is not, the result is a JCL error:
IEF2121 jobname stepname ddname -DATASET NOT FOUND
- It must be on disk (Tapes not allowed).
- It must reside on a volume that is accessible (online).

This dataset is called a model DSCB. The DCB information from the label of the model is extracted and can be used.

For example, DCB=DA0001T.EMPFILE

For example, In case you want to override some of the subparameters, the overriding subparameters must follow the DCSB model dataset name.
DCB=(DA0001T.EMPFILE,LRECL=100,BLKSIZE=800)

Models are generally used, during the creations of GDGs and dummying the PDS.

Subparameters: There is a vast number of subparameters, the great majority of which are seldom or never used.

- **BLKSIZE:** This specifies the size of the block (also known as the physical record). For **RECFM=FB**, the blocksize must be multiple of the logical record length, and it identifies the exact size of the block. For **RECFM=VB**, the blocksize can be any value up to the limit but atleast 4 bytes larger than the logical record length. For **RECFM=U**, the blocksize can be any value up to the limit.

- **Remark:** There is no default for **BLKSIZE**. Coding **BLKSIZE=0**, the system computes the optimum blocksize based on the device type.

For example,
e, DCB=BLKSIZE=800

- **LRECL:** This specifies the size of the logical record. The maximum size is 32,760, and it cannot be larger than blocksize, unless **RECFM=VBS** is used.

For example, DCB=(LRECL=80,BLKSIZE=800)

Instructor Notes:**5.9: Models**
Models: Subparameters

DSORG = XX Specifies the datasets organization

- PS = Physical Sequential
- PO = Partitioned
- DA = Direct
- IS = Indexed Sequential

RECFM = XX Format of the file's records

- F = Fixed length, unblocked
- FB = Fixed length, blocked
- V = Variable length, unblocked
- VB = Variable length, blocked
- LRECL = n length of file's records
- BLKSIZE = n length of file's block

Models: Subparameters (Contd.):

RECFM: Specifies the record format. There are several values (or combinations of values) that can be coded:

F: This indicates that all blocks and all logical records are fixed in size.

V: This indicates that blocks as well as logical records are of variable size. The first four bytes of each block (and logical record) describe its length.

B: This indicates that one or more logical records reside in each block.

B cannot be coded alone. It is used in conjunction with F or V. For example FB or VB.

U: This indicates that blocks are of variable size. There are no logical records. Mainly used with Load Library.

S: For fixed-size records, this indicates that no short blocks are permitted anywhere but the end of the data. For variable-size records, it indicates that a logical record can span more than one block. S cannot be coded alone. It must follow F, V, FB or VB.

A: This indicates that the first character of each record is an ANSI control character to be used for printer carriage control. A cannot be coded alone. It must follow F, V, FB, VB or U.

For example, DCB=(LRECL=80,RECFM=FB,BLKSIZE=800)

Instructor Notes:**Models: Subparameters (Contd.):**

If **RECFM** is not supplied through any means, **U** is the default.

- **DEN:** This identifies the density of the tape. **DEN=3(or 4)** indicates 1600 (or 6250) BPI density.
- **BUFNO:** This identifies the number of buffers to be allocated in virtual storage by the **OPEN** routines, which will contain the blocks to be read in or written out. If omitted, default is 5. The maximum is 255. Coding for **BUFNO** a number greater than 5 may require that the **REGION** parameter be increased. However, default of 5 is more than adequate in most cases of dataset processing.

For example,

```
//INFILE DD  
DSN=DA00011.EMPFILE,DISP=SHR,DCB=BUFNO=8
```

- **DSORG:** This identifies the organization of the datasets.
 - **PS:** This specifies physical sequential organization. Mostly **QSAM** and sometimes **BSAM**.
 - **PO:** This specifies partitioned organization (or **BPAM**).
 - **DA:** This specifies direct organization (or **BDAM**).
 - **IS:** This specifies indexed sequential organization (or **ISAM**).

It is important to understand which of these often-used parameters are normally hard-coded and which are not:

- **BLKSIZE:** This is seldom hard-coded. The **BLKSIZE** is unrelated to the logic of the program and hard-coding its value would cause unnecessary changes whenever the **BLKSIZE** is changed. In COBOL, **BLOCK CONTAINS 0 RECORDS** must be coded to avoid hard-coding the **BLKSIZE**. Omitting this clause causes a default of 1 to be used. The result is a hard-coded **BLKSIZE** equal to **LRECL**. Many installation standards disallow hard-coding the **BLKSIZE** for sequential and partitioned datasets.
- **LRECL:** This is frequently hard-coded. The logic of any ordinary program is dependent on the **LRECL** and, as a result, the **LRECL** cannot be changed without changing the logic of the program. Many high-level languages such as COBOL always hard-code the **LRECL**.
- **RECFM:** This is frequently hard-coded. The logic of any ordinary program is dependent on the **RECFM** and, as a result, the **RECFM** cannot be changed without changing the logic of the program. Many high-level languages like COBOL always hard-code the **RECFM**.

Instructor Notes:

5.9: Models
DD STATEMENT

LRECL, indicates the logical record length of dataset in bytes

- For fixed length records, the dataset contains records all of the same length and hence LRECL = the actual length of the data-bytes.
 - LRECL for fixed length records can range from 1 to 32,760 bytes. You can access all 32,760 bytes.
- For variable length records, the length of the records vary from an average to a maximum. So, the LRECL = longest data-bytes length + 4 bytes; the 4 bytes called the RDW(record descriptor word) is pre-pended to each record and indicates the actual length of the record in its first two bytes
 - LRECL can be as high as 32,760 but only 32,752 are accessible.
- For Undefined format, code LRECL=0. Conventionally used to store load modules.

Instructor Notes:**5.9: Models**
DD STATEMENT

BLKSIZE, specifies the maximum length of a block. The value must be the multiple of LRECL and 4 bytes greater than longest record for variable length record.

- For fixed length blocked records (RECFM=FB)
 - Block size is multiples of LRECL.
 - Must not be larger than 32,760 bytes.
 - Example: DCB=(RECFM=FB,LRECL=80,BLKSIZE=8000)
- For variable length blocked records (RECFM=VB)
 - Block size is multiples of LRECL plus 4 bytes, for Block descriptor word (BDW) which is pre-pended to the block of records. The first two bytes of the BDW indicates the length of the block.
 - Example: DCB=(RECFM=VB,LRECL=84,BLKSIZE=8404).
- For Undefined format (RECFM=U) records, let the OS determine the block size.
- DCB=(RECFM=U,LRECL=0)

Instructor Notes:

5.9: Models
Models: Example

Example

```
DCB = (DSORG = PS, RECFM = FB, LRECL = 80,  
       BLKSIZE = 800)
```

Instructor Notes:**5.9: Models**
Models: Remarks

A DCB exists for every non-VSAM dataset to be opened by the program (for input or output). Certain values must be hard-coded by the program itself. Others can be left out, giving the user the option of supplying these values via the DCB parameter as well as by other means.

ABEND failures while supplying inconsistent DCB values:

- S013-20 ABEND when RECFM=FB is used but the LRECL is not an exact multiple of the block size.
- S013-34 ABEND when RECFM=FB is used and the LRECL is greater than the BLKSIZE.
- S013-34 ABEND when RECFM=VB is used and the LRECL is greater than the BLKSIZE-4.
- S001-04 ABEND when BLKSIZE in the DCB parameter is smaller than the actual blocksize and a multiple of the LRECL in the DSCB of the dataset.

Instructor Notes:**5.10: In-Stream Data Format**
Characteristics of In-Stream Data

* and DATA are positional parameters that follow DD.

Example

```
*  
//ddname      DD    DATA,  
               DLM = XX
```

```
//INPUT      DD *  
A0014214 CHAR  
A0024342 DABL  
/*
```

Characteristics of In-Stream Data:

The input stream submitted to the system for execution consists of two possible parts:

JCL: The mandatory part of the input stream

Data mixed in with JCL in the input stream: This data is known as SYSIN data or input stream data. It is an optional part of the input stream and always has a logical record length of 80. Any records encountered in the input stream which are not JCL statements are treated as the SYSIN data.

SYSIN data must be preceded by a DD statement such as follows:

SYSIN data encountered by JES2 or JES3 following a DD * statement saved on the SPOOL volume for future use. This is known as input spooling. The SYSIN is delimited (the spooling stops) by the following:
A /* (delimiter) statement found
A valid JCL statement
An end-of-file condition on reading device

```
//DDNAME      DD *  
               data  
/*
```

SYSIN data must be preceded by a DD statement such as follows:
SYSIN data encountered by JES2 or JES3 following a DD * statement saved on the SPOOL volume for future use. This is known as input spooling. The SYSIN is delimited (the spooling stops) by the following:
A /* (delimiter) statement found
A valid JCL statement
An end-of-file condition on reading device

Instructor Notes:**5.10: In-Stream Data Format**
In-Stream Data: Example

```
//SYSIN      DD *  
1234CG LTD  
2345CG INDUSTRIES  
//DD1 DD DSN=DA0001T.CG.PAYROLL.MASTER,  
.....
```

In-Stream Data: Example

The asterisk (*) is a positional parameter. The DD* is a special statement which is under complete JES2 or JES3 control.

SYSIN is a very common ddname used by many vendor-written programs to pass control information to the utility. For example, SORT, IEBGENER, and IDCAMS utilities.

In user written programs, if you use COBOL ACCEPT statement, then in the run JCL one of the DD statements is SYSIN DD statement.

```
//SYSIN  DD *  
1234  
/*
```

Remark: If SYSIN data is not preceded by DD*, the system generates a statement and place it in front of the SYSIN data.

Instructor Notes:**5.10: In-Stream Data Format
SPECIAL DD STATEMENTS****SYSIN (Input Stream Data)**

An input data stream is data, that supplies data to the load module at the time that the job is submitted. When the OS detects an asterisk * or DATA in the DD statement it pauses for input and input is supplied at this point. When there is no further input required, a /* is entered.

EXAMPLE:

```
//SYSIN DD *  
//SYSIN DD DATA /*
```

- //STEP2 EXEC PGM=LOAD1
- //SYSIN DD *
- HELLO WORLD /* More than one SYSIN statements can be placed for same job or job step.

Instream data to a Cobol program:

In the JCL

```
//STEP010 EXEC PGM=LM of a Cobol program  
//SYSIN DD *  
001 JOHN .....  
/*
```

In the Cobol program:

```
IDENTIFICATION DIVISION.  
DATA DIVISION.  
WORKING-STORGAE SECTION.  
77 DATA1-IN PIC X(80).  
PROCEDURE DIVISION.  
ACCEPT DATA1-IN
```

Instructor Notes:**In-Stream Data: Example**

Note: A line with blanks is the most common offender. It is invisible to the user but it will be treated as data by the system this may or may not cause problem. Let us look at the following example.

The system will interpret the above JCL in the following way:

Conclusion: If there are two or more DD statements by the same name in the same step, this is not an error condition. When the program opens for SYSIN the first of the two be used. The other will be allocated and ignored.

```
//DA0001TA JOB LA2719,....  
//S1      EXEC PGM=ASS1  
//STEPLIB  DD ...  
1234  
//DD1      DD ...
```

```
//DA0001TA JOB LA2719,....  
//S1      EXEC PGM=ASS1  
//STEPLIB  DD ...  
//SYSIN    DD *  (generated statement)  
1234  
//DD1      DD ...
```

```
//DA0001TA JOB LA2719,....  
//S1  EXEC PGM=ASS1  
//STEPLIB  DD ...  
//SYSIN  DD *  
1234  
//DD1  DD ...
```

```
//DA0001TA JOB LA2719,....  
//S1  EXEC PGM=ASS1  
//SYSIN  DD *  
  
//STEPLIB  DD ...  
//SYSIN  DD *  (generated statement)  
1234  
//DD1  DD ...
```

Instructor Notes:**5.11: The SYSOUT Parameter Characteristics**

The SYSOUT parameter can assign sysout or output class, to a dataset. Such datasets are called sysout or output datasets.

This is a keyword parameter.

Format:

- Class: identifies the sysout class of the dataset from A-Z and 0-9

SYSOUT=(class | *)

Characteristics:

Print records generated by a program are not normally routed directly to a physical printer (theoretically it is possible; but in practice, it is seldom done). Instead, they are written on the SPOOL pack, and saved there for later viewing on a terminal or printing (or both). This is called output spooling, and is under the control of JES2 or JES3, which later can use one of their print routines to print the dataset. These print routines must schedule the datasets for printing, and message classes are used for this purpose. All print routines (called printers or writers) are associated with one or more classes (in all 36 classes) and each dataset to be printed must also be assigned classes. The printer routines select datasets for printing in a very similar way as initiators selects jobs for executions. Use S.ST option of the ISPF menu to view the output dataset.

The SYSOUT parameter can assign this class, known as sysout or output class, to a dataset. Such datasets are called sysout or output datasets.

SYSOUT

```
//MFCVT01 JOB (),MSGCLASS=A  
//STEP01 EXEC PGM=PROGRAM1  
//SYSPRINT DD SYSOUT = *  
//  
//MFCVT01 JOB (),CLASS=A  
//STEP01 EXEC PGM=PROGRAM1  
//DD1 DD SYSOUT = A  
//  
//MFCVT01 JOB (),CLASS=A  
//STEP01 EXEC PGM=PROGRAM1  
//DD1 DD DSN=FILE1,SYSOUT = A  
//DD2 DD DSN=FILE2,SYSOUT = B  
//
```

Instructor Notes:

**5.11: The SYSOUT Parameter
Characteristics (contd.)**

Format:

- *: This indicates the same class used in the MSGCLASS parameter of the JOB statement (or the installation-defined default, if MSGCLASS is omitted) is to be used.

Example:

```
//SYSUT1 DD SYSOUT = C
//ddname DD SYSOUT = *
```

E.g. 1

```
//SYSOUT DD SYSOUT=A
```

E.g. 2

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

This DD statement is used for printing system messages generated by JES2 or JES3. Each step must have the SYSPRINT DD statement. Absence causes "SYSPRINT DD STATEMENT MISSING" message in the sysout.

E.g. 3

//SYSOUT DD SYSOUT=*(or any sysout class may be assigned)

This DD statement is used when you have the COBOL DISPLAY clause in your program.

SYSPRINT

Specifies that an execution report of the load module (PGM) is required. It defines the output file containing the execution messages.

EXAMPLE:

```
//STEP1 EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1.....
```

Note:

The SYSOUT parameter specifies the output stream dataset. It routes the report to the class (device) mentioned. SYSOUT=CLASS-code, is the syntax. The class can be any alphanumeric character or an asterisk(*), which refers to the device coded in the MSGCLASS parameter of JOB statement.

Instructor Notes:

[5.12: The DD Statement – Concatenation Characteristics](#)

Only sequential and partitioned datasets can be concatenated.

For sequential datasets, the maximum number of concatenation is 255; and for PDS, it is 16.

Concatenation has meaning only for sequential processing.

Characteristics:

At times, program may have to read in sequence several input datasets as if they were one. This can be accomplished without physically putting the data in one datasets. This is done by concatenating the datasets in JCL code with comparable DCB characteristics without programming changes.

Note that only sequential and partitioned datasets can be concatenated. For sequential datasets, the maximum number of concatenations is 255 and for PDS it is 16. Concatenation has meaning only for sequential processing.

Instructor Notes:**5.12: The DD Statement – Concatenation
Concatenation: Examples**

Concatenation of PS files:

```
//ddname      DD DSN=DA0001T.CG.SEQ1,DISP=SHR
//           DD DSN=DA0001T.CG.SEQ2,DISP=SHR
//           DD DSN=DA0001T.CG.SEQ3,DISP=SHR
```

Concatenation of PDS's:

```
//ddname      DD DSN=DA0001T.PDS1,DISP=SHR
//           DD DSN=DA0001T.PDS2,DISP=SHR
//           DD DSN=DA0001T.PDS3,DISP=SHR
```

Example: Concatenate datasets by coding a DD statement for each dataset

```
//IN       DD      DSN=Jan.DS,DISP=SHR
//           DD      DSN=Feb.DS,DISP=SHR
//           DD      DSN=Mar.DS,DISP=SHR
```

DDname should be coded for the first DD statement only
If DUMMY dataset is coded, the rest of the datasets down are not processed

```
//DD1      DD      DSN=A1.PDS,DISP=SHR
//           DD      DUMMY
//           DD      DSN=C1.PDS,DISP=SHR
```

Instructor Notes:

5.12: The DD Statement – Concatenation

Rules and regulations for Concatenation

LRECL, RECFM must be the same, but BLKSIZE could be different.

The blocksize of the first concatenation must be greater than or equal to the blocksizes of all the subsequent concatenations. Violation of this rule results in S001-4 ABEND failure.

Both sequential datasets and partitioned datasets can be concatenated but not with each other, that is, Sequential with Sequential and Partitioned with Partitioned only.

Rules and Restrictions for Concatenation:

There are number of rules and restrictions for concatenations:

The first concatenation is the only one with a ddname.

The logical record length and the record format of concatenated datasets must be the same. However, the blocksizes need not be.

The blocksize of the first concatenation must be greater than or equal to blocksizes of all subsequent concatenation. Violation of this rule results in S001-04 ABEND failure.

For example, assume that in the JCL below, the first concatenation has a blocksize of 800, the second has a blocksize of 800- and the third has a blocksize of 23400.

Rules and Restrictions for Concatenation (Contd.):

Both sequential datasets and partitioned datasets can be concatenated, but not with each other – sequential with sequential and partitioned with partitioned only. Member of a PDS is treated as sequential dataset and thus can be concatenated with sequential dataset.

For example,

Disk as well as tape datasets can be concatenated but not with each other. Only like devices should be concatenated, disk with disk and tape with tape.

```
//INFILE DD DSN=DA0001T.CG.GROUP1,DISP=SHR,DCB=23400  
//      DD DSN=DA0001T.CG.GROUP2,DISP=SHR  
//      DD DSN=DA0001T.CG.GROUP3,DISP=SHR
```

Instructor Notes:

5.13: The DUMMY Parameter
The DUMMY Parameter - Characteristics

• Format

- Allocates no devices no input/output ; read/write operations.
- It is a positional parameter.
- Physical sequential files and VSAM files can be dummed.
- PDS cannot be dummed.
- PDS (member) is treated as a PS file and therefore can be dummed.

```
//ddname      DD DUMMY
```

The DUMMY Parameter – Characteristics:

The DUMMY parameter is a positional parameter. At times, one might want to execute a program but suppress read or write operations in certain jobs. For example. not print a report. At other times, one might want to test a program without actually processing data. At times, a DD statement referring to a dataset may be coded in the in a JCL in production region .The DUMMY parameter may be coded in a test region when the same JCL is to be executed.

The DUMMY parameter specifies the following:

- No device or external storage be allocated.
- No disposition processing is performed.
- No input or output operations are performed for sequential access methods.

Instructor Notes:

5.13: The DUMMY Parameter

The DUMMY Parameter - Remarks

When an attempt to dummy a PDS is made, it causes an S013-64 ABEND failure.

The DCB parameter may be required while coding dummy. Failure to do so may cause an S013-10 ABEND failure.

NULLFILE is a keyword parameter and is same as DUMMY.

DUMMY provides a safe way to eliminate I/O activity when required.

Instructor Notes:

5.13: The DUMMY Parameter

The DUMMY Parameter - Example

Example

```
//ddname    DD DSN=NULLFILE
```

Instructor Notes:**5.14: The JOBLIB DD Statement
Characteristics**

The JOBLIB DD statement identifies the program library where the programs to be executed throughout the job reside.

It must be placed between the JOB and the first EXEC statement.

Instructor Notes:

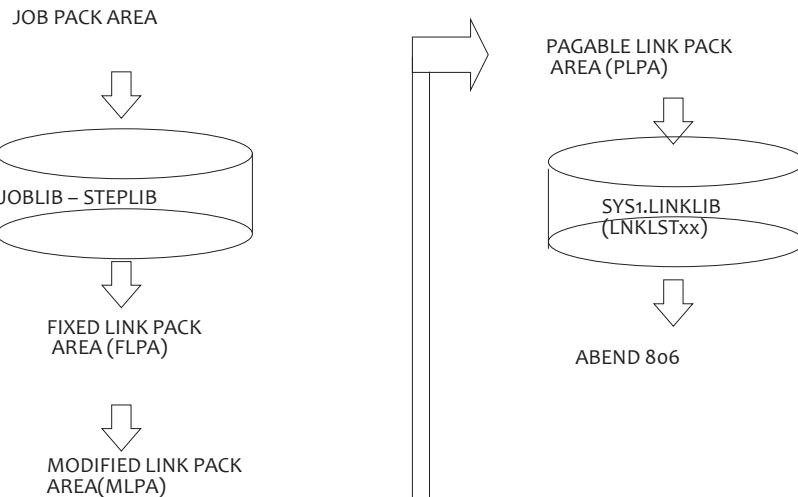
5.14: The JOBLIB DD Statement
SPECIAL DD STATEMENTS

JOBLIB & STEPLIB:

- It is not enough to know the program name to be executed, the system needs to know where that program resides.
- The EXEC statement identifies only the member name only so its location in the system has to be specified by the JOBLIB or STEPLIB statement
- Load modules will be checked first in this library and then in the system libraries and if it is not found in both the places, then JOB would ABEND with S806 code.
- Both are DD statements.

Instructor Notes:

5.14: The JOBLIB DD Statement
PROGRAM MANAGEMENT-SEARCH SEQUENCE



Instructor Notes:

5.14: The JOBLIB DD Statement

PROGRAM MANAGEMENT – SEARCH SEQUENCE

```
//DA0001TA      JOB LA2719,CG,CLASS=A  
//JOBLIB        DD    DSN=DA0001T.LIB.LOAD, DISP=SHR  
// S1           EXEC  PGM = PROGA  
// S2           EXEC  PGM = PROGB
```

- The PROGA Program is expected to reside in DA0001T.LIB.LOAD as a member of a library and the system searches the directory. If not found, system searches certain predefined libraries. If it is still not found, S806 ABEND failure occurs.

A JOBLIB DD statement can have several concatenations (maximum : 16).

Instructor Notes:

5.14: The JOBLIB DD Statement
The JOBLIB DD - Example 2

- All concatenations may be searched to locate a program. If, however, the program is found in a concatenation other than the last one, other concatenations will not be used.
- When duplicate member names exists in different concatenations, the user can decide which one is to be executed by determining the sequence of the concatenations

```
//DA0001TA   JOB LA2719,CG,CLASS=A
//JOBLIB      DD   DSN=DA0001T.LIB.LOAD, DISP=SHR
// S1         EXEC PGM = PROGA
// S2         EXEC PGM = PROGB
```

Instructor Notes:**5.15: The STEPLIB DD Statement
Characteristics**

The STEPLIB DD statement identifies the library, where the program to be executed for the step where STEPLIB is found resides.

It can be placed anywhere after the EXEC statement.

Instructor Notes:

5.15: The STEPLIB DD Statement
Example1

- Program PROGA is expected to reside in PROD.LOADLIB1 as a member of the library.
- If not found, default libraries are searched

```
//PROD213   JOB    SH21, CLASS=P
//S1         EXEC   PGM=PROGA
//STEPLIB     DD     DSN=DA0001T LIB LOAD1,
//                  DISP=SHR
//S2         EXEC   PGM=PROGB
//STEPLIB     DD     DSN = DA0001T LIB LOAD2,
//                  DISP=SHR
```

Instructor Notes:

[5.15: The STEPLIB DD Statement](#)
Example2

- A STEPLIB DD statement has the effect of negating the JOBLIB DD statement for a particular step.

```
//DA0001TA   JOB    LA2719,CG,CLASS=P  
//JOBLIB      DD      DSN=DA0001T.LIB.LOAD1,DISP=SHR  
//S1          EXEC   PGM=PROGA  
//S2          EXEC   PGM=PROGB  
//STEPLIB     DD      DSN =DA0001T.LIB.LOAD2,  
//                      DISP=SHR  
//S3          EXEC   PGM=PROGC
```

Instructor Notes:

5.16: Storage Dump

Requesting a Storage Dump

To request a storage dump, one of the following three DD statements must be included in the step:

- A SYSUDUMP DD statement
- A SYSMDUMP DD statement
- A SYSABEND DD statement

```
//SYSUDUMP DD SYSOUT = *
```

- All the virtual storage allocated to your program, that is, the user region of JOB's private address space, is used.

Requesting a Storage Dump:

When a step encounters an ABEND failure, it is often advantageous to request a virtual storage dump, which can then be helpful in determining the cause of an ABEND. To request a storage dump, one of the following three DD statements must be included in the step:

- A SYSUDUMP DD statement
- A SYSMDUMP DD statement
- A SYSABEND DD statement

```
//SYSUDUMP DD SYSOUT=*
```

All virtual storage allocated to your program, that is, user region of job's address space, is used. It is a formatted dump. SYSUDUMP usually writes to sysout. It can, however, write to a disk dataset, providing a way to preserve the SYSUDUMP information for later viewing and analysis.

No DCB is required.

```
//SYSUDUMP DD DSN=DA0001T.DUMPFILE,SPACE=(TRK,(0,5),RLSE),  
// DISP=(,DELETE,CATLG),UNIT=SYSDA
```

Instructor Notes:**5.16: Storage Dump
The SYSABEND DD Statement**

```
// SYSABEND DD SYSOUT = *
```

- user region + system areas outside the user region that are associated with the job step

```
// SYSMDUMP DD SYSOUT = *
```

- system areas + entire private address space

The SYSABEND DD Statement:

This is same as SYSUDUMP DD statement except for the fact that the dump is unformatted. This type of dump is very difficult to analyze unless it is saved on a disk and then processed by the PRDUMP service aid. SYSMDUMP is seldom used.

```
//SYSMUDUMP DD SYSOUT=*
```

When a SYSUDUMP DD statement is included in a step which ABEND's, a formatted virtual storage dump will be provided. This dump also includes information about the failed step, as well as most of the MVS storage-resident information, which is of no use to the average user.
SYSABEND is intended for system programmer.

```
//SYSABEND DD SYSOUT=*
```

Remark:

If neither a SYSUDUMP nor a SYSMDUMP nor a SYSABEND statement is coded within a JCL of an ABENDING step, a small amount of information is provided. This information is seldom useful in resolving the problem that caused the ABEND failure.

Instructor Notes:**5.16: Storage Dump
The SYSABEND DD Statement - Example**

Remark:

- When more than one of the above statements is included in the JCL of a step, only the last one is used.

```
//SYSDUMP    DD    SYSOUT=*
//SYSDUMP    DD    DSN=DA0001T.DUMP1,
//                           DISP=(,DELETE,CATLG),UNIT=SYSDA,
//                           SPACE=(CYL,(0,5),RLSE)
```

Instructor Notes:

5.16: Storage Dump
SPECIAL DD STATEMENTS

JOBCAT & STEPCAT DD STATEMENT

- The dataset used in the step are first checked in the STEPCAT (ICF or VSAM catalog) before checking in system catalog.
- If no STEPCAT in the step and there is a JOBCAT, then the datasets are first searched in JOBCAT before checking in the system catalog.

Instructor Notes:**5.17: OUTPUT Statement
OUTPUT STATEMENTS****OUTPUT Statement**

- Used to specify processing options for a system output (SYSOUT) data set.
- These processing options are used only when the OUTPUT JCL statement is explicitly or implicitly referenced by a SYSOUT DD statement.
- JES combines the options from this OUTPUT JCL statement with the options from the referencing DD statement.

OUTPUT JCL statements are useful in processing the output of one SYSOUT data set in several ways.

For example, a SYSOUT dataset can be sent to a distant site for printing, as shown in statement OUT1, while it is also printed locally, as shown in statement OUT2:

```
//OUT1 OUTPUT DEST=STLNODE.WMSMITH  
//OUT2 OUTPUT CONTROL=DOUBLE  
//DS DD SYSOUT=C,OUTPUT=(*.OUT1,*.OUT2)
```

Instructor Notes:

5.17: Storage Dump
Lab

Day 2 Lab



Instructor Notes:**Summary**

DD statement must appear in a step to define the resources used by that specific step.

DSN parameter identifies the name of the dataset to be created or retrieved.

DISP parameter specifies how to dispose of the dataset when the step terminates (normally or abnormally).

UNIT and VOLUME parameters work together to specify the location of the dataset.

The SPACE parameter is used to allocate or alter space to a dataset.

The LABEL parameter is used to specify sequence on a volume and type of label of a tape dataset.



Instructor Notes:**Summary**

The DCB parameter specifies characteristics that are stored in the Data Control Block of the file.

The SYSOUT parameter can assign the sysout or output class to a dataset.

Only sequential and partitioned datasets can be concatenated.

The DUMMY parameter allocates no devices no input/output and read/write operations.

The JOBLIB and STEPLIB DD and statement identifies the program library.

The STORAGE DUMP is used place a request for a storage dump.



Instructor Notes:**Review Question**

Question 1. Which of the following is/are positional parameter?

- Option1 : DUMMY
- Option2 : STORAGE DUMP
- Option3 : JOBLIB

Question 2. Which of the following statements is/are true?

- Option1 : A JOBLIB statements can have several concatenations.
- Option2 : A STEPLIB negates the effect of JOBLIB DD statement.



Instructor Notes:



Job Control Language (JCL)

Lesson 06 Procedure

©2016 Capgemini. All rights reserved.

The information contained in this document is proprietary and confidential.
For Capgemini only.

Instructor Notes:**Lesson Objectives**

Catalog procedures

Instream procedures

EXEC and DD statement overriding

Symbolic parameters and Symbolic overrides

Set and Include Statement



Instructor Notes:

6.1: PROCEDURES

Description

A procedure is basically a set of standard job steps which are invoked to execute a function within a single job.

In a working environment the same JCL can be utilized by several users.

The use of procedure helps minimize duplication of code & probability of error because a procedure consist of pre-tested statements.

The names of the procedures can be recorded and stored like any other members of PDS.

Sometimes the same JCL can be utilized by different users but with different parameters, in such cases JCL permits to override one parameter with another.

Instructor Notes:

6.1: PROCEDURES

Invoking a procedure

Invoking procedure

- //STEPNAME EXEC PROC=PROCNAME
- //STEPNAME EXEC PROCNAME

Restriction

- Max of 255 steps

Instructor Notes:**6.1: PROCEDURES**
Description

Following are not permitted to reside in a procedure:

- JOB statement
- EXEC statement that invokes a procedure.
- JOBLIB
- JOBCAT
- DD * or DATA
- // null statement
- PEND statement
 - If any of the above is included in a procedure then "Invalid statement in procedure" error is displayed while executing the procedure.

Instructor Notes:**6.1: PROCEDURES**
Description (Contd...)

There are two types of procedures :

- CATALOGED PROCEDURE
 - Member of a PDS, often referred to as procedure library or PROCLIB.

INSTREAM PROCEDURES

- Contained within a job's input stream.

Instructor Notes:**6.1: PROCEDURES**
Catalogued Procedure

A procedure can be cataloged by placing it in one of three types of proclibs:

- SYS1.PROCLIB – IBM-supplied system procedure library.
 - System PROCLIBs – defined by an installation.
 - A user-defined PROCLIB – OS/390 or MVS/ESA SP V4 or Higher
- The catalogued procedure is a set of JCL statements that refer to a procedure stored as library (proclib) i.e. Code JCLLIB ORDER statement for Catalogued procedures stored in user private libraries.
 - It is a member of a PDS. This procedure can be used by any number of jobs.
 - The procedure name must be unique within the procedure library in which it is placed.
 - The PEND statement is not required.

Instructor Notes:

6.1: PROCEDURES
Catalogued Procedure (Contd...)

Example: Procedure Library: MAINUSR.JCL.CNTL(JCLPROC)

```
//JCLPROC      PROC
//STEP1 EXEC PGM=IEFBR14
//DD1   DD DSN=MAINUSR.PROC1.PROC,
//
//DISP=(NEW,CATLG,DELETE),VOL=SER=LP2WK1,
//          UNIT=SYSDA,SPACE=(TRK,(1,1,1),RLSE),
//
//DCB=(BLKSIZE=800,LRECL=80,RECFM=FB)
//SYSPRINT    DD SYSOUT=*
//          PEND
//          /*
```

Main program

```
//JOBNAME   JOB      A123,'SUSAN JOHN',.....
//DD1        JCLLIB ORDER=(MAINUSR.JCL.CNTL,...)
//CREAT EXEC  PROC=JCLPROC
//SYSIN DD    DUMMY
///*
```

```
//PROC1 PROC
//STEP01 EXEC PGM=FIRST
//DD1   DD DSN=SPEC.A.INFILE1,DISP=SHR
//STEP02 EXEC PGM=SECOND
//DD1   DD DSN=SPEC.A.INFILE2,DISP=SHR
//          PEND
```

The segment of JCL that executes this above procedure

```
//MYJOB JOB (),CLASS=A
//PROCLIB JCLLIB ORDER=SPECTRUM.A.CBL
//BATCH1 EXEC PROC1
```

At Runtime:

```
//MYJOB  JOB
//PROCLIB JCLLIB ORDER=SPECTRUM.A.CBL
//* PROC1 PROC=PROC1
//* This is what the system visualizes
// STEP01 EXEC PGM=FIRST
//DD1   DD DSN=SPEC.A.INFILE1,DISP=SHR
//STEP02 EXEC PGM=SECOND
//DD1   DD DSN=SPEC.A.INFILE2,DISP=SHR
```

Instructor Notes:

6.1: Demo
Demo

Example 1
(Catalogued procedure)



Instructor Notes:**6.1: PROCEDURES**
Instream Procedure

An in-stream procedures is a part of a job's input stream and exists only for the duration of the job.

PROC statement in an in-stream procedure is mandatory and serves two functions:

- It signals the beginning of in-stream procedure.
- It contains default symbolic overrides.

PEND statement must be coded in an in-stream procedure to provide a delimiter.

As the name says it is a set of JCL statements which is contained within an input stream of a job.

This procedure can be used by only one job but can be executed any number of times within a job.

Must begin with PROC & end with PEND statement

Must be coded immediately after the JOB statement and before the first EXEC statement.

No more than 15 in-stream procedures can be coded in one JOB.

```
//MFCVT01 JOB (),CLASS=A  
//PROC1 PROC  
//STEP01 EXEC PGM=FIRST  
//DD1 DD DSN=SPEC.A.INFILE1,DISP=SHR  
//STEP02 EXEC PGM=SECOND  
//DD1 DD DSN=SPEC.A.INFILE2,DISP=SHR  
//      PEND  
//STEP03 EXEC PROC1
```

Instructor Notes:6.1: PROCEDURES
Example of Instream Procedure

```
//DA0001TA   JOB  LA2719,CG,MSGCLASS=A,  
//                           MSGLEVEL=(1,1),NOTIFY=DA0001T  
//* INSTREAM PROCEDURE  
//PROCBR14  PROC  
//S1          EXEC  PGM=IEFBR14  
//SYSPRINT   DD    SYSOUT=*  
//DD1         DD    DSN=DA0001T.TEMP,  
//                           DISP=(OLD,DELETE)  
//                           PEND  
//*Invoking of procedure named PROCBR14  
//STEP1       EXEC  PROC=PROCBR14
```

Instructor Notes:

6.1: Demo
Demo on Instream Procedure

Example 2
(Instream procedure)



Instructor Notes:

6.1: Notes

Notes on Instream Procedure

An Instream procedure can call a Cataloged procedure.

A Cataloged procedure cannot call an Instream procedure.

A Cataloged procedure can call a Cataloged procedure.

There can exist a maximum of 15 nested 'PROC statements' – PROC operation

- If each procedure resides in 15 different PDS, then the JCLLIB ORDER must identify those PDS to the OS.

Instructor Notes:**6.1: THE PROC STATEMENT**
Description

The purpose of the PROC statement is to contain symbolic override defaults.

When a procedure is executed, the system substitutes symbolic parameters using symbolic overrides coded in the EXEC statement.

- For symbolic overrides not found in the EXEC statement, default symbolic overrides in the PROC statement are used.

Instructor Notes:

6.2: Rules to Override JCL Procedures
Common Rules for EXEC & DD Statement

Parameter:

- Can be replaced, added or nullified.
- When you replace an existing parameter, the overriding parameter must be specified in its complete format.
- DCB is an exception.
- An overriding parameter replaces the same parameter, if it exists. It is added to the statement if it does not exist
- A syntactical JCL error inside a procedure cannot be corrected by overriding the erroneous parameter.

Instructor Notes:**6.2: Rules to Override JCL Procedures**
Rules for EXEC Statement

To override an EXEC parameter:

- Code "parameter.stepname=value" when you add or replace a parameter.
- Code "parameter.stepname=" when you nullify a parameter.

PGM parameter cannot be overridden.

Complete all overrides to EXEC parameters for a step before you override parameters in a subsequent step.

Within a particular step the sequence of overriding parameters is not important.

Instructor Notes:

6.2: Rules to Override JCL Procedures
Description

Add or Remove an EXEC statement by overriding.

All overriding EXEC parameters must be coded in the EXEC statement that invokes the procedure.

All overrides to EXEC parameters must be completed before overriding parameters in a subsequent step.

Instructor Notes:

6.2: Rules to Override JCL Procedures

Rules for DD statement overriding

To override any parameter in a DD statement, an independent DD statement must be supplied in the following format:

```
//stepname.ddname DD overriding parameters
```

Add an entire DD statement:

```
// stepname.ddname DD complete parameter field must be  
coded
```

Instructor Notes:6.2: Rules to Override JCL Procedures
Description

To override any parameter in a concatenation other than the first one, code the following:

```
//stepname.ddname DD
//          DD
//          .
//          .
//          DD overriding parameters
```

Instructor Notes:**6.2: Rules to Override JCL Procedures
Description (Contd...)**

Sequence of overriding DD statements must be the same as the sequence of the corresponding overridden statements.

The sequence of overriding parameters is not important, except for positional parameters.

An additional DD statement must be the last one in a step's overriding statements. When several additional DD statements are supplied, their relative sequence is not important, unless referbacks are used.

A DD statement cannot be removed by overriding.

Overriding DD statements of a Procedure

```
//MYJOB JOB
//MYPROC PROC
//STEP1 EXEC PGM=IEFBR14
//DD1 DD DSN=MAINUSR.JCL.PDS,DISP=SHR,
//                                         VOL=SER=LP1WK1,LRECL=80
//     PEND
//STEP2 EXEC MYPROC
//STEP1.DD1 DD VOL=SER=LP2WK1,LRECL=
//STEP1.DD2 DD DSN=MAINUSR.COPYLIB,DISP=SHR
AT RUNTIME
//MYJOB JOB
//STEP1 EXEC PGM=IEFBR14
//DD1 DD DSN=MAINUSR.JCL.PDS,DISP=SHR,VOL=SER=LP2WK1
//DD2 DD DSN=MAINUSR.COPYLIB,DISP=SHR
```

In the above example

- In DD1, LP1WK1 was changed to LP2WK1
- The LRECL parameter was nullified (discarded)
- A new DD statement DD2 was added

Overriding EXEC statements of a Procedure

```
//MYJOB JOB NOTIFY=USERID
//MYPROC PROC
//STEP1 EXEC PGM=IEBGENER,TIME=NOLIMIT,REGION=4M
//SYSUT1 DD DUMMY
//SYSUT2 DD DUMMY
//     PEND
//STEP2 EXEC MYPROC,TIME.STEP1=10,REGION.STEP1=
AT RUNTIME
//MYJOB JOB NOTIFY=USERID
//STEP1 EXEC PGM=IEBGENER,TIME=10
//SYSUT1 DD DUMMY
//SYSUT2 DD DUMMY
```

In the example above:

The value of the TIME parameter has been changed from NOLIMIT to 10
The REGION parameter has been nullified

Instructor Notes:

6.2: Rules to Override JCL Procedures

Example – Procedure LAM (S1)

```
//S1          EXEC PGM=ED, PARM=(A,B,C,E)
//                      REGION=900K, TIME=(5,30)
//STEPLIB      DD DSN=DEV.LOADLIB,DISP=SHR
//IN1          DD DSN=USER1.FILE2,DISP=SHR
//IN2          DD DSN=USER1.FILEX,DISP=OLD,
//                      UNIT=TAPE, VOL=SER=000101
//REP          DD SYSOUT =*,,
//OUT          DD DSN=USER1.PLA,DISP=(,CTLG,DELETE),
//                      UNIT=SYSDA,VOL=SER=BS3003,
//                      SPACE=(CYL,(20,5),DCB=(BLKSIZE=4000,
//                      LRECL=80, RECFM=FB)
//
```

Instructor Notes:

[6.2: Rules to Override JCL Procedures](#)
Example

Following is required in step S1:

- PARM must be (A,B,C,D) and TIME nullified.
- In IN1, DSN must be USER1.FILE3.
- IN2 must retrieve USER1.FILEX as a cataloged dataset.
- In OUT, BLKSIZE must be 23440.

Instructor Notes:

6.2: Rules to Override JCL Procedures

Example - Procedure LAM (S2)

```
//S2    EXEC  PGM=FORM,REGION=900K
//INA   DD    DSN=USER1.PLA,DISP=SHR
//      DD    DSN=USER1.F226,DISP=SHR
//      DD    DSN=USER1.F232,DISP=SHR
//      DD    DSN=USER1.F118,DISP=SHR
//OUTA  DD    DSN=USER.F323, DISP=(,CATLG,DELETE),
//                  UNIT=TAPE, VOL=SER=001110,
//                  DCB=BLKSIZE=32700, LRECL=100,
//                  RECFM=FB)
//PRNT  DD  SYSOUT=*
```

Instructor Notes:**6.2: Rules to Override JCL Procedures**
Example

Following is required in step S2:

- COND = (0, LT) must be coded.
- In INA DSN in the third concatenation must be USER1.F228.
- In DD statement OUTA, UNIT be SYSDA.
- An entire DD statement:
 - //STEPLIB DD DSN=DEV.LOADLIB,DISP=SHR must be coded.

Instructor Notes:

6.2: Rules to Override JCL Procedures

Example - Procedure LAM (S3)

Following is required in step S3:

- EVEN must be added to the COND parameter.
- In DD statement OUT3, RLSE must be removed and VOLUME parameter must be nullified.

```
//S3      EXEC PGM=REPO,REGION=400K,COND=(O,LT)
//IN3     DD    DSN=USER1.F333,DISP=OLD
//OUT3    DD    DSN=USER1.F111,DISP=(,CTLG,DELETE),
//              UNIT=SYSDA,VOL=SER=DEV012,
//              SPACE=(CYL,(50,15),RLSE),
//              DCB=(BLKSIZE=23440,LRECL=80,RECFM=FB)
//PRINT DD
/
```

Instructor Notes:

6.2: Rules to Override JCL Procedures

Solution

```
//ZP          EXEC LAM PARM.S1=(A,B,C,D),TIME.S1=,
//                      COND.S2=(0,LT), COND.S3=((0,LT),EVEN)
//S1.IN1      DD   DSN=USER1.FILE3
//S1.IN2      DD   VOL=           ALTERNATIVE:
VOL=SER=
//S1.OUT      DD   DCB=BLKSIZE=23440
//S2.INA      DD
//
//          DD   DSN=USER1.F228
//S2.OUTA     DD   UNIT=SYSDA
//S2.STEPLIB   DD   DSN=DEV.LOADLIB,DISP=SHR
//S3.OUT3      DD   SPACE=(CYL,(50,15)),VOL=
```

Instructor Notes:6.2: Rules to Override JCL Procedures
Some Typical Examples

Example 1

```
//S1          EXEC PGM=ONE
//OUT1        DD    DSN=U1.S1,
//              DISP=(,CTLG,DELETE),
//              UNIT=TAPE,
//              DCB=(BLKSIZE=32700)
```

Required:

OUT1 must be dummied.

- Override:

Regardless of the contents, no other parameters are needed.

```
//S1.OUT1     DD DUMMY
```

Instructor Notes:6.2: Rules to Override JCL Procedures
Some Typical Examples (Contd...)

Example 2:

```
//S1      EXEC PGM=ONE
//IN1      DD    DSN=U1.B1, DISP=SHR
//          DD    DSN=U1.B2, DISP=SHR
//          DD    DSN=U1.B3, DISP=SHR
```

Required:

- Second concatenation of IN1 must be dummy

Override:

```
//S1.IN1   DD
//DD    DSN=U1.B3
//DD    DUMMY
```

Overriding and dummying the second concatenation causes the third concatenation to also act as DUMMY.

```
//S1.IN1   DD
//          DD    DUMMY
```

Instructor Notes:

6.2: Rules to Override JCL Procedures Some Typical Examples (Contd...)

Example 3:

```
//S1      EXEC PGM=ONE
//CNTL DD  DSN=U1.CNTLIB(S1), DISP=SHR
```

Required:

- DD statement CNTL must be //CNTL DD*

Override:

- Regardless of the contents DD * will override all

```
//S1.CNTL    DD *
```

Instructor Notes:

6.2: Rules to Override JCL Procedures
Some Typical Examples (Contd...)

Example 4:

```
//S1      EXEC PGM=ONE
//OUT4 DD  DSN=U1.D1, DISP=NEW
//          DISP=SYSDA, VOL=SER=TEST26,
//          SPACE=CTRK,(500,50)),
//          DCB=(BLKSIZE=23400,
//          LRECL=100,RECFM=FB)
```

Required:

- DCB parameter must be eliminated.

Override:

```
//S1.OUT4    DD DCB=(BLKSIZE=,LRECL=,RECFM=)
```

Instructor Notes:**6.3: Symbolic Parameters & Symbolic Overrides**
Description**Symbolic Overrides**

- Can be used only when symbolic parameters have been coded inside the procedure.

Symbolic Parameter:

- Name preceded by an ampersand (&).
- Can be coded in place of any parameter, part of a parameter in the parameter field of an EXEC, DD or OUTPUT statement.

The default values for the symbolic parameter can be coded in the PROC statement. To override the default parameter you will have to code the values for the symbolic parameter in the EXEC statement that invokes the procedure.

The method of overriding existing DD statements is easily prone to errors due to the rigid sequencing requirements imposed by OS.

An alternative approach is to anticipate which JCL parameters may change, and to define these as SYMBOLIC parameters in the procedure.

Symbolic parameters are used to override parameters on the DD statements & used both in Catalogued & In-stream PROC.

The same JCL can be used by different users to implement common task, such as opening, reading, writing of datasets.

A value assigned to a symbolic parameter may be overridden by another value, as long as the redefinition is within the same job. If it is not overridden then the same value will be assigned to it each time it is called.

If Positional parameters are coded as symbolic then a period should be inserted between them.

Symbolic overrides can be used only when symbolic parameters have been coded inside the procedure.

```
//MYPROC PROC A=LP2WK1,B=SYSDA2
//STEP1 EXEC PGM=IEFBR14
//DD1  DD DSN=MAINUSR.ABC.INPUT,DISP=SHR,VOL=SER=&A,UNIT=&B
//      PEND
//STEPX  EXEC MYPROC,A=LP1WK1,B=SYSSQ
AT RUNTIME
//STEP1 EXEC PGM=IEFBR14
//DD1  DD DSN=MAINUSR.ABC.INPUT,DISP=SHR,
//      VOL=SER=LP1WK1,UNIT=SYSSQ
```

In the above example, A and B are symbolic parameters. Instead of hard-coding the values for the VOL and UNIT parameters, they have been assigned the values contained in the symbolic parameters.

In case we do not change the value of the Symbolic parameters when the Procedure is called, the default values specified at the Procedure-declaration statement (PROC) are taken.

Instructor Notes:

6.3: Symbolic Parameters & Symbolic Overrides

Symbolic Parameter - Example 1

First period works as a delimiter.

```
//S1 EXEC PGM=BL  
//IN DD DSN=&HQ..INFILE, DISP=SHR  
//OUT DD DSN=&HQ..OUTFILE,DISP=,CATLG,DELETE),  
// UNIT=SYSDA, DCB=(BLKSIZE=32700)  
  
//PSK EXEC BLTX, HQ=PROD
```

Instructor Notes:

6.3: Symbolic Parameters & Symbolic Overrides

Symbolic Parameter - Example 2

Procedure BLTX:

```
//S1 EXEC PGM=BL  
//IN DD DSN=&HQ.INFILE, DISP=SHR  
//OUT DD DSN=&HQ.OUTFILE, DISP=,CATLG,DELETE),  
// UNIT=SYSDA, DCB=(BLKSIZE=32700)  
  
//PSK EXEC BLTX, HQ='PROD.'
```

Instructor Notes:**6.3: Symbolic Parameters & Symbolic Overrides**
Symbolic Overriding

Rules for Symbolic Overriding:

- EXEC statement keyword (TIME, REGION etc.) cannot be used as a symbolic parameter.
- Symbolic override in either the EXEC or PROC statement that has no corresponding parameter in the procedure results in a 'SYMBOL NOT DEFINED' JCL error.
- In a symbolic and regular override conflict, the regular override always prevails.

Instructor Notes:

6.3: Symbolic Parameters & Symbolic Overrides
Symbolic Overriding (Contd...)

- A symbolic parameter which is immediately followed by an alphabetic, numeric or national character must have a period at its end.
- A symbolic parameter can be coded many times in a procedure. During substitution, all occurrences receive the same value.
- When nothing must be substituted for a symbolic parameter, "symbolic-override=" must be coded in the EXEC or PROC statements.

Instructor Notes:

6.3: Symbolic Parameters & Symbolic Overrides

Symbolic Overriding - Example 1

- Assume that possible values that PARM parameter assumes are ALD, BLD, CLD, etc.

```
//S1 EXEC PGM = P1, PARM = &PEL
```

- The above example does not work.

```
//S1 EXEC PGM=P1, PARM=&PELLD
```

Instructor Notes:

6.3: Symbolic Parameters & Symbolic Overrides

Symbolic Overriding - Example 1 (Contd...)

- Procedure SSP can be coded as:

```
//SS1 EXEC PGM=P1, PARM = &PEL.LD
```

- Now if the procedure is invoked:

```
//A EXEC SSP, PEL=A
```

- Substitution results in:

```
//S1 EXEC PGM=P1, PARM = ALD
```

Instructor Notes:

6.3: Symbolic Parameters & Symbolic Overrides
Symbolic Overriding - Example 1 (Contd...)

```
//A EXEC SSP, PEL=FLD
```

Substitution results in:

```
//S1 EXEC PGM=P1, PARM=FLD
```

Instructor Notes:6.3: Symbolic Parameters & Symbolic Overrides
Symbolic Overriding - Example

Procedure SSP:

```
//B EXEC SSP, PEL=FLD, TIME = (5, 10)
```

Substitution results in:

```
//S1 EXEC PGM=P1, PARM=FLD,TIME=(5, 10)
```

Instructor Notes:6.3: Symbolic Parameters & Symbolic Overrides
Symbolic Overriding - Example

```
//ABC  PROC  R=800K, Q=AUX, U=TAPE
//S1   EXEC  PGM=P2, REGION=&R
//IN   DD    DSN=&Q..FILEX, DISP=SHR
//OUT  DD    DSN=&Q..FILEY, DISP=(, CATLG,DELETE),
//                  UNIT = &U
//A    EXEC  SWP, Q=MAX
```

Substitution results in:

```
//S1   EXEC  PGM=P2, REGION=800K
//IN   DD    DSN=MAX.FILEX, DISP=SHR
//OUT  DD    DSN=MAX.FILEY,DISP=(,CATLG,DELETE),
//                  UNIT = TAPE
```

Instructor Notes:

6.3: Demo on Symbolic Overriding
Demo

Example 3
(Symbolic Overriding)



Instructor Notes:**6.4: Statements
The SET statement**

The SET statement is another way of assigning values to Symbolic parameters.

```
//MYPROC PROC A=LP2WK1,B=SYSDA
//STEP1 EXEC PGM=IEFBR14
//DD1 DD DSN=MAINUSR.INPUT.FILE1,DISP=SHR,
//           VOL=SER=&A,UNIT=&B
//           PEND
//SET1  SET A=LP1WK1
//SET2  SET B=SYSSQ
//STEPX EXEC MYPROC
```

AT RUNTIME, the SET statement can appear anywhere in a JCL between the JOB statement and the first point where a SET – statement-assigned symbolic parameter is referenced.

```
//STEP1 EXEC PGM=IEFBR14
//DD1 DD DSN= MAINUSR.INPUT.FILE1,DISP=SHR,
//           VOL=SER=LP1WK1,
//           UNIT=SYSSQ
```

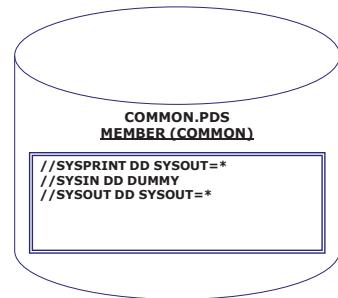
The SET Statement

```
//MCVT01A JOB NOTIFY=&SYSUID
//SETPROC PROC A=MCVT01
//STEP1 EXEC PGM=IEFBR14
//DD1 DD DSN=MFCVT01.GTP74.SET6,
//           UNIT=SYSDA,VOL=SER=&A,
//           DCB=(LRECL=80,BLKSIZE=800,RECFM=FB),
//           SPACE=(TRK,(2,1)),DISP=(NEW,CATLG,DELETE)
//           PEND
//SET  SET A=USER02
//STEP2 EXEC PROC=SETPROC
//SYSIN DD DUMMY
//
```

Instructor Notes:**6.4: Statements
The INCLUDE statement**

The INCLUDE statement allows you to copy statements from any member.

Similar to the way PROCs are used, INCLUDE allows you to code a single set of JCL statements that you can use in multiple jobs.

**The Include Statement**

```
//MCVT01A JOB NOTIFY=&SYSUID
//DD1 JCLLIB ORDER=(MCVT01.GTP74.SATYA)
//STEP1 EXEC PGM=IEFBR14
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//DD1 DD DSN=MCVT01.GTP.FILEX, DISP=(NEW,CATLG,DELETE),
// VOL=SER=USER1,UNIT=SYSDA, SPACE=(TRK,(2,1)),
// DCB=(LRECL=80,BLKSIZE=800,RECFM=FB)
//INC1 INCLUDE MEMBER=PDSSTAT
```

And INCLUDE STATEMENTS(PDSSTAT)

```
//SYSPRINT DD SYSOUT=*
//SYSIN DD DUMMY
//
```

Instructor Notes:

6.4: Statements The INCLUDE statement (contd..)

Example:

```
//MYJOB JOB
//DD1 JCLLIB ORDER=COMMON.PDS
//STEP1 EXEC PGM=MYPGM
//INDD DD DSN=A.B.C,DISP=SHR
//INC1 INCLUDE MEMBER=COMMON
```

AT RUNTIME

```
//MYJOB JOB
//DD1 JCLLIB ORDER=COMMON.PDS
//STEP1 EXEC PGM=MYPGM
//INDD DD DSN=A.B.C,DISP=SHR
//*INC1 INCLUDE MEMBER,COMMON
//SYSPRINT DD SYSOUT=*
//SYSIN DD DUMMY
//SYSOUT DD SYSOUT=*
```

Instructor Notes:

6.4: Demo Demo on Statements

Set and Include Statements.



Instructor Notes:

6.5: Lab
[Lab](#)

Day 3 and Day 4 Labs



Instructor Notes:**Summary**

CATALOG PROCEDURES is a member of a PDS, which is often referred to as procedure library, or just PROCLIB.

INSTREAM PROCEDURES is contained within job's input stream.

Symbolic overrides can be used only when symbolic parameters have been coded inside the procedure.

**Summary**

Instructor Notes:**Review Question**

Question 1: Which of the following must be present in case of instream procedure?

- PROC
- PEND
- INSTREAM

Question 2: A PROC statement in cataloged procedure is optional.

- True/ False



Instructor Notes:



Job Control Language (JCL)

Lesson 07 Utility

©2016 Capgemini. All rights reserved.
The information contained in this document is proprietary and confidential.
For Capgemini only.

Instructor Notes:

Lesson Objectives



IEFBR14 Utility

IEBGENER Utility

IEBCOPY

IEHLIST

SORT Utility

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED

2

Instructor Notes:

Overview of JCL Utilities



Utilities are IBM-supplied programs that are intended to perform certain routine and frequently occurring tasks.

Utilities are used in DASD, tape drives, print and punch operations.

Utilities are used to allocate, update, delete, catalog and uncatalog data sets, and also to list the contents of VTOC (Volume Table of Contents).

MVS provides a number of pre-written utility programs that can be used for maintaining and organizing data.

IEBGENER, IEBCOPY, IEFBR14, IEBCOMPR

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED

3

Utilities are broadly classified into two different categories:

Data set Utilities (prefixed with IEB)
System Utilities (prefixed with IEH)

Data set utilities are used to copy, print, update, reorganize, and compare data at the dataset and/or record level.

System utilities are used to list VTOC information, copy, delete, catalog and uncatalog datasets, to write tape labels and to add or delete dataset passwords.

Instructor Notes:

7.1: IEFBR14 Utility

Example 1

Commonly used to delete, allocate and to un-catalog dataset.

- Example 1:

```
//DELETE      EXEC PGM=IEFBR14
//* TO DELETE A FILE
//SYSPRINT      DD   SYSOUT=*
//DD1           DD   DSN=DA0001T.EMPLOYEE,
//                  DISP=(MOD,DELETE,DELETE),
//                  UNIT=SYSDA, SPACE=(TRK,0)
```

© 2018 Capgemini. All rights reserved.

 OurUniversity
EFMD ELP ACCREDITED

4

Instructor Notes:

7.1: IEFBR14 Utility

Example 2

```
//CREATE      EXEC PGM=IEFBR14
//* TO ALLOCATE A NEW FILE
//SYSPRINT      DD SYSOUT=*
//DD1          DD DSN=DA0001T.EMPLOYEE,
//                  DISP=(NEW,CATLG,DELETE),
//                  UNIT=SYSDA, SPACE=(TRK,(2,1)),
//                  DCB=(BLKSIZE=800,LRECL=80,
//                  RECFM=FB,DSORG=PS)
```

© 2018 Capgemini. All rights reserved.

 OurUniversity
EFMD ELP ACCREDITED

5

Instructor Notes:

7.2: IEFBR14 Utility

**Demo****IEFBR14 utility**

- Show a code illustrating, the deletion of a dataset, before COBRUN1 step is executed.

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED

6

An except of a code illustrating, the deletion of a dataset, before COBRUN1 step is executed.

```
//DA0001TA    JOB   LA2819,CG,NOTIFY=DA0001T,MSGCLASS=X,  
MSGLEVEL=(1,0)  
//DELETE      EXEC  PGM=IEFBR14  
//SYSPRINT          DD    SYSOUT=*  
//LOGFILE     DD  
DSN=DA0001T.MYFILE2,DISP=(MOD,DELETE,DELETE),  
//  SPACE=(TRK,(0),),UNIT=SYSDA  
/*  
//COBRUN1    EXEC  PGM=ASS1, PARM='AAAA'  
//STEPLIB    DD    DSN=DA00021T,PATNI.LOADLIB,DISP=SHR  
//SYSPRINT    DD    SYSOUT=*  
//INFILE     DD    DSN=DA00021T,EMPLOYEE,DISP=OLD  
//OUTFILE    DD    DSN=DA00021T.MYFILE2,DISP=(NEW,CATLG,  
DELETE),  
// DCB=(LRECL=80, DSORG=PS, BLKSIZE=80, RECFM=FB),  
// VOL=SER=BS3011, SPACE=(TRK, (45, 15))  
//SYSOUT     DD    SYSOUT  
//
```

Instructor Notes:

7.2: IEBGENER Utility

**Description****Uses:**

- To copy, concatenate and empty sequential datasets.
- To reformat records while copying
 - Can be compared to selecting specific columns
- To specify conditions while copying
 - Can be compared to selecting specific rows
- To re-block copied records
 - Changing the LRECL and BLKSIZE
- To concatenate datasets
- To write instream data into a dataset

© 2018 Capgemini. All rights reserved.

OurUniversity

7

To copy sequential datasets**Example**

```
//STEP010 EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DSN=<Input dataset to be Read from>
//SYSUT2 DD DSN=<Output dataset to be Written into>
//SYSIN DD DUMMY
//
```

The input dataset can be a PS or a member of PDS

The output dataset can also be a PS or a PDS member

Output datasets as PS apply to backing-up operations on tapes

The SYSPRINT DD statement defines the message dataset.

The SYSUT1 DD statement defines the input dataset.

The SYSUT2 DD statement defines the output dataset (can not have multiple SYSUT2).

The SYSIN DD statement defines the control dataset. This is where IEBGENER looks for any utility control statements. When DUMMY is specified, there are no control statements being used.

To copy sequential datasets

To copy an input sequential dataset to many members of a PDS Code
Generate Maxname and Member name.

```
//DSRC012A JOB NOTIFY=DSRC012
//STEP010 EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DSN=<Input dataset>
//SYSUT2 DD DSN=<Output PDS>
//SYSIN DD *
   Generate Maxname=3
   (To create 3 members in the output)
   Member name=(mem1,mem2,mem3)
   (mem1,mem2,mem3 will have the contents of sysut1)
/*
```

Instructor Notes:**7.2: IEBGENER Utility****Sample**

```
//DA0001TA      JOB      LA2719,CG,NOTIFY=DA0001T,  
//                         MSGCLASS=X  
//*****  
/*  
/* USING THE IEBGENER Utility TO EMPTY EXISTING DATASET  
//*****  
/*  
/CPYSTEP EXEC PGM=IEBGENER  
//SYSPRINTDD   SYSOUT=*  
//SYSUT1 DD     DUMMY, DCB=(BLKSIZE=800,  
//                           LRECL=80,RECFM=FB)  
//SYSUT2 DD     DSN=DA0001T.MYOUT,DISP=OLD  
//SYSIN DD      DUMMY  
//
```

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED

8

Instructor Notes:

7.2: IEBGENER Utility

**Demo**

IEBGENER Utility to merge the data from two sequential files
To empty the existing dataset

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED

9

```
//DA0001TA      JOB          LA2719, CG, NOTIFY=DA0001T,  
//                                         MSGCLASS=X  
//*****  
/* USING THE IEBGENER Utility TO MERGE DATASETS SYSUT1  
PROVIDING  
/* THE INPUT AND SYSUT2 BEING THE OUTPUT  
//*****  
//CPYSTEP      EXEC      PGM=IEBGENER  
//SYSPRINT     DD      SYSOUT=*  
//SYSUT1       DD      DSN=DA0001T.INDATA1, DISP=SHR  
//SYSUT2       DD      DSN=DA0001T.NEW,DISP=MOD  
//SYSIN        DD      DUMMY  
//
```

Instructor Notes:

7.2: IEBGENER Utility



Reformatting Data

Reformat data during copy

- By reformatting, you can select data-bytes to be output
 - Code Generate Maxflds and Record Field in the control statement (similar to Inrec Fields in Sort utility)
- Say, to output only the Empnum and Salary data-bytes, in the order of Salary and Empnum

```
//STEP010 EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DSN=<Input dataset>
//SYSUT2 DD DSN=<Output dataset>
//SYSIN DD *
Generate Maxflds=2
Record field=(5,1,CH,6)
field=(5,46,CH,1)
(Length,Location in input,Format,Location in the output)
```

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD-EQUIS ACCREDITED 10

Example:-

```
//JOBNAME JOB NOTIFY=IGTRN30
//STEP01 EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DSN=IGTRN30.CG.INFILE,DISP=SHR
//SYSUT2 DD DSN=IGTRN30.CG.OUTFILE,DISP=OLD
//** LENGTH,START POSITION,,DESTINATION IN OUTFILE
//SYSIN DD *
GENERATE MAXFLDS=2
RECORD FIELD=(4,4,,10),FIELD=(10,4,,20)
/*
//
```

Instructor Notes:

7.2: IEBGENER Utility

Reformatting Data (Contd...)



And to have some character-literals in the output, say, two asterisks between Salary and Empnum fields (similar to 'X' in the Sort utility), code Maxlits:

```
//STEP010 EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DSN=<Input dataset>
//SYSUT2 DD DSN=<Output dataset>
//SYSIN DD *
Generate Maxflds=2, Maxlits=2
Record field=(5,1,CH,8)
          field=(5,46,CH,1)
          (Length,Location in input, Format, Location in the output)
          field=(2,'**',,6)
```

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD-EQUIS ACCREDITED 11**Example2:-**

```
/IGTRN30A JOB NOTIFY=IGTRN30
//STEP010 EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DSN=IGTRN30.CG.INFILE,DISP=SHR
//SYSUT2 DD DSN=IGTRN30.CG.PDS,DISP=OLD
//SYSIN DD *
  GENERATE MAXNAME=3
  MEMBER NAME=(MEM1,MEM2,MEM3)
/*
```

Instructor Notes:

7.2: IEBGENER Utility

Concatenating Datasets

To concatenate datasets

```
//DSRC012A   JOB    NOTIFY=DSRC012
//STEP010     EXEC   PGM=IEBGENER
//SYSPRINT    DD      SYSOUT=*
//SYSUT1       DD      DSN=<First unsorted
dataset>
//           DD      DSN=<Second unsorted
dataset>
//SYSUT2       DD      DSN=<The concatenated
dataset>
//SYSIN        DD      DUMMY
//
```

© 2018 Capgemini. All rights reserved.

 12

Instructor Notes:

7.2: IEBGENER Utility

Writing Instream Data

To write instream data

```
//DSRC012A   JOB      NOTIFY=DSRC012
//STEP010    EXEC      PGM=IEBGENER
//SYSPRINT   DD       SYSOUT=*
//SYSUT1     DD       *
<Instream data>.....
/*
//SYSUT2     DD       DSN=<Output dataset to be
Written>
//SYSIN     DD       DUMMY
//
```

© 2018 Capgemini. All rights reserved.

 OurUniversity
EFMD ELP ACCREDITED 13

Instructor Notes:

7.2: IEBGENER Utility

**Demo**

IEBGENER Utility to copy the reformatted data

IEBGENER Utility to concatenate the data

IEBGENER Utility to write the instream data into a sequential file

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 14

Instructor Notes:**7.3: IEBCOPY Utility****Description**

The IEBCOPY is used to copy members of partitioned datasets.

The COPY statement identifies the input and output files by referring to their DDNAMES in the JCL.

The format is:

- COPY OUTDD=output-DDname , INDD=input-Ddname

```
//MFCVT01A JOB NOTIFY=MFCVT01
//STEP1 EXEC PGM=IEBCOPY
//SYSPRINT DD      SYSOUT=*
//IN     DD DSN=MFCVT01.FILE1,DISP=SHR
//OUT    DD DSN=MFCVT01.FILE2,DISP=SHR
//SYSIN  DD  *
COPY OUTDD=OUT,INDD=IN
/*
```

The above example copies all of the members from the PDS, 'MFCVT01.FILE1' to an existing PDS, 'MFCVT01.FILE2'.

The IN and OUT DD statements define data sets to be used by IEBCOPY.

The COPY control statement specifies the input and output DDNAMES.

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 15

Instructor Notes:

7.3: IEBCOPY Utility

Copying Members

Copying Specific Members

- The SELECT statement identifies the members of the PDS to be copied. The format is:
 - SELECT MEMBER=NAME (to specify a single member)

```
//MFCVT01A JOB '0.2AMIP',....  
//STEP1 EXEC PGM=IEBCOPY  
//SYSPRINT DD SYSOUT=*  
//IN DD DSN= MFCVT01.FILE1,DISP=SHR  
//OUT DD DSN= MFCVT01.FILE2,DISP=SHR  
//SYSIN DD *  
      COPY OUTDD=OUT,INDD=IN  
      SELECT MEMBER=ALLOCATE  
/*  
'MFCVT01.FILE1 to an existing PDS, MFCVT01.FILE2.'
```

© 2018 Capgemini. All rights reserved.

 OurUniversity
EFMD ELP ACCREDITED

16

Instructor Notes:

7.3: IEBCOPY Utility

Copying Members (Contd...)

Copying Multiple Specific Members

- SELECT MEMBER=(NAME,NAME,NAME) (to specify multiple members)

```
// MFCVT01A JOB '0.2AMIP',.....
//STEP1 EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=*
//IN    DD DSN= MFCVT01.FILE1,DISP=SHR
//OUT   DD DSN= MFCVT01.FILE2,DISP=SHR
//SYSIN  DD *
      COPY OUTDD=OUT,INDD=IN
      SELECT MEMBER=(FILE1,FILE2,FILE3)
/*
* The above example copies selected members FILE1,FILE2 and FILE3 from the
* PDS, 'MFCVT01.FILE1' to an existing PDS, 'MFCVT01.FILE2'.
```

© 2018 Capgemini. All rights reserved.

 OurUniversity
EFMD ELP ACCREDITED

17

Instructor Notes:

7.3: IEBCOPY Utility

Copying Members (Contd...)

Copying and Renaming Specific Members

```
//MFCVT01A JOB '0.2AMIP',....  
//STEP1 EXEC PGM=IEBCOPY  
//SYSPRINT DD SYSOUT=*  
//IN DD DSN=MFCVT01.FILE1,DISP=SHR  
//OUT DD DSN=MFCVT01.FILE2,DISP=SHR  
//SYSIN DD *  
      COPY OUTDD=OUT,INDD=IN  
      SELECT MEMBER=(JOBA,(PROD,TEST,R))  
/*
```

- The above example copies the member called "PROD" from the PDS, 'MFCVT01.FILE1' to an existing PDS, 'MFCVT01.FILE2'.
- The SELECT control statement specifies:
 - Copy the member JOBA
 - the member PROD is to be copied in the following manner
 - rename PROD to TEST,
 - copy the renamed member TEST to the output dataset,
 - if a member by that name exists in the output dataset replace it.

© 2018 Capgemini. All rights reserved.

Instructor Notes:

7.3: IEBCOPY Utility

Copying Members (Contd...)

Copying Using EXCLUDE

```
//MFCVT01A   JOB '0.2AMIP',....  
//STEP1      EXEC PGM=IEBCOPY  
//SYSPRINT DD   SYSOUT=*  
//IN        DD  DSN=MFCVT01.FILE1,DISP=SHR  
//OUT       DD  DSN=MFCVT01.FILE2,DISP=SHR  
//SYSIN     DD  *  
             COPY OUTDD=OUT,INDD=IN  
             EXCLUDE MEMBER=ALLOCATE  
/*  
▪ The above example copies all members MFCVT01.FILE1 except the member  
'ALLOCATE'
```

© 2018 Capgemini. All rights reserved.

Instructor Notes:

7.3: IEBCOPY Utility

Copying Members (Contd...)



Compressing Data Sets

```
//MFCVT01A JOB '0.2AMIP',....  
//STEP1 EXEC PGM=IEBCOPY  
//SYSPRINT DD SYSOUT=*  
//INPDS DD DSN=MFCVT01.FILE1,DISP=SHR  
//SYSUT3 DD UNIT=SYSDA,SPACE=(TRK,(1,1))  
//SYSUT4 DD UNIT=SYSDA,SPACE=(TRK,(1,1))  
//SYSIN DD *  
      COPY INDD=INPDS,OUTDD=INPDS  
/*
```

- The above example compresses the library MFCVT01.FILE1.
- Notice that the same DD name is specified in both the INDD and OUTDD parameters.

© 2018 Capgemini. All rights reserved.

OurUniversity EFMD-CLIL ACCREDITED 20

Instructor Notes:

7.2: IEBGENER Utility

Demo

IEBGENER Utility to copy the partitioned dataset
To copy the selected members
Usage of exclude members
Usage of compressing the dataset



© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 21

Instructor Notes:

7.4: IEHLIST Utility



Description

The IEHLIST utility is used to

- list entries in a DASD VTOC (Volume Table of Contents)
- list entries in a PDS Directory.
- list entries in a system catalog

Example 1:

```
//STEP1 EXEC PGM=IEHLIST
//SYSPRINT DD SYSOUT=*
//DD1 DD DISP=OLD,UNIT=SYSDA,VOL=SER=ABC
//DD2 DD DISP=OLD,UNIT=SYSDA,VOL=SER=DEF
//SYSIN DD *
      LISTVTOC FORMAT,VOL=SYSDA=ABC
      LISTVTOC FORMAT,VOL=SYSDA=DEF
          DSNAME=(MTPL.FILE1,MTPL.FILE2)
/*
//
```

X

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 22

Instructor Notes:

7.4: IEHLIST Utility



Description (Contd...)

The above example uses IEHLIST to print two VTOC listings:

The IEHLIST looks for utility control statements coded below the SYSIN DD statements:

The first LISTVTOC control statement requests an formatted (FORMAT) VTOC listing for pack ABC. This includes DSCB and space allocation information. If FORMAT is omitted, an abbreviated version is listed.

The second LISTVTOC control statement requests a formatted VTOC listing for two datasets: MTPL.FILE1 and MTPL.FILE2.

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 23

Instructor Notes:

7.4: IEHLIST Utility



Description (Contd...)

Example 2:

```
//STEP1 EXEC PGM=IEHLIST  
//SYSPRINT DD      SYSOUT=*  
//DD1    DD  DISP=OLD,UNIT=SYSDA,VOL=SER=ABC  
//SYSIN  DD  *  
        LISTPDS DSNAME=MTPL.FILE,VOL=SYSDA=ABC  
/*
```

- The above example uses IEHLIST to list entries in a PDS directory.
- The LISTPDS control statement requests a listing of the directory for the PDS, MTPL.FILE.

NOTE: DSNAME cannot be abbreviated as DSN on a control statement.

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 24

Instructor Notes:

7.5: SORT Utility



Description

Provided by MVS

Commonly used to:

- sort data
- copy selective data
- remove duplicates
- change data throughout the file

Reorders Physical Sequential dataset as per requirement on given field(s).

- These fields are called *control* or *key* fields.

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD EQUIS ACCREDITED 25

Instructor Notes:

7.5: SORT Utility



Description (Contd...)

Working

- Assumes all input records to be out of sequence.
- Puts them in a sequence you request.
 - Example: Employee data is sorted in the sequence of Emp. no., Emp.name or Salary etc.

© 2018 Capgemini. All rights reserved.

Instructor Notes:

7.5: SORT Utility



The SORT Utility - DFSort

To reformat data to be sorted

- Can be compared to selecting specific columns
- To specify conditions for selecting data
- Can be compared to selecting specific rows

To concatenate datasets and sort

```
//SYSIN DD *  
      SORT FIELDS=(1,5,CH,A)  
      /*
```

- The SORT statement supports a SORT KEY mapping.
- Start sorting the record at the absolute byte address (1),
- Length, the number of bytes to be included in sorting (5) ,
- Format of sorting (EBCDIC character),
- Sequence of sorting (ascending / descending)

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 27

Instructor Notes:

7.5: SORT Utility

Format

Sort fields=(position, length, format, sequence)

or

Sort fields=(position,length,sequence....),format=format

Syntax is used if all fields on which the dataset to be sorted are of same type:

Position: Location of input record's 1st byte of the key field

Length: Length in bytes of the key field.

Sum of all key fields (lengths) should not exceed 4092.

© 2018 Capgemini. All rights reserved.

 OurUniversity
EFMD EQUIS ACCREDITED

28

Instructor Notes:

7.5: SORT Utility



Format (Contd...)

- Format: Two characters code identify the data format (type).
- Sequence:
 - A – Ascending
 - D – Descending

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD EQUIP ACCREDITED

29

Instructor Notes:

7.5: SORT Utility



DFSORT - Description

DFHSORT:

- Member of IBM's Data Facility family of products.

DFSORT:

- Licensed program. High-performance data arranger.
- Developed by IBM for MVS users.
- Sort, merge, and copy data sets.
- Aids complex tasks such as inventory or billing system management.
- Record-level editing capability to perform data management tasks.

© 2018 Capgemini. All rights reserved.

 OurUniversity
EFMD-EQUIP ACCREDITED 30

Sorting Data Sets

You can use DFSORT to rearrange the records in your datasets. Sorting is arranging records in either ascending or descending order within a file.

The fields in the records can be in any IBM System/370 format (for example EBCDIC character, decimal, and binary)

You can sort data in several different formats. Following table shows the most common data formats and the codes you use to specify them.

Data Format	Code
EBCIDIC (Character)	CH
Binary (Numeric)	BI
Zoned Decimal (Numeric)	ZD
Packed Decimal (Numeric)	PD

Instructor Notes:**Merging Data Sets**

You can also use DFSORT to merge data sets. DFSORT merges data sets by combining two or more files of sorted records to form a single data set of sorted records.

You can merge up to 16 data sets. The data sets you merge must be previously sorted into the same order (ascending or descending order).

The JCL needed for a merge is the same as that for a SORT, with the following exceptions:

- You do not use the SORTWKnn statement
- Instead of SORTIN DD statement, you use SORTINnn DD statements to define the input datasets. The SORTINnn DD statements name the input datasets to be merged and tell how many datasets are to be merged. The value nn in SORTINnn is a number from 0 to 16, indicating the number of datasets to be merged.

Copying Data Sets

DFSORT can also copy data sets without any sorting or merging taking place. You copy data sets in much the same way that you sort or merge them.

What else can you do with DFSORT?

While sorting, merging, or copying data sets, you can also:

- Select a subset of records from an input data set. You can include or omit records that meet specified criteria.
- Reformat records, add or delete fields, and insert blanks, constants, or binary zeros. For example, you can make a report more legible by inserting blank characters to separate fields.
- Sum the values in selected records while sorting or merging (but not while copying).
- Alter the collating sequence when sorting or merging records (but not while copying). For example, you can have the lowercase letters collate after the uppercase letters.

Instructor Notes:**Creating and Running DFSORT Jobs**

Processing data sets with DFSORT involves two steps:

1. Creating a DFSORT job
2. Running a DFSORT job.

You can run a DFSORT job by invoking processing in a number of ways stated as follows:

- With a JCL EXEC statement using the name of the program or the name of the catalogued procedure.
- With interactive panels supported under ISPF and ISMF.
- Within programs written in COBOL, PL1, or basic Assembler language.

Remarks: JCL-invoked means that the DFSORT program is initiated by JCL EXEC statement. The phrase dynamically invoked means that the DFSORT program is initiated from another program.

The JCL statements you need for most jobs are described as follows:

//jobname JOB

Signals the beginning of a job.

//stepname EXEC

Signals the beginning of a job step and tells the operating system what program to run.

//stepname EXEC PGM=SORT

//STEPLIB DD

defines the library containing DFSORT program. If your DFSORT program is in system library, you can omit the STEPLIB statement.

//SYSOUT DD

DD defines the output data set for messages.

//SORTIN DD

DD defines the input data set.

//SORTWKnn DD

DD defines a work storage data set for a sort. For most

applications, one work storage data set is sufficient. Increasing the number of work storage data sets does not improve performance.

//SORTOUT DD

defines the output dataset.

//SYSIN DD

Control statements.

All the control information within SYSIN DD can be coded freely between column 2 and column 71.

Instructor Notes:

7.5: SORT Utility

SORT 1 JCL - Example

```
//DA0001T JOB LA2719,CG, NOTIFY=DA0001T, MSGCLASS=X
//** SORT ON THE EMPLOYEE NAME IN ASCENDING /*ORDER
///SRTSTEP    EXEC   PGM=SORT
//SYSIN      DD   *
      SORT FIELDS=(1,5,CH,A)
/*
//SORTIN      DD   DSN=DA0001T.EMPLOYEE,DISP=SHR
//SORTOUT     DD   DSN=DA0001T.OUTSORT,
//                  DISP=(NEW,CATLG, DELETE),
//                  SPACE=(TRK,(3,3)),UNIT=SYSDA
//                  SPACE=(TRK,(10,5)), UNIT=SYSALLDA
```

© 2018 Capgemini. All rights reserved.

 OurUniversity
EFMD ELP ACCREDITED

33

Instructor Notes:

7.5: SORT Utility

SORT 1 JCL - Example (Contd...)

```
//SORTWK02 DD SPACE=(TRK,(10,5)), UNIT=SYSALLDA  
//SYSPRINT DD SYSOUT=*  
//SYSOUT DD SYSOUT=*  
//SORTMSG DD SYSOUT=*  
//
```

© 2018 Capgemini. All rights reserved.

 OurUniversity
EFMD ELP ACCREDITED

34

Instructor Notes:

7.5: SORT Utility

SORT 1 JCL - Example (Contd...)

```
//STEP1 EXEC PGM=SORT
//SYSOUT DD SYSOUT=*
//SYSPRINT DD SYSOUT=A
//SORTIN DD DSN=MAINUSR.SEQ1.INPUT,DISP=OLD
//SORTOUT DD DSN=MAINUSR.SEQ2.OUTPUT,DISP=OLD
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(20,10),RLSE)
//SYSIN DD *
          SORT FIELDS=(21,2,CH,A)
/*
```

The above example will sort the records of the input dataset specified in the SORTIN DD statement based on the field specified in the control statement of the SYSIN DD. The sorted dataset is copied to the output dataset specified in the SORTOUT DD statement.

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 35

Instructor Notes:

7.5: SORT Utility

Demo

SORT Utility (SORT JCL 1)



© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 36

Instructor Notes:

7.5: SORT Utility

Sorting by Multiple fields - JCL 2

```
//DA0001T JOB LA2719,CG,NOTIFY=DA0001T, MSGCLASS=X  
//* SORTS ON ASCENDING DEPTNO & DESCENDING ENAME  
//SRTSTEP      EXEC   PGM=SORT  
//SYSIN        DD    *  
//              SORT FIELDS=(17,2,PD,A,2,6,CH,D)  
/*  
//SORTIN        DD    DSN=DA0001T.DEPT,DISP=SHR,  
//                      SPACE=(TRK,(3,3)),UNIT=SYSDA  
//SORTOUT       DD    DSN=DA0001T.SORTOUT2,  
//                      DISP=(NEW,CATLG, DELETE)
```

© 2018 Capgemini. All rights reserved.

 OurUniversity
EFMD ELP ACCREDITED

37

Instructor Notes:

7.5: SORT Utility

Sorting by Multiple fields - JCL 2 (Contd...)

```
//SORTWK01    DD      SPACE=(TRK,(10,5)), UNIT=SYSALLDA  
//SORTWK02    DD      SPACE=(TRK,(10,5)), UNIT=SYSALLDA  
//SYSPRINT    DD      SYSOUT=*  
//SYSOUT      DD      SYSOUT=*  
//SORTMSG     DD      SYSOUT=*  
//
```

© 2018 Capgemini. All rights reserved.

 OurUniversity
EFMD ELP ACCREDITED

38

Instructor Notes:

7.5: SORT Utility JCL 2 and 3

**Demo****SORT Utility**

- Simple SORT
- SORT on multiple fields

© 2018 Capgemini. All rights reserved.

 39
EFMD E-Learning ACCREDITED

You can further sort the records in the data set by specifying multiple control fields. When you specify two or more control fields, you specify them in the order of greater to lesser priority.

Instructor Notes:

7.5: SORT Utility



Copying Data Sets

With DFSORT, copy data sets directly without performing a sort or merge.

- Use any of the following:
 - SORT FIELDS=COPY
 - MERGE FIELDS=COPY
 - OPTION COPY

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD E-Learning ACCREDITED 40

You can use COPY with all of the other DFSORT control statements except SUM. DFSORT can select and reformat the specific data sets you want to copy by using the control statements covered later.

Instructor Notes:

7.5: SORT Utility

Copying Data Sets - JCL 4

```
//DA0001TA JOB LA2719,CG, NOTIFY=DA0001T,MSGCLASS=X
//***** *****
*** *****
//*****
//*****
*** *****
//SRTSTEP      EXEC   PGM=SORT
//SYSIN DD *
  SORT FIELDS = COPY
/*
//SORTIN       DD      DSN=DA0001T.DEPT,DISP=SHR
//SORTOUT      DD      DSN=DA0001T.SORTOUT2,
  //          DISP=(NEW,CATLG,DELETE),
  //          SPACE=(TRK,(3,3)),UNIT = SYSDA
//SYSPRINT     DD      SYSOUT=*
//SYSOUT       DD      SYSOUT=*
//SORTMSG      DD      SYSOUT=*
//
```

© 2018 Capgemini. All rights reserved.

OurUniversity EPICLIP ACCREDITED 41

The JCL for a copy application is the same as for a sort, except that you do not use the SORTWKnn DD statement.

Instructor Notes:

7.5: SORT Utility JCL 4



Demo

SORT Utility

(SORT JCL 4)

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 42

Instructor Notes:

7.5: SORT Utility

Tailoring Input Data Set with INCLUDE and OMIT

**Tailor Data Sets:**

- You may need only a subset of the data set records for any application. Hence, you can tailor data sets.
- Increase the speed of the sort, merge, or copy.
 - Fewer the records, lesser is the time taken to process them.
- Steps to tailor an input data set:
 - Use an INCLUDE control statement to collect wanted records.
 - Use an OMIT control statement to exclude unwanted records.
 - Your choice of INCLUDE and OMIT depends on which is easier and more efficient to write for a given application.

Note: INCLUDE and OMIT control statements are mutually exclusive.

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD E-Learning ACCREDITED 43

Instructor Notes:

7.5: SORT Utility



Tailoring Input Data Set with INCLUDE and OMIT (Contd...)

Select from the following comparison operators:

Comparison Operators	Meaning
EQ	Equal to
NE	Not Equal to
GT	Greater than
GE	Greater than or Equal to
LT	Less than
LE	Less than or equal to

© 2018 Capgemini. All rights reserved.

Instructor Notes:

7.5: SORT Utility



Rules to Tailor Input Data Set with INCLUDE and OMIT

DFSORT uses following rules to pad and truncate strings.

- Padding adds fillers in data, usually zeros or blanks.
- Truncation deletes or omits leading or trailing portions.
- Field-to-field Comparison: Shorter field is padded as appropriate (with blanks or zeros).
- Field-to-Constant Comparison:
 - Constant is padded or truncated to the length of the field.
 - Decimal constants are padded or truncated on the left.
 - Character and hexadecimal constants are padded or truncated on the right.

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 45

Instructor Notes:

7.5: SORT Utility

Allowable Comparisons: INCLUDE and OMIT

Following table shows field-to-field and field-to-constant comparisons:

Field Format	BI	CH	ZD	PD
BI	✓	✓		
CH	✓	✓		
ZD			✓	✓
PD			✓	✓

© 2018 Capgemini. All rights reserved.

 46

Instructor Notes:

7.5: SORT Utility



SORT JCL 5 - Copy Selective Data

INCLUDE COND copies data that matches a condition.

- Example:
 - In this case it copies data with one character in the 19th position that equals 'M' or 'S'.

```
//DA0001TA JOB LA2719,CG,NOTIFY=DA0001T,MSGCLASS=X  
//* SORTS ON THE INPUT FILE ON JOB AND SELECTS JOB BEGINING  
//* WITH M OR S INTO A NEW DATASET  
//SRTSTEP EXEC PGM=SORT  
//SYSIN DD *  
OPTION EQUALS  
SORT FIELDS=(19,6,A),FORMAT=CH  
INCLUDE COND=(19,1,CH,EQ,C'M',OR,19,1,CH,EQ,C'S')  
/*
```

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 47

Instructor Notes:

7.5: SORT Utility

SORT JCL 5 - Copy Selective Data (Contd...)

```
//SORTIN      DD  DSN=DA0001T.INDATA3,DISP=SHR  
//SORTOUT     DD  DSN=DA0001T.SORTOUT3, DISP=(NEW,CATLG),  
//                           SPACE=(TRK,(3,3)), UNIT=SYSDA,  
//                           DCB=(BLKSIZE=800, LRECL=80,RECFM=FB,  
//                           DSORG=PS)  
//SORTWK01    DD  SPACE=(TRK,(10,5)),UNIT=SYSALLDA  
//SORTWK02    DD  SPACE=(TRK,(10,5)),UNIT=SYSALLDA  
//SYSOUT      DD  SYSOUT=*  
//
```

© 2018 Capgemini. All rights reserved.

 OurUniversity
EFMD ELP ACCREDITED 48

Instructor Notes:

7.5: SORT Utility JCL 5

Demo

SORT Utility (SORT JCL 5)



© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 49

Instructor Notes:

7.5: SORT Utility



SORT 6 - Sort Selective Data

Sort on Job.

- Select jobs that begin with "M" and Deptno begins with **1**.

```
//DA0001TA JOB LA2719,CG,NOTIFY=DA0001T,MSGCLASS=X  
///* SORTS ON JOB INCLUDES JOBS BEGINING WITH M AND  
DEPTNO  
///* BEGINNING WITH 1  
//SRTSTEP EXEC PGM=SORT  
//SYSIN DD *  
OPTION EQUALS  
SORT FIELDS=(19,6,A),FORMAT=CH  
INCLUDE COND=(19,1,CH,EQ,C'M',AND,51,1,CSF,EQ,1)  
/*
```

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD E-Learning ACCREDITED

50

Instructor Notes:

7.5: SORT Utility

SORT 6 - Sort Selective Data (Contd...)

```
//SORTIN      DD  DSN=DA0001T.INDATA3,DISP=SHR  
//SORTOUT     DD  DSN=DA0021T.SORTOUT4, DISP=(NEW,CATLG),  
//                           SPACE=(TRK,(3,3)), UNIT=SYSDA,  
//                           DCB=(BLKSIZE=800,LRECL=80,RECFM=FB,  
//                           DSORG=PS)  
//SORTWK01    DD  SPACE=(TRK,(10,5)),UNIT=SYSALLDA  
//SORTWK02    DD  SPACE=(TRK,(10,5)),UNIT=SYSALLDA  
//SYSOUT      DD  SYSOUT=*  
//
```

© 2018 Capgemini. All rights reserved.

 OurUniversity
EFMD ELP ACCREDITED

51

Instructor Notes:

7.5: SORT Utility JCL 6

Demo

SORT Utility (SORT JCL 6)



© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 52

Instructor Notes:

7.5: SORT Utility



SORT JCL 7 - OMIT COND

INCLUDE and OMIT are mutually exclusive.

- Records that do not satisfy the condition are sorted and copied into the output dataset.
- Example: Sorts on Job and omits Jobs that begin with M or S.

```
//DA0001TA JOB LA2719,CG,NOTIFY=DA0001T,MSGCLASS=X  
//** SORTS ON JOB OMITS JOBS BEGINNING WITH M OR S  
//SRTSTEP      EXEC PGM=SORT  
//SYSIN       DD *  
          OPTION EQUALS  
          SORT FIELDS=(19,6,A),FORMAT=CH  
          OMIT COND=(19,1,CH,EQ,C'M',OR,19,1,CH,EQ,C'S')  
/*
```

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD E-Learning ACCREDITED 53

Instructor Notes:

7.5: SORT Utility

SORT JCL 7 - OMIT COND (Contd...)

```
//SORTIN      DD  DSN=DA0001T.INDATA3,DISP=SHR  
//SORTOUT     DD  DSN=DA0001T.SORTOUT5,DISP=(NEW,CATLG),  
//                  UNIT= SYSDA,SPACE=(TRK,(3,3)),  
//                  DCB=(BLKSIZE=800, LRECL=80,RECFM=FB,  
//                  DSORG=PS)  
//SORTWK01    DD SPACE=(TRK,(10,5)),UNIT=SYSALLDA  
//SORTWK02    DD SPACE=(TRK,(10,5)),UNIT=SYSALLDA  
//SORTWK03    DD SPACE=(TRK,(10,5)),UNIT=SYSALLDA  
//SORTWK04    DD SPACE=(TRK,(10,5)),UNIT=SYSALLDA  
//SYSOUT      DD  SYSOUT=*  
//
```

© 2018 Capgemini. All rights reserved.

 OurUniversity
EFMD ELP ACCREDITED

54

Instructor Notes:

7.5: SORT Utility JCL7

Demo

SORT Utility (SORT JCL 7)



© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 55

Instructor Notes:

7.5: SORT Utility

SORT JCL 8

```
//DA0001TA   JOB LA2719,CG, NOTIFY=DA0001T,MSGCLASS=X
//* Merges fields beginning with column 110 having length 5
//* INDATA1 and INDATA2 are sorted on the control field
//SRTSTEP      EXEC   PGM=SORT
//SORTIN01     DD DSN=DA0001T.INDATA1,DISP=OLD
//SORTIN02     DD DSN=DA0001T.INDATA2,DISP=OLD
//SORTOUT      DD DSN=DA0001T.SORTOUT4,DISP=(NEW,CATLG),
//                  SPACE=(TRK,(3,3,)), UNIT = SYSDA,
//                  DCB=(BLKSIZE=800, LRECL=80,
//                  RECFM=FB,DSORG=PS)
//SYSOUT       DD SYSOUT=*
//SYSIN DD *
MERGE        FIELDS = (110,5,A),FORMAT=CH
/*
```

© 2018 Capgemini. All rights reserved.

 OurUniversity
EFMD-EQUACCREDITED 56

Instructor Notes:

7.5: SORT Utility JCL 8

Demo

SORT Utility (SORT JCL 8)



© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 57

Instructor Notes:

7.5: SORT Utility



Writing Constants

Formats to write character strings, hexadecimal strings and decimal numbers are as follows:

- Character Strings

- Format for writing a character string is: **C'x.....x'** where x is an EBCDIC character. For example, C'Sheela'.
 - If you wish to include a single apostrophe in the string, you must specify it as two single apostrophes. For example, O'NEILL must be specified as **C'O''NEILL'**.

- Hexadecimal Strings

- Format for writing a hexadecimal string is: **X'yy.....yy'** where yy is a pair of hexadecimal digits. For example X'C1C2' is equivalent to C'AB'.

- Decimal Strings

- Format for writing a decimal number is:
n....n or +n....n or -n...n
where n....n is a decimal digit. Examples are 24, +24, and -24.
 - Decimal number must not contain commas and decimal points.

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD CLILP ACCREDITED 58

Instructor Notes:

7.5: SORT Utility



Summing Records – SUM Statement

Department TRG wishes to know the total salary of all trainers.

- Use the INCLUDE statement to tailor the file to include only records for the TRG department.
- Use SORT and SUM to get the sum of salaries.

On the SUM control statement,

- Specify one or more numeric fields to be summed whenever records have equal control fields.
 - Control fields are specified on the SORT statement.
- Numeric fields: Binary, Packed Decimal, Zoned Decimal.

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 59

Instructor Notes:

7.5: SORT Utility

Summing Records – SUM Statement (Contd...)



When you sum records, keep in mind that two types of fields are involved:

- Control fields specified on the SORT statement.
- Summary fields specified on the SUM statement.

Writing the SUM Statement:

```
SUM FIELDS=(location, length, data-format,....)
```

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 60

Instructor Notes:

7.5: SORT Utility

Summing Records – SUM Statement (Contd...)

Example

- INCLUDE, SORT, and SUM statements are shown below:

```
INCLUDE COND=(26,4,CH,EQ,C'TRG ')
  SORT FIELDS=(26,4,CH,A)
    SUM FIELDS=(35,5,BI)
```

- Returns the total salary of the TRG department.

© 2018 Capgemini. All rights reserved.

Instructor Notes:

7.5: SORT Utility

Summing Records – SUM Statement (Contd...)

Final sum appears in the SALARY field of one record.

- Other records are deleted.

By default, records with equal control fields appear in the original order.

When summing records keeping the original order, DFSORT chooses the first record to contain the original sum.

© 2018 Capgemini. All rights reserved.

 OurUniversity
EFMD E-Learning ACCREDITED

62

Remark: Some of the fields in your summation might not be meaningful, such as the employee number field. You could use the OMIT statement to omit this field. There are two other ways to leave out fields that are not meaningful.

Instructor Notes:

7.5: SORT Utility



Suppress Records with Duplicate Control Fields

Use SUM to delete records with duplicate control fields.

- Specify FIELDS=NONE on the SUM statement.
- Example: List all the distinct departments in ascending order.

```
SORT FIELDS=(25,4,CH,A)  
SUM FIELDS=NONE
```

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 63

Instructor Notes:

7.5: SORT Utility



Overflow

Occurs when a sum becomes larger than the space available for it.

If it occurs, the two records involved are left unsummarized.

- Contents of the records are left undisturbed
- Neither record is deleted
- Records are still available for summarization

Does not prevent further summary

- Correctable in some cases
- Use INREC control statement to pad summary fields with zeros

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 64

Instructor Notes:

7.5: SORT Utility



Reformatting Records

Reformat records in your data sets using OUTREC and INREC control statements

- Delete fields
- Reorder fields
- Insert separators (blanks, zeros, or constants)

Difference from DFSORT control statements:

- OUTREC reformats records after they are sorted, copied, or merged
- INREC reformats records before they are sorted, copied, or merged

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 65

Instructor Notes:

7.5: SORT Utility



Reformatting Records (Contd...)

INREC and OUTREC perform the same functions.

Consider their processing order when you choose which to use:

- Use INREC to delete fields
 - Shorter records take less time to sort, merge, or copy (INREC reformats the records before they are processed).
- Use OUTREC to insert separators.
 - Inserts separators into records after they are processed.
- To reorder fields, use either control statements.
 - This does not affect the record length.

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 66

Note: If you use INREC or OUTREC to change the record length, be sure to specify the final record length on the SORTOUT DD statement using the DCB parameter. The final length is either:

The INREC length if you are using just INREC.

The OUTREC length if you are using just OUTREC or both INREC and OUTREC.

Instructor Notes:

7.5: SORT Utility



Reformatting Records - OUTREC

Delete all unrequired fields for the application

- Fields without meaningful contents in a summation record.
- **Note:** In an OUTREC statement, you do not specify the data format.

```
SORT FIELDS=(26,4,CH,A)
SUM FIELDS=(35,5,BI)
OUTREC FIELDS=(26,4,35,5)
```

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD EQUIS ACCREDITED 67

Instructor Notes:

7.5: SORT Utility



Reformatting Records - OUTREC (Contd...)

As the record length changed, specify the new length on the SORTOUT DD statement.

- For example:

```
//SORTOUT DD DSN=DA0001T.SORTOUT,  
//           DISP=(NEW,CATLG,DELETE),  
//           SPACE=(TRK,(1,1)),UNIT=SYSDA,  
//           DCB=LRECL=9
```

© 2018 Capgemini. All rights reserved.

 OurUniversity
EFMD ELP ACCREDITED 68

Instructor Notes:

7.5: SORT Utility



Reorder Fields to Reserve Space

Fields always appear in the order in which you specify them.

Therefore, if you wish for salary to appear before department, simply reverse the order in the OUTREC statement.

```
SORT FIELDS=(26,4,CH,A)  
SUM FIELDS=(35,5,BI)  
OUTREC FIELDS=(35,5,26,4)
```

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 69

Instructor Notes:

7.5: SORT Utility



Inserting Binary Zeros

Assume you want to reformat the records to include a new 4-byte binary field after the salary field (beginning at byte 39). In this case, you can insert binary zeros as placeholders for the new field (to be filled in with data at later date).

- To insert the zeros, write 4Z after the last field:
- This time, you must specify on the SORTOUT DD statement the new record length is 13 bytes.

```
SORT FIELDS=(26,4,CH,A)
SUM FIELDS=(35,5,BI)
OUTREC FIELDS=(26,4,35,5,4Z)
```

© 2018 Capgemini. All rights reserved.

 OurUniversity
EFMD E-Learning ACCREDITED 70

You can insert binary zeros before, between, or after fields. You can use Z or 1Z to specify a single binary zero.

Instructor Notes:

7.5: SORT Utility



Inserting Blanks

Make a printout more legible with OUTREC

- Separate fields with blanks.
- Create margins.
 - Example: Print only employee number and employee name fields.

SORT FIELDS=(1,4,ZD,A)

- Specify OUTREC FIELDS=(10x,1,4,,4x,5,20)
- Insert blanks before, between, or after fields.
 - Use X or 1X to specify a single space.

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 71

If an output data set contains only character data, you can print it by writing the SORTOUT DD statement as follows:

```
//SORTOUT DD SYSOUT=*
```

Instructor Notes:

7.5: SORT Utility



Inserting Constants

Using OUTREC, insert constants to set up a report format.

Formats to write constants are shown below:

- Character Strings
 - Format to write a character string is: C'x....x' where x is an EBCDIC character.
For example, C'Sheela'.
 - Format to write a character string repetition is: nC'x....x'

Where n can be from 1 to 4095; n repetitions of the character string constant (C'x...x' inserted into the reformatted input records. If n is omitted, 1 is used instead.

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD-EQUIP ACCREDITED 72

Instructor Notes:

7.5: SORT Utility



Inserting Constants (Contd...)

If you want to include a single apostrophe in the string, you must specify it as two single apostrophes. For example, O'NEILL must be specified as C'O''NEILL'.

- Hexadecimal Strings

- Format for writing a hexadecimal string is: X'yy.....yy' where yy is a pair of hexadecimal digits. For example X'C1C2 is equivalent to C'AB'.
 - Format for a hexadecimal string repetition is: nC'yy....yy'

Where n can be from 1 to 4095; n repetitions of the hexadecimal string constant

X'yy...yy') are inserted into the reformatted input records. If n is omitted, 1 is used instead.

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 73

Instructor Notes:

7.5: SORT Utility

Setting up the Report Format - Example

Following statement sets up the report as shown below:

```
OPTION COPY
```

```
OUTREC FIELDS=(11:C'THE EMPLOYEE NUMBER IS ',1,4,  
30:C'THE EMPLOYEE NAME IS ',5,20,4X,25,4)
```

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 74

Instructor Notes:

7.5: SORT Utility



Reformat Records Using INREC Statement

INREC statement has the same format as OUTREC

INREC FIELDS=(26,4,35,5)

➤ **SORT FIELDS=(1,4,4,CH,A)**

SUM FIELDS=(5,5,BI)

As INREC reformats records before they are sorted.

- SORT and SUM statements must refer to the reformatted records as they appear in the output data set.

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 75

Instructor Notes:

7.5: SORT Utility



Preventing Overflow Summing Values

Prevent overflow in some cases using INREC to pad summary fields with zeros.

However, you cannot use this method for negative fixed-point binary data.

- Padding with zeros rather than ones would change the sign.

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 76

Instructor Notes:

7.5: SORT Utility



Padding Summary fields

If summary fields overflow, you can pad each of them on the left with 4 bytes (binary fields must be 2, 4, or 8 bytes long).

```
INREC FIELDS=(26,4,4Z,35,5)
SORT FIELDS=(1,4,CH,A)
SUM   FIELDS=(5,10,BI)
```

You cannot use the OUTREC statement to prevent overflow, as it is processed after summarization.

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 77

Instructor Notes:

7.5: SORT Utility



Processing Order Of Control Statements

Subsequent flowchart shows the order in which control statements are processed.

- SUM is processed at the same time as SORT or MERGE.
- It is not used with COPY.

You can write statements in any order.

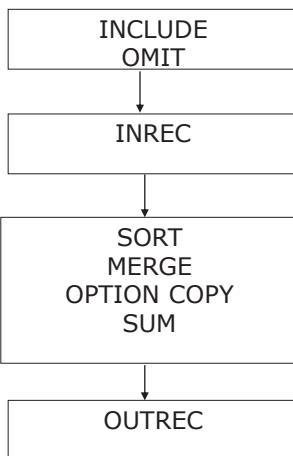
However, DFSORT always processes them in the order shown as follows.

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 78

Instructor Notes:

7.5: SORT Utility

Processing Order Of Control Statements - Flow Chart

© 2018 Capgemini. All rights reserved.

 OurUniversity
EFMD E-Learning ACCREDITED 79

Instructor Notes:

7.5: SORT Utility



SORT JCL 9 - Example

Sorts on salary removing duplicates, includes only salaries > 2000.

- To sort and pick up selective data, remove duplicates.
- SUM FIELDS=NONE is used to remove duplicates.
- It compares data in columns mentioned in SORT FIELDS= and removes second occurrence of matching data.

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD EQUIS ACCREDITED 80

Instructor Notes:

7.5: SORT Utility

SORT JCL 9 - Example (Contd...)

```
//DA0001TA      JOB LA2819,CG,NOTIFY=DA0001T,MSGCLASS=X  
//SRTSTEP      EXEC PGM=SORT  
//SYSIN        DD *  
          OPTION EQUALS  
          SORT FIELDS=(41,4,A),FORMAT=CSF  
          INCLUDE COND=(41,4,CSF,GT,2000)  
          SUM FIELDS=NONE  
/*  
//SORTIN       DD   DSN=DA0001T.INDATA3,DISP=SHR  
//SORTOUT      DD   DSN=DA0001T.SORTOUT6,DISP=(NEW,CATLG),  
//                      SPACE=(TRK,(3,3)), UNIT=SYSDA,  
//                      DCB=(BLKSIZE=800,LRECL=80,  
//                      RECFM=FB,DSORG=PS)
```

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 81

Instructor Notes:

7.5: SORT Utility

SORT JCL 9 - Example (Contd...)

```
//SORTWK01 DD      SPACE=(TRK,(10,5)),UNIT=SYSALLDA  
//SORTWK02 DD      SPACE=(TRK,(10,5)),UNIT=SYSALLDA  
//SORTWK03 DD      SPACE=(TRK,(10,5)),UNIT=SYSALLDA  
//SORTWK04 DD      SPACE=(TRK,(10,5)),UNIT=SYSALLDA  
//SYSOUT      DD      SYSOUT=*  
//
```

© 2018 Capgemini. All rights reserved.

 82

Instructor Notes:

7.5: SORT Utility JCL 9

Demo

SORT Utility (SORT JCL 9)



© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 83

Instructor Notes:

7.5: SORT Utility

SORT JCL 10 - Example

```
//DA0001TA JOB LA2719,CG,NOTIFY=DA0001T, MSGCLASS=X
//*Copies those records where either employee jobs begin with
'M' or 'S'
//SRTSTEP      EXEC   PGM=SORT
//SYSOUT       DD     SYSOUT=*
//SORTIN        DD     DSN=DA0021T.INDATA3,DISP=SHR
//SORTOUT       DD     DSN=DA0021T.SORTOUT7,
//                           DISP=(CATLG,DELETE),
//                           UNIT=SYSDA,
//                           SPACE=(TRK,(5, 2))
//
//SYSIN         DD   *
//          OPTION COPY
//          INCLUDE COND=(19,1,CH,EQ,C'M',OR,19,1,CH,EQ,C'S')
/*
//
```

© 2018 Capgemini. All rights reserved.

Instructor Notes:

7.5: SORT Utility JCL 10

Demo

SORT Utility (SORT JCL 10)



© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 85

Instructor Notes:

7.5: SORT Utility



SORT JCL 11 - Example

Change data throughout file.

- In this example, the 163rd character in the file is changed to C.
- Useful to change data in a file which is more than 255 characters in length as TSO edit option cannot be used for it.

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 86

Instructor Notes:

7.5: SORT Utility

SORT JCL 11 - Example (Contd...)

```
//DA0001TA    JOB LA2719,CG,NOTIFY=DA0001T, MSGCLASS=X  
//SRTSTEP      EXEC PGM=SORT  
///SYSOUT       DD SYSOUT=*  
//SORTIN        DD DSN=DA0001T.EMPLOYEE,DISP=SHR  
//SORTOUT       DD DSN=DA0001T.EMPLOYEE,DISP=SHR  
//SYSIN         DD *  
OPTION COPY  
OUTREC FIELDS =(1,162,C'C',164,137)  
/*  
//
```

© 2018 Capgemini. All rights reserved.

 OurUniversity
EFMD ELP ACCREDITED

87

Instructor Notes:

7.5: SORT Utility JCL 11

Demo

SORT Utility (SORT JCL 11)



© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 88

Instructor Notes:

7.5: SORT Utility



Reformatting Code

To reformat, code:

- Before sort fields, INREC FIELDS to reformat the record before sorting
- After sort fields, OUTREC FIELDS to reformat the record after sorting

Consider the following layout of a PS with fixed length records:

- Empno :starting at absolute byte 1, for 5 bytes long
- Empname :at absolute byte 6, for 25 bytes long
- Department :at absolute byte 31, for 15 bytes long
- Salary :at byte 46, for 5 bytes long
- The PS being populated with the following records:
 - 11111sujit admin 10000
 - 55555danny marketing 15000
 - 22222ajay admin 07000
 - 33333mala projects 20000^

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD EQUIS ACCREDITED 89

Instructor Notes:

7.5: SORT Utility

Reformatting Code (Contd...)



Before sorting, to include only employee number and salary field and to sort in descending order of salary field:

```
//SYSIN      DD      *
INREC FIELDS=(1,5,46,5)-----1111110000
      SORT FIELDS=(6,5,CH,D)           5555515000
      /*                                2222207000
                                     3333320000
```

■ The sorted dataset will contain:

3333320000
5555515000
1111110000
2222207000

© 2018 Capgemini. All rights reserved.

OurUniversity EPMO CLIP ACCREDITED 90

Instructor Notes:

7.5: SORT Utility



Reformatting Data-Bytes

To reformat data-bytes after sorting, OUTREC FIELDS is used.

- Also used to space out fields in the output.
- To obtain only the Empno and Salary fields in the output, but to sort on Empname field in ascending order:

```
//SYSIN      DD      *
```

```
SORT FIELDS=(6,25,CH,A) ----- 22222ajay.....  
OUTREC FIELDS=(1,5,46,5)           55555danny.....  
                                33333mala.....  
                                11111sujith.....
```

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD EQUACCREDITED 91

Instructor Notes:

7.5: SORT Utility



Reformatting Data-Bytes (Contd...)

To include character-literals, say, spaces between the Empno and Salary fields in the output dataset:

- Code OUTREC FIELDS=(1,5,2X,46,5)

Two spaces

The output dataset will contain: 22222 07000

**55555 15000
33333 20000
11111 10000**

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 92

Instructor Notes:

7.5: SORT Utility

SORT JCL 12 - Example

```
DA0001TA    JOB LA2719,CG, NOTIFY=DA0001T,MSGCLASS=X
/* Merges fields beginning with column 110 having length 5
/* INDATA1 and INDATA2 are sorted on the control field
//SRTSTEP      EXEC   PGM=SORT
//SORTIN01     DD DSN=DA0001T.INDATA1,DISP=OLD
//SORTIN02     DD DSN=DA0001T.INDATA2,DISP=OLD
//SORTOUT      DD DSN=DA0001T.SORTOUT4,DISP=(NEW,CATLG),
//                  SPACE=(TRK,(3,3,)), UNIT = SYSDA,
//                  DCB=(BLKSIZE=800, LRECL=80,
//                  RECFM=FB,DSORG=PS)
//SYSOUT        DD SYSOUT=*
//SYSIN DD  *
MERGE          FIELDS = (110,5,A),FORMAT=CH
/*
```

© 2018 Capgemini. All rights reserved.

 OurUniversity
EFMD ELP ACCREDITED 93

Instructor Notes:

7.5: SORT Utility JCL 12

Demo

SORT Utility (SORT JCL 12)



© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 94

Instructor Notes:

7.5: SORT Utility

Example

To select employees getting a salary of 1000 and sort in ascending order of employee number

```
//SYSIN      DD      *
INCLUDE COND=(46,5,CH,EQ,C'1000)
      SORT FIELDS=(1,3,CH,A)
      /*
```

Conversely, OMIT command can be used to exclude employees with salary of 1000.

INCLUDE and OMIT are mutually exclusive

Can use connect operators AND and OR to form several logical conditions

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 95

Instructor Notes:

Why ICETOOL ?



- ICETOOL is a multipurpose DFSORT utility that uses the capabilities of DFSORT to perform multiple operations on one or more data sets in a single step.
- It provides advance file processing features with easy to use control statements.

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 96

Instructor Notes:**JCL FOR ICETOOL**

```
//ICTLJOB1 JOB (0000000TS,W002),'ILEARN @ IGATE PATNI',MSGCLASS=H,  
//      TIME=(1,15),REGION=1M,MSGLEVEL=(1,1),NOTIFY=&SYSUID  
/*  
//ICTLSTEP EXEC PGM=ICETOOL  
//IN1   DD DSN=DSRP041.EMPLOYEE.DATA.INFILE,  
//      DISP=SHR  
//OUT1  DD DSN=DSRP041.EMPLOYEE.DATA.ICTL.OUTFILE,  
//      DISP=(NEW,CATLG,DELETE),  
//      SPACE=(TRK,(50,5),RLSE),  
//      DCB=(RECFM=FB,LRECL=80,BLKSIZE=8000)  
//BACKUP1 DD DSN=DSRP041.EMPLOYEE.DATA.ICTL.BACKUP1,  
//      DISP=(NEW,CATLG,DELETE),  
//      SPACE=(TRK,(50,5),RLSE),  
//      DCB=(RECFM=FB,LRECL=80,BLKSIZE=8000)  
//TOOLIN DD *  
COPY FROM(IN1) TO(OUT1,BACKUP1) USING(CTL1)  
/*  
//CTL1CNTL DD *  
SORT FIELDS=COPY  
/*  
//SYSOUT DD SYSOUT=*  
//TOOLMSG DD SYSOUT=*  
//DFSMMSG DD SYSOUT=*  
/*
```

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 97

Instructor Notes:

Comment & Blank



- Comment Statements
 - **Asterisk (*) in column 1 indicates a comment statement.**
 - **Comment statements are printed with other ICETOOL statements, but otherwise ignored.**
- Blank Statements
 - **Blank in columns 1-72 indicates a blank statement.**
 - **Blank statements are ignored since ICETOOL prints blank lines where appropriate.**

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 98

Instructor Notes:

ICETOOL RETURN CODES



- ICETOOL sets a return code for each operation it performs.
- For the step, ICETOOL sets the return code to the highest operator return code.
- The return codes are:
 - **0 - Successful completion. No errors were detected.**
 - **4 - Successful completion. DFSORT detected one or more warning conditions.**
 - **12 - Unsuccessful completion. ICETOOL detected one or more errors. Can also be set if the record count meets a specified criteria (for example, a data set is empty, or a data set contains more than 50000 records).**
 - **16 - Unsuccessful completion. DFSORT detected one or more errors.**
 - **20 - Message data set error. The TOOLMSG DD statement was not present or the TOOLMSG data set was not opened.**
 - **24 - Unsupported operating system.**

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD-EQUIP ACCREDITED 99

Instructor Notes:

1.0: [ICETOOL-COPY]

COPY

- It is used to copy one file to one or more file.
- Why ICETOOL COPY and not the COPY in DFSORT?

COPY FROM(IN1) TO(NEW,BACKUP) USING(CTL1)

© 2018 Capgemini. All rights reserved.

 OurUniversity 100
EFMD EQUACCREDITED**NOTE:**Copies up to 10 datasets**FROM - the ddname of the input data set.****Provide a dd name for DD Statement.****TO - the ddnames of 1 to 10 output data sets.****Provide ddnames for DD Statement.****TO, USING, or TO and USING must be specified.****USING - the first 4 characters of the ddname (xxxxCNTL) for the DFSORT control statement data set must supply a DD statement for xxxxCNTL if you specify USING(xxxx). TO, USING, or TO and USING must be specified.**

Instructor Notes:

1.1: [ICETOOL-COPY]
Demo

- Create 2 files using the Employee file. One file for Manager and other for Non-Managers



© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 101

```
//ICTLJOB1 JOB (00000000TS,W002),'ILEARN @ IGATE
PATNI',MSGCLASS=H,
//      TIME=(1,15),REGION=1M,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//*
//ICTLSTEP EXEC PGM=ICETOOL
//IN1    DD DSN=DSRP041.EMPLOYEE.DATA.INFILE,
//        DISP=SHR
//OUT1   DD DSN=DSRP041.EMPLOYEE.DATA.ICTL.OUTFILE,
//        DISP=(NEW,CATLG,DELETE),
//        SPACE=(TRK,(50,5),RLSE),
//        DCB=(RECFM=FB,LRECL=80,BLKSIZE=8000)
//BACKUP1 DD DSN=DSRP041.EMPLOYEE.DATA.ICTL.BACKUP1,
//        DISP=(NEW,CATLG,DELETE),
//        SPACE=(TRK,(50,5),RLSE),
//        DCB=(RECFM=FB,LRECL=80,BLKSIZE=8000)
//TOOLIN DD *
//COPY FROM(IN1) TO(OUT1,BACKUP1) USING(CTL1)
/*
//CTL1CNTL DD *
//        SORT FIELDS=COPY
/*
//SYSOUT  DD SYSOUT=*
//TOOLMSG DD SYSOUT=*
//DFSMMSG DD SYSOUT=*
//*
```

Instructor Notes:

2.0: [ICETOOL-SORT]
SORT



- It is used to sort data.
- Why ICETOOL SORT and not the SORT in DFSORT?

• SORT FROM(indd) USING(xxxx) TO(outdd,...)

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 102

Instructor Notes:

2.1: [ICETOOL-SORT]
Demo

- Sort based on Employee First Name, DOB and Salary using the Employee file.



© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD-EQUIS ACCREDITED 103

```
//ICTLJOB1 JOB (00000000TS,W002),'ILEARN @ IGATE
PATNI',MSGCLASS=H,
//      TIME=(1,15),REGION=1M,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//*
//ICTLSTEP EXEC PGM=ICETOOL
//IN1    DD DSN=DSRP041.EMPLOYEE.DATA.INFILE,
//      DISP=SHR
//OUT1   DD DSN=DSRP041.EMPLOYEE.DATA.ICTL.SORTED1,
//      DISP=(NEW,CATLG,DELETE),
//      SPACE=(TRK,(50,5),RLSE),
//      DCB=(RECFM=FB,LRECL=80,BLKSIZE=8000)
//TOOLIN DD *
      SORT FROM(IN1) TO(OUT1) USING(CTL1)
/*
//CTL1CNTL DD *
      SORT FIELDS=(7,9,CH,A,55,10,CH,A,65,05,CH,D)
/*
//SYSOUT  DD SYSOUT=*
//TOOLMSG DD SYSOUT=*
//DFSMMSG DD SYSOUT=*
//*
```

Instructor Notes:

3.0: [ICETOOL-COUNT]
COUNT



➤ It is used to print a message in TOOLMSG containing the count of records in the INDD data set..

•COUNT FROM(IN1) USING(CTL1)

© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 104

COUNT - prints a message containing the count of records in a data set.

Required Operand

FROM - the ddname of the input data set.

Provide a DD statement for the ddname you specify.

Optional Operands

USING - the first 4 characters of the ddname (xxxxCNTL) for the DFSORT control statement data set.

Provide a DD statement for xxxxCNTL if you specify USING(xxxx).

VSAMTYPE - the record format for a VSAM input data set (F or V).

LOCALE - overrides the installation default for locale processing.

EMPTY, NOTEMPTY, HIGHER, LOWER, EQUAL, NOTEQUAL - defines the criteria against which the record count is to be matched. If the criteria is met, ICETOOL sets RC=12 for this COUNT operator by default, or RC=4 if RC4 is specified. If the criteria is not met, ICETOOL sets RC=0 for this COUNT operator.

Instructor Notes:

3.1: [ICETOOL-COUNT]
Demo

- Count Employees with different designation using the Employee file



© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD-EQUIP ACCREDITED 105

```
//ICTLJOB1 JOB (00000000TS,W002),'ILEARN @ IGATE PATNI',MSGCLASS=H,  
//           TIME=(1,15),REGION=1M,MSGLEVEL=(1,1),NOTIFY=&SYSUID  
/*  
//ICTLSTEP EXEC PGM=ICETOOL  
//IN1   DD DSN=DSRP041.EMPLOYEE.DATA.INFILE,  
//           DISP=SHR  
//TOOLIN DD *  
COUNT FROM(IN1) USING(CTL1)  
COUNT FROM(IN1) USING(CTL2)  
COUNT FROM(IN1) USING(CTL3)  
/*  
//CTL1CNTL DD *  
INCLUDE COND=(44,8,CH,EQ,C'DESIGNER')  
/*  
//CTL2CNTL DD *  
INCLUDE COND=(44,8,CH,EQ,C'MANAGER')  
/*  
//CTL3CNTL DD *  
INCLUDE COND=(44,8,CH,EQ,C'FIELDREP')  
/*  
//SYSOUT  DD SYSOUT=*  
//TOOLMSG DD DSN=DSRP041.EMPLOYEE.DATA.ICTL.COUNT1,  
//           DISP=(NEW,CATLG,DELETE),  
//           SPACE=(TRK,(50,5),RLSE),  
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=8000)  
//*TOOLMSG DD SYSOUT=*  
//DFSMMSG DD SYSOUT=*  
/*
```

Instructor Notes:

7.5: Lab

Lab

Day 3 and Day 4 Labs



© 2018 Capgemini. All rights reserved.

OurUniversity
EFMD ELP ACCREDITED 106

Instructor Notes:**Summary**

IEFBR14 is commonly used to delete, allocate and uncatalog a dataset.

IEBGENER Utility is commonly used to copy, concatenate and to empty sequential datasets.

SORT Utility is used to sort data, copy selective data, remove duplicates, and change data throughout the file.



Instructor Notes:**Review Question**

Question 1: Which of the following utility is used to uncatalog a dataset?

- IEBGENER
- IEFBR14
- SORT

Question 2: Which of the following utility can be used to copy contents of a sequential dataset?

- IEFBR14
- IEBGENER
- SORT



JCL

Lab Book

Document Revision History

Date	Revision No.	Author	Summary of Changes
23-July-2009	4.0	Arjun Singh	Content creation
26-Oct-2009		CLS Team	Review
23 rd - May- 2011	5.0	Vaishali Kasture	Revamp and Content Creation
8 th -Aug-12	5.1	Rajita Dhumal	Revamped after Assignment Review

Table of Contents

Getting Started	1
Lab 1. Learning the various parameters of JCL statements.....	2
Lab 2. Executing COBOL programs with File Handling	16
<u>!Unexpected End of Formula</u>	
Lab 4. Utilities & Procedures	20
Lab 5. Extra Assignments on Procedures.....	21
Appendices	114
<i>Appendix A: COMPLINK JCL.....</i>	114
Operators Procedure to enquire the details of tape using CA1 22	
Xpeditor 31	
Map the IBM MVS Environment into Windows 95 or Windows NT 88	
Executing JCL Using CA-Realia II workbench 94	
<i>Appendix B: Table of Figures</i>	116
<i>Appendix C: Table of Examples</i>	119

Getting Started

Overview

This lab book is a guided tour for learning JCL. It comprises assignments.

Setup Checklist for JCL

Here is what is expected on your machine in order for the lab to work.

Minimum System Requirements

- Intel Pentium 90 or higher
- Microsoft Windows 95, 98, or NT 4.0, 2k, XP.
- Memory: 32MB of RAM (64MB or more recommended)
- Internet Explorer 6.0 or higher

Please ensure that the following is done:

- PASSPORT PC-TO-HOST (mainframe terminal simulator) is installed
- Connectivity to the Mainframe

Instructions

- You need three PDS, one each for storing COBOL source code, JCL code and loadlib. The convention followed for naming the PDS is as follows:
USRID.NAME.X where:
USRID is IBM login id used by the participant,
NAME is name of the participant and
X is either COBOL/SRCLIB or JCL or LINKLIB.
- For example: The participant using IBM login id DA0001T and name as SHEELA will have three PDS as follows:

as	DA0001T.ARJUN.COBO	– for storing COBOL source programs PDS members
	DA0001T.ARJUN.JCL	– for storing the JCL's as PDS members
	DA0001T.ARJUN.LINKLIB	– for storing load modules of programs as PDS members

If these PDSs do not exist, then these are to be created. Kindly refer to the MVS lab book for the same.

Learning More (Bibliography)

- Expert MVS/XA JCL by Mani Carathanassis
- MVS/JCL by Doug Lowe

Lab 1. Learning the various parameters of JCL statements

Goals	• Learning the various parameters of JCL statements
Time	120 minutes

Problem 1:

1. Allocate a data set to contain your JCL.
2. Edit the JCL data set and add the necessary JCL. Use ISPF to edit the data set that you just allocated.
3. Enter the following JCL statements into the data set after making the necessary changes as per the suggestions given below:

```
//JOB1 JOB 'accounting_data',
// 'user_name',
// NOTIFY=,
// MSGCLASS=message_class,
// MSGLEVEL=(1,1),
// CLASS=n,
//COBRUN EXEC PROG=your own Cobol program name
//STEPLIB DD DSN=your own dataset name,DISP=SHR
//SYSPRINT DD SYSOUT=*
```

In the JCL code above:

- a. Replace accounting_data with the appropriate identification information. Account information should be with the batch name i.e. MF123.
- b. Replace Job name with your userid.
- c. Replace user_name with your name.
- d. Job notification should be given to you
- e. Replace message_class with the appropriate message class value.
- f. Tell the system to reproduce this JCL code in the output, and to include Allocation/termination messages if the job terminates abnormally/normally

Problem 2:

Refer to shared file named 'Sysin Prg' for Cobol and Run JCL coding

- The participants have to analyze the program, remove the error and successfully execute the program.
- After completion of the assignment, the Program/JCL i.e. Run JCL should successfully execute.
- All the participants have to document the error in the excel sheet along with steps taken to resolve the error.

Problem 3:

Refer to shared file named 'if_else Prg' for Cobol and JCL coding

- Do analyze the given programs. Participants have to modify JCL in such a ways that after submitting the job, 2nd program i.e. named 'SYSPRG' should get executed though prior program i.e. 'CHAP1' do have one alignment error.
- All the participants have to document the error in the excel sheet along with steps taken to implement the change request.

Problem 4:

Implement a Referrback on DD Statement .Make your own assumptions.

Problem 5:

Refer to shared file named 'File_Prg' for Cobol and JCL coding. This shared Cobol code will copy data from source dataset to target dataset.

- The participants have to analyze the program and successfully execute the program.
- Specify a DD statement that creates the new file in your RUN JCL.

Problem 6:

Execute a Simple COBOL program (For ex. CHAP1 from shared folder) and specify the JOBCAT and STEPCAT commands for the same

Problem 7:

Create 2 physical sequential files of LRECL as 80, BLKSIZ as 800
(Userid.MPST.JCL01.MAST.IN.PS and Userid.MPST.JCL02.MAST.IN.PS)

- Refer to shared files MPST-JCL01-IN01-PS.Dat and MPST-JCL01-IN02-PS.Dat for input records and copy the records into the dataset created in the previous step.
- Perform Concatenation of both the PS
- Create 2 PDS Userid.PDS1 and Userid.PDS2 and implement concatenation of 2 PDS.

Problem 8:

Refer to shared file Instream_Symbolic_Prg' for Cobol and Run JCL coding

- The participants have to analyze the program, remove the error and successfully execute the program.
- After completion of the assignment, the Program/JCL i.e. Run JCL should successfully execute.
- All the participants have to document the error in the excel sheet along with steps taken to resolve the error

Lab 2. Utilities & Procedures

Goals	<ul style="list-style-type: none"> • Understand how to use utilities and write procedures.
Time	180 Minutes

2.1: Assignments on Utilities and Procedures:

1. Write a JCL to create a sequential data set using **IEFBR14**.
2. Write a JCL to delete a sequential data set using **IEFBR14**.
3. Write a JCL for **IEBGENER** to copy from **EMPFILE** into another. Try out all the JCLs given in the handout.
4. Try concatenation of sequential files using **IEBGENER**.
5. Write a JCL to sort the **EMPFILE** in descending or ascending order of **EMP-NO**.
6. Try all the parameters for Sort, namely, INREC, OUTREC, SUM fields.
7. Create a PS Dept which includes (Deptno,DeptName,Loc) & list all the distinct Departments in Ascending order from the Dept File
8. Write a cataloged procedure for invoking either **IEFBR14** or **IEBGENER** or **SORT** utility.
9. Write an **instream** procedure for the assignment 7.
10. Using the ITEM-FILE as Sample Input, write the solution for following assignments.

ITEM-FILE

Item Code	Vendor	Status	Quantity
E001	780005	Y	10
E003	780005	N	0
E002	780006	Y	8
E003	780005	N	0
E005	780004	Y	2
E004	780005	Y	4
E006	780006	N	0
E005	780004	Y	2

E007	780006	N	0
E006	780006	N	0
E008	780008	Y	14

Assignment-1: Write a JCL to create an output file having unique records (with respect to Item Code column) using the **SORT** utility.

Assignment-2: Write a JCL to extract records having Item-code 'E003' in new file.

Assignment-3: Write a JCL to create a file having only two columns (using OUTREC) using SORT utility.
[Create Vendor-Item-File having only two fields say Item-Code and Vendor].

Assignment-4: Taking above **Item-File** as input, write a JCL to create another file (ITEM-FILE-SEQ) having same data and sequence-number as additional column using **SORT** utility.

Assignment-5: Taking above **Item-File** as input, write a JCL to create another file (ITEM-FILE-REV) having same records in reverse order of sequence number using **SORT** utility. (Use **INREC** and **OUTREC**)

Assignment-6: Considering the file created in Lab 5, that is ITEM-FILE, as input, write a JCL to concatenate these two files into a single file (ITEM-CONCT_FILE) lengths using **IEBGENER**.

Assignment-7: Taking above **Item-File** as input, write a **Job** to skip the first 3 records and copying next 4 subsequent records into a new dataset (ITEM-FILE-4R) using **SORT** utility.

11. Write a JCL that will create new PDS USerid.MyPDS and will be cataloged on successful execution of the step and dataset will be deleted if job terminates abnormally.

Lab 3. Analysis, Enhancement and Debugging Assignments based on Utilities & Procedures

Goals	• Solve Analysis, Enhancement and Debugging Assignments.
Time	180 Minutes

Analysis/Enhance Assignment

Problem 1: MPST0PG01

MPST01-Concatenate, Duplicate, Sort, Report

The participants have to analyze the program/JCL, remove the error and successfully execute the program.

1. Create partitioned dataset Userid.MPST.JCL01.PDS for writing the Source code\JCL.
2. Create 2 physical sequential files of LRECL as 80, BLKSIZ as 800(Userid.MPST.JCL01.IN01.PS and Userid.MPST.JCLxx.IN02.PS)
3. Refer to MPST-JCL01-IN01-PS.Dat and MPST-JCL01-IN01-PS.dat for input records and copy the records of these two .dat files into the dataset created in the previous step.
4. Create a member (JCL01REP) into Userid.MPST.JCLxx.PDS and upload the program into it from JCL01REP.CBL
5. Create a member (CMPLNK01) into Userid.MPST.JCLxx.PDS and upload the program into it from CMPLNK01.JCL
6. Create a member (MSTJCL01) into Userid.MPST.JCLxx.PDS and upload the program into it from MSTJCL01.JCL

Member Name	Description
JCL01REP	Cobol program for Generating report from Userid.MPST.JCLxx.OUT03S.PS
CMPLNK01	Comp link for Cobol program JCL01REP
MSTJCL01	Master JCL for execution of all steps

After completion of the assignment, the jcl (MSTJCL01) should successfully execute. All the participants have to document the error in the excel sheet along with steps taken to resolve the error.

Stepwise Functionality of MSTJCL01:

Step 1:

The records inside the dataset IN01.PS & IN02.PS have to be concatenated inside physical sequential dataset (Userid.MPST.JCLxx.OUT01.PS)

Input dataset: Userid.MPST.JCLxx.IN01.PS (Existing)
Userid.MPST.JCLxx.IN02.PS (Existing)
Output dataset: Userid.MPST.JCLxx.OUT01.PS (New)

Step 2:

The step remove duplicate records from Userid.MPST.JCLxx.OUT01.PS and rest of the records are copied into the physical sequential dataset (Userid.MPST.JCLxx.OUT02S.PS)

Input dataset: Userid.MPST.JCLxx.OUT01.PS
Output dataset: Userid.MPST.JCLxx.OUT02S.PS (New)

Step3:

This step sort the records of the Userid.MPST.JCLxx.OUT02S.PS on the field ITEM-GROUP and the sorted records are stored in physical sequential dataset (Userid.MPST.JCLxx.OUT03S.PS)

Input dataset: Userid.MPST.JCLxx.OUT02S.PS
Output dataset: Userid.MPST.JCLxx.OUT03S.PS (New)

Step4:

The steps generate a Group wise report into the dataset Userid.MPST.JCLxx.REP.PS showing details of item that will be supplied by the vendor

Input dataset: Userid.MPST.JCLxx.OUT03S.PS
Output dataset: Userid.MPST01.JCLxx.REP.PS (New)

Dataset created after execution of MSTJCL01:

Userid.MPST.JCLxx.OUT01.PS
Userid.MPST.JCLxx.OUT02S.PS
Userid.MPST.JCLxx.OUT03S.PS
Userid.MPST01.JCLxx.REP.PS

Record Layout of files:

DSRB048.MPST.JCL01.IN01.PS, DSRB048.MPST.JCL01.IN02.PS

Column Name	PIC Desc
Item-Code	X(4)
Vendor-Code	X(4)

Quantity	X(2)
Group-Name	X(10)
Order-Date	X(10)

Note: Between two fields have a space

Problem 2: MPST0PG02

MPST02-Extraction of input records, Empty, Sort, Report, Abend

The participants have to analyze the program, remove the error and successfully execute the program.

1. Create partitioned dataset Userid.MPST.JCL02.PDS for writing the Source code\JCL.
2. Create physical sequential files of LRECL as 80, BLKSIZ as 800(Userid.MPST.JCL02.MAST.IN.PS)
3. Refer to MPST-JCL02-MAST-IN-PS.Dat for input records and copy the records into the dataset created in the previous step.
4. Create a member (GRITMAST) into Userid.MPST.JCL02.PDS and upload the program into it from GRITMAST.CBL
5. Create a member (CMPLNK01) into Userid.MPST.JCL02.PDS and upload the program into it from CMPLNK01.JCL
6. Create a member (MST02REP) into Userid.MPST.JCL02.PDS and upload the program into it from MST02REP.CBL
7. Create a member (CMPLNK02) into Userid.MPST.JCL02.PDS and upload the program into it from CMPLNK02.JCL
8. Create a member (MST02AB) into Userid.MPST.JCL02.PDS and upload the program into it from MST02AB.CBL
9. Create a member (CMPLNK03) into Userid.MPST.JCL02.PDS and upload the program into it from CMPLNK03.JCL.
10. Create a member (MSTJCL02) into Userid.MPST.JCL02.PDS and upload the program into it from MSTJCL02.JCL

After completion of the assignment, the jcl (MSTJCL01) should successfully execute. All the participants have to document the error in the excel sheet along with steps taken to resolve the error.

Member Name	Description
GRITMAST	Cobol program for extraction of GRIT records from DSRB048.MAIN.PS.
MST02REP	Cobol program for generating report BU wise
MST02AB	Cobol program for displaying Abends Error in JCL
CMPLNK01	Comp link for Cobol program GRITMAST
CMPLNK02	Comp link for Cobol program MST02REP

CMPLNK03	Comp link for Cobol program MST02AB
MSTJCL02	Master JCL for execution of all steps

Stepwise Functionality of MSTJCL02:

Step 1:

To extract records of a particular BU into Userid.MPST.JCL02.BUOUT.PS from the master file (Userid.MPST.JCL02.MAST.IN.PS)

Input dataset: Userid.MPST.JCL02.MAST.IN.PS (Existing)
Output dataset: Userid.MPST.JCL02.BUOUT.PS (New)

Step 2:

To sort the records of Userid.MPST.JCL02.BUOUT.PS on Empid and copy the sorted records into Userid.MPST.JCL02.BU.PS

Input file: Userid.MPST.JCL02.BUOUT.PS
Output file: Userid.MPST.JCL02.BU.PS (New)

Step 3:

This Step generates BU wise report in sequence of employee number.

Input file: Userid.MPST.JCL02. BU.PS
Output file: Userid.MPST.JCL02. BUREP.PS (New)

Step 4:

This Step executes only when if any previous step has been abended.

Dataset created after execution of MSTJCL02:
DSRB048.MPST.JCL02.BU.PS

DSRB048.MPST.JCL02.BUOUT.PS
DSRB048.MPST.JCL02.BUREP.PS

Record Layout of files:

Userid.MPST.JCL02.MASTER.IN.PS

Column Name	PIC Desc
EMP-NO	9(4)
EMP-FNAME	X(12)
EMP-LNAME	X(10)
EMP-BU	X(8)
EMP-SBU	X(8)

EMP-DESIG	X(15)
BASIC	9(5)V99
HRA	9(5)V99
CHLD-ALLOW	9(5)V99
EXTRA-ALLOW	9(5)V99

Note: Between two fields have a space

Problem 3: MPST0PG03

MPST03- File& Table, Search & output, Sort, Report

The participants have to analyze the program, remove the error and successfully execute the program.

1. Create partitioned dataset Userid.MPST.JCL03.PDS for writing the Source code\JCL.
2. Create 2 physical sequential files of LRECL as 80, BLKSIZ as 800(Userid.MPST.JCL03.IN.PRMTR.PS and
3. Userid.MPST.JCL03.IN.ITEM.PS)
4. Refer to MPST- JCL03-IN-PRMTR-PS.Dat and MPST-JCL03-IN-ITEM-PS.dat for input records and copy the records of these two .dat files into the dataset created in the previous step.
5. Create a member (JCL03TRN) into Userid.MPST.JCL03.PDS and upload the program into it from JCL03TRN.CBL
6. Create a member (CMPLNK01) into Userid.MPST.JCL03.PDS and upload the program into it from CMPLNK01.JCL
7. Create a member (JCL03REP) into Userid.MPST.JCL02.PDS and upload the program into it from JCL03REP.CBL
8. Create a member (CMPLNK02) into Userid.MPST.JCL03.PDS and upload the program into it from CMPLNK02.JCL
9. Create a member (MSTJCL03) into Userid.MPST.JCL03.PDS and upload the program/JCL into it from MSTJCL03.JCL

After completion of the assignment, the Program/JCL i.e. MSTJCL03 should successfully execute.

All the participants have to document the error in the excel sheet along with steps taken to resolve the error.

Member Name	Description
JCL03TRN	Cobol program for fetching the data from DSRB048.MPST.JCL03.IN.PRMTR.PS And DSRB048.MPST.JCL03.IN.ITEM.PS And creation of DSRB048.MPST.JCL03.ITMGRPO.OUT.PS

JCL03REP	Cobol program for Report Generation
CMPLNK01	Comp link for Cobol program JCL03TRN
CMPLNK02	Comp link for Cobol program JCL03REP
MSTJCL03	Master JCL for execution of all steps

Stepwise Functionality of MSTJCL03:

Step 1:

This step Executes program(JCL03TRN) using Userid.MPST.JCL03.IN.ITEM.PS and Userid.JCL03.IN.PRMTR.PS as input files and write output records into Userid..MPST.JCL03.ITMGRPO.OUT.PS

Input file: Userid.MPST.JCL03.IN.ITEM.PS (Existing)
Userid.MPST.JCL03.IN.PRMTR.PS (Existing)
Output File: Userid.MPST.JCL03.ITMGRPO.OUT.PS (New)

Step 2:

Sorting records of Userid.MPST.JCL03.ITMGRPO.OUT.PS on group-name and item-code and placing the sorted records into Userid.MPST.JCL03.ITMGRPS.PS

Input File: Userid.MPST.JCL03.ITMGRPO.OUT.PS
Output File: Userid.MPST.JCL03.ITMGRPS.PS (New)

Step 3:

This step Executes program JCL03REP and creates vendor-wise report into dataset Userid.MPST.JCL03.REP.PS

Input File: Userid.MPST.JCL03.ITMGRPS.PS
Output File: Userid.MPST.JCL03.REP.PS (New)

Dataset created after the execution of MSTJCL03

Userid.MPST.JCL03.ITMGRPO.OUT.PS
Userid.MPST.JCL03.ITMGRPS.PS
Userid.MPST.JCL03.REP.PS

Record Layout of files:

Userid.MPST.JCL03.IN.PRMTR.PS

Column Name	PIC Desc
ITEM-NO	X(04)
GROUP-NO	X(03)
GROUP-NAME	X(10)

Userid..MPST.JCL03.IN.ITEM.PS

Column Name	PIC Desc
ITEM-CODE	X(04)
VENDOR-NO	X(03)
ITEM-QTY	9(03)
ITEM-RATE	9(04)V99

DSRB048.MPST.JCL03.ITMGRPO.OUT.PS

Column Name	PIC Desc
OUT-ITEM-NO	X(04)
OUT-GROUP-NO	X(03)
OUT-GROUP-NAME	X(10)
OUT-ITEM-RATE	9(4)
OUT-ITEM-QTY	9(03)

Note: Between two fields have a space

Problem 4:

To Do

1. Write a JCL which will implement reformatting while copying datasets using IEBGENER utility
2. Write a JCL which will implement relocking of records.
3. Write a JCL using IEBCOPY which will copy to datasets but exclude a single PS.(Make your own assumptions).(Hint implement Exclude with IEBCOPY)
4. Compress any dataset using IEBCOPY
5. Write a JCL to Merge two unsorted datasets into one sorted dataset.
6. Using Set statement assign Symbolic Parameter to Unit and Volume parameter
7. Define in a PDS SYSPrint and SYSOUT statements and then using the INCLUDE statement include them in RUN JCL.

Lab 4. Assignments on Procedures

Goals	<ul style="list-style-type: none"> To understand how to write procedures.
Time	60 minutes

Sample input for **ITEM-FILE-TRANS1**

Item Code	Vendor	Qty-Ordered	Item-Group
E001	780005	12	Infra
E003	780005	10	Infra
E002	780006	10	Hardware
E003	780005	10	Infra
E005	780004	19	Admin

Sample input for **ITEM-FILE-TRANS2**

E004	780005	23	Infra
E006	780006	34	Hardware
E005	780004	19	Admin
E007	780006	80	Hardware
E006	780006	37	Hardware
E008	780008	60	Admin

Procedures:

Specification for Procedure **CUSTOM**:

Step1: Copy all records from ITEM-FILE-TRANS1 to ITEM-FILE-TRANS3.

Step2: Append records from ITEM-FILE-TRANS2 into the file which was created by step1, that is ITEM-FILE-TRANS3.

Step3: Sort records vendor-wise.

Step4: Process both dataset ITEM-FILE-TRANS1 and ITEM-FILE-TRANS2 data by a Program REPORT (program 'report' a simple single level control break report).

Solution:

Step 1: Write a catalog procedure **CUSTOM** to perform the above tasks as described.

Step 2: Write a JCL to invoke the catalog procedure **CUSTOM**. However, program **REPORT** should receive records only from **ITEM-FILE-TRANS1** as input.

Step 3: Write an in-stream procedure **IN-CUSTOM** to perform the task as described in Lab1. (Use ITEM-FILE-TRANS30 instead of ITEM-FILE-TRANS3)

Step 4: Write a JCL to override **DSN** of step1 in an in-stream procedure **IN-CUSTOM** to **ITEM-FILE-TRANS3** by symbolic and regular override together.

Appendices

Executing the COBOL program

Solution:

Steps:

Step 1: Key in the program.

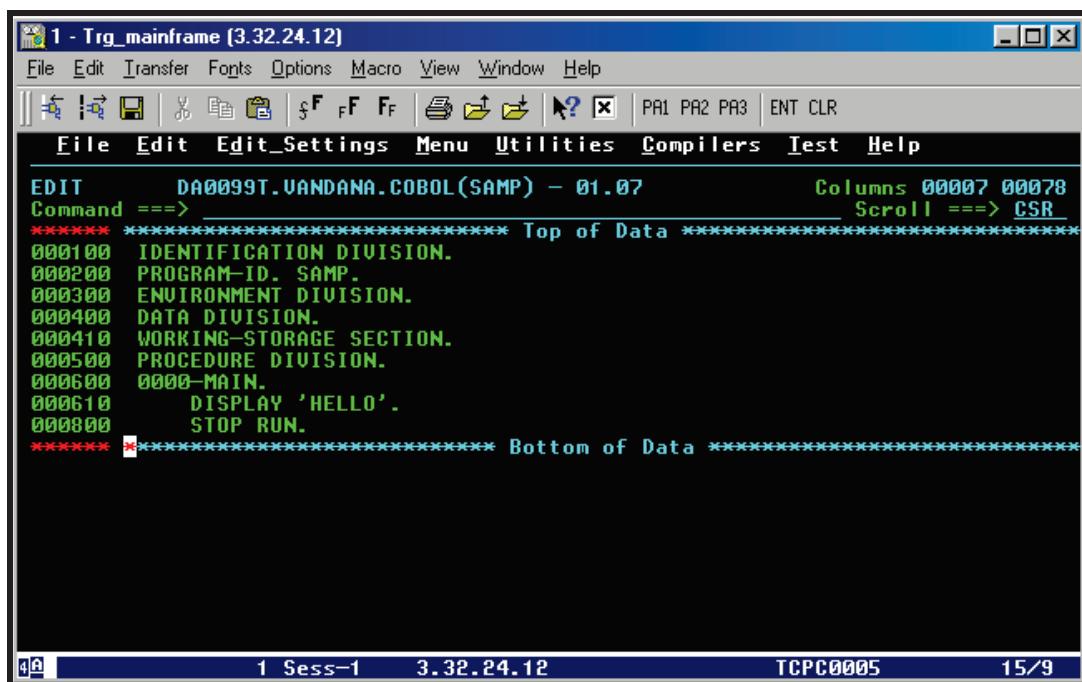
Step 2: Modify and submit the **Complink JCL**.

Step 3: Modify and submit the **Run JCL**.

Step 4: Check the output.

Example 1: Program to display 'Hello'

The following program is a simple COBOL program, which displays 'Hello' in the SYSOUT.



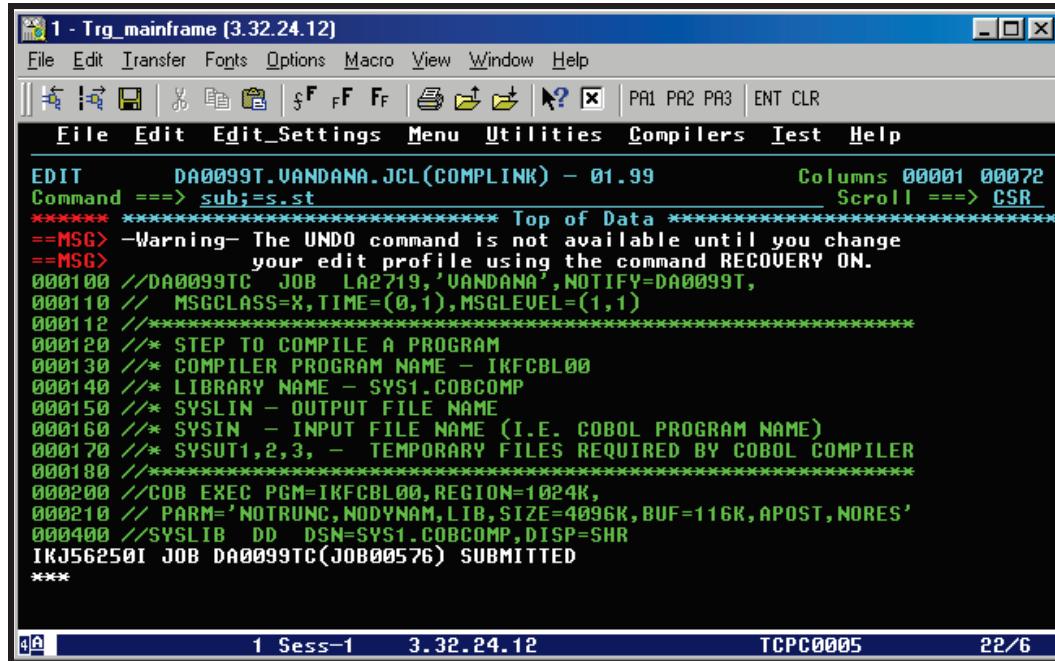
A screenshot of a terminal window titled "1 - Trg_mainframe [3.32.24.12]". The window contains the following COBOL source code:

```
DA0099T.VANDANA.COBOL(SAMP) - 01.07
*****
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. SAMP.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000410 WORKING-STORAGE SECTION.
000500 PROCEDURE DIVISION.
000600 0000-MAIN.
000610      DISPLAY 'HELLO'.
000800      STOP RUN.
*****
1 Sess-1 3.32.24.12 TCPC0005 15/9
```

Figure 1: Sample COBOL source code

Use the complink JCL to compile and link a COBOL program as shown below.

Type **SUB;=S.ST** on the command line as shown below, and then press **ENTER**.



```

1 - Trg_mainframe [3.32.24.12]
File Edit Transfer Fonts Options Macro View Window Help
File Edit Edit_Settings Menu Utilities Compilers Test Help
EDIT DA0099T.VANDANA.JCL(COMLINK) - 01.99 Columns 00001 00072
Command ==> sub;=s.st Scroll ==> CSR
***** ***** Top of Data *****
==MSG> -Warning- The UNDO command is not available until you change
==MSG> your edit profile using the command RECOVERY ON.
000100 //DA0099TC JOB LA2719,'VANDANA',NOTIFY=DA0099T,
000110 // MSGCLASS=X,TIME=(0,1),MSGLEVEL=(1,1)
000112 //*****
000120 // STEP TO COMPILE A PROGRAM
000130 /* COMPILER PROGRAM NAME - IKFCBL00
000140 /* LIBRARY NAME - SYS1.COBCOMP
000150 /* SYSLIN - OUTPUT FILE NAME
000160 /* SYSIN - INPUT FILE NAME (I.E. COBOL PROGRAM NAME)
000170 /* SYSUT1,2,3, - TEMPORARY FILES REQUIRED BY COBOL COMPILER
000180 //*****
000200 //COB EXEC PGM=IKFCBL00,REGION=1024K,
000210 // PARM='NOTRUNC,NODYNAM,LIB,SIZE=4096K,BUF=116K,APOST,NORES'
000400 //SYSLIB DD DSN=SYS1.COBCOMP DISP=SHR
IKJ56250I JOB DA0099TC(JOB00576) SUBMITTED
***
```

48 1 Sess-1 3.32.24.12 TCP0005 22/6

Figure 4: Submission of COMLINK JCL

The following panel is displayed which indicates the status of your job successful (zero return code, that is 0) or unsuccessful (non zero return code, that is 12). Then again press **ENTER**.

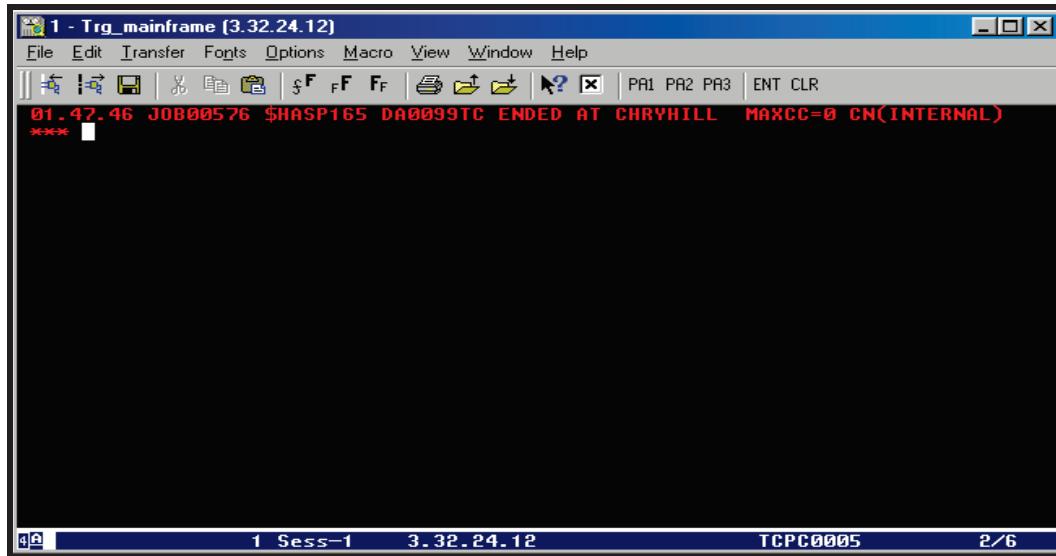


Figure 5: Notification message of COMPLINK JCL

After pressing **ENTER**, you will get the following panel.

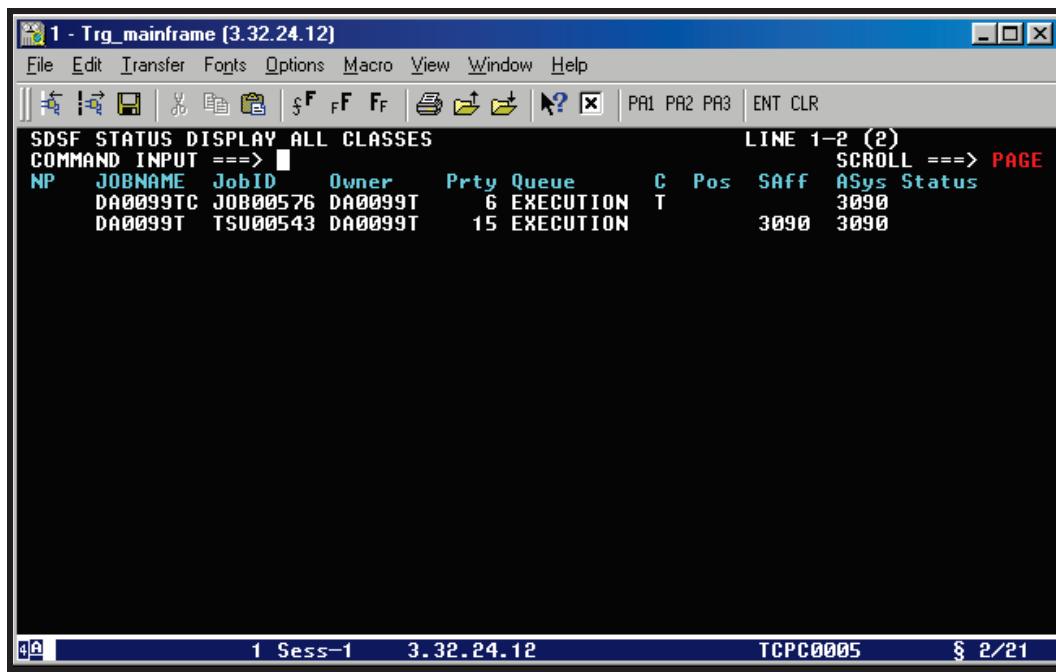
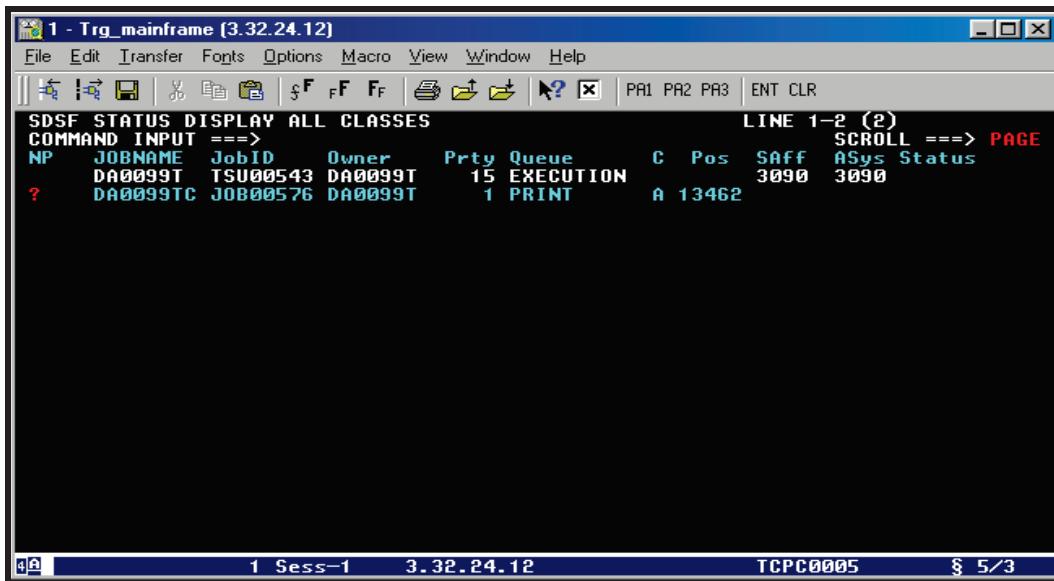


Figure 6: Execution status of COMPLINK JCL

You keep pressing **ENTER** till you get the following panel, then press **TAB** till the cursor is positioned as shown in the following figure and type **?**. Then press **ENTER**.



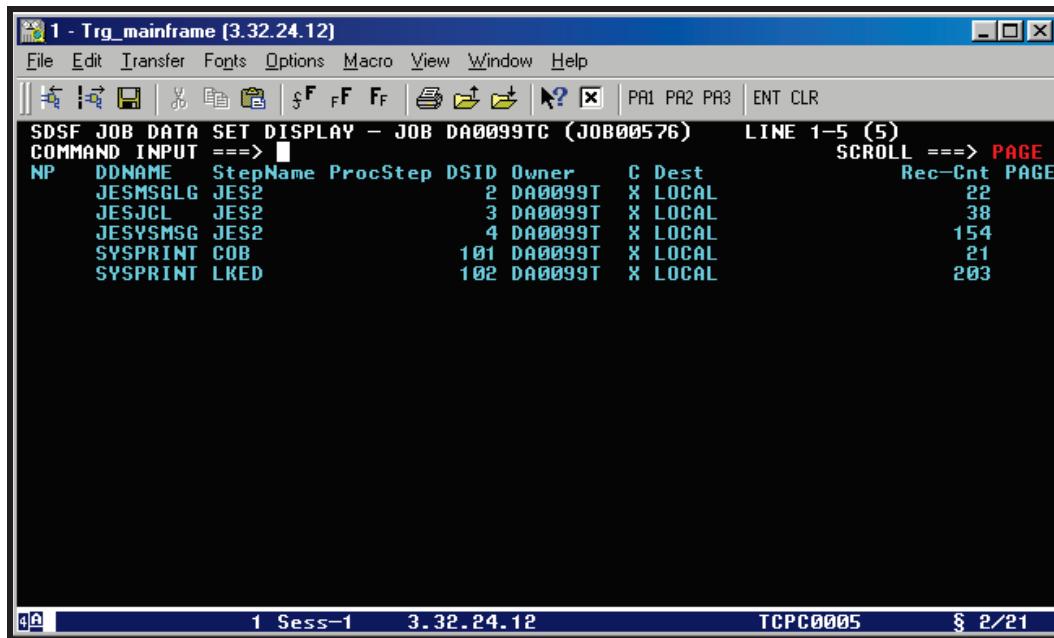
The screenshot shows a mainframe terminal window titled "1 - Trg_mainframe (3.32.24.12)". The window has a menu bar with File, Edit, Transfer, Fonts, Options, Macro, View, Window, Help. Below the menu is a toolbar with various icons. The main area displays a command-line interface for SDSF. The command entered is "SDSF STATUS DISPLAY ALL CLASSES". The output shows job details:

NP	JOBNAME	JobID	Owner	Prtv	Queue	C	Pos	SAFF	ASys	Status
	DA0099T	TSU00543	DA0099T		15 EXECUTION	C		3090	3090	
?	DA0099TC	JOB00576	DA0099T		1 PRINT	A	13462			

At the bottom of the terminal window, there is status information: "1 Sess-1 3.32.24.12 TCPC0005 § 5/3".

Figure 7: Execution status of COMPLINK JCL (Cont...)

You get the next panel, which indicates that compilation, and linking is successful.



The screenshot shows a terminal window titled "1 - Trg_mainframe (3.32.24.12)". The window contains a table of file creation details:

NP	DDNAME	StepName	ProcStep	DSID	Owner	C	Dest	SCROLL	Rec-Cnt	PAGE
	JESMSGLG	JES2		2	DA0099T	X	LOCAL			22
	JESJCL	JES2		3	DA0099T	X	LOCAL			38
	JESYSMSC	JES2		4	DA0099T	X	LOCAL			154
	SYSPRINT	COB		101	DA0099T	X	LOCAL			21
	SYSPRINT	LKED		102	DA0099T	X	LOCAL			203

At the bottom of the window, status information is displayed: "1 Sess-1 3.32.24.12 TCPC0005 § 2/21".

Figure 8: List of Files created after COMPLINK process

Alternately, you can type **S** on the line command and press **ENTER**.



The screenshot shows a terminal window titled "1 - TRG_MAINFRAME (3.32.24.12)". The menu bar includes File, Edit, Transfer, Fonts, Options, Macro, View, Window, Help. The toolbar contains various icons. The main display area shows the command "SDSF STATUS DISPLAY ALL CLASSES" followed by a table of job statistics. The table has columns: NP, JOBNAME, JobID, Owner, Prty, Queue, C, Pos, SAFF, ASys, Status. One row is highlighted in yellow: "DA0099T TSU10400 DA0099T 15 EXECUTION C 3090 3090". The bottom status bar shows "1 Sess-1 3.32.24.12 TCPC0044 § 5/3".

SDSF STATUS DISPLAY ALL CLASSES								DATA SET DISPLAYED		
COMMAND INPUT ==>								SCROLL ==> PAGE		
NP	JOBNAME	JobID	Owner	Prty	Queue	C	Pos	SAFF	ASys	Status
DA0099T	TSU10400	DA0099T		15	EXECUTION	C	3090	3090		
S	DA0099TC	JOB10713	DA0099T		1	PRINT	A	18358		

Figure 9: Panel to see the Statistics of the Job Submitted

You get the next panel, which indicates that compilation and linking is successful.

1 - TRG_MAINFRAME (3.32.24.12)

File Edit Transfer Fonts Options Macro View Window Help

SDSF OUTPUT DISPLAY DA0099TC JOB10713 DSID 2 LINE 0 COLUMNS 02- 81
COMMAND INPUT ==> ■ SCROLL --> PAGE

***** TOP OF DATA *****

J E S 2 J O B L O G — S Y S T E M 3 0 9 0 — N O D E

```

07.10.29 JOB10713 $HASP373 DA0099TC STARTED - INIT 31 - CLASS A - SYS 3090
07.10.29 JOB10713 ACF9CCCD USERID DA0099T IS ASSIGNED TO THIS JOB - DA0099TC
07.10.29 JOB10713 IEF403I DA0099TC - STARTED - TIME=07.10.29
07.10.29 JOB10713 OPS1181H INIT OPSS (*Local*) MVS N/A ATMRULES.TRJGAAAF E
07.10.29 JOB10713 E DA0099TC,SRUCLASS=BATPRDL
07.10.29 JOB10713 -
07.10.29 JOB10713 -JOBNAME STEPNAM PROCSTEP RC EXCP CPU SRB CLOCK
07.10.29 JOB10713 -DA0099TC COB 00 190 .00 .00 .0
07.10.30 JOB10713 -DA0099TC LKED 00 171 .00 .00 .0
07.10.30 JOB10713 IEF404I DA0099TC - ENDED - TIME=07.10.30
07.10.30 JOB10713 -DA0099TC ENDED. NAME=VANDANA TOTAL CPU TIME=
07.10.30 JOB10713 $HASP395 DA0099TC ENDED
```

—TIMINGS (MINS.)—

—JOB STATISTICS—

23 MAY 2003 JOB EXECUTION DATE

38 CARDS READ

438 SYSPUT PRINT RECORDS

0 SYSPUT PUNCH RECORDS

1 SYSPUT PAGE RECORDS

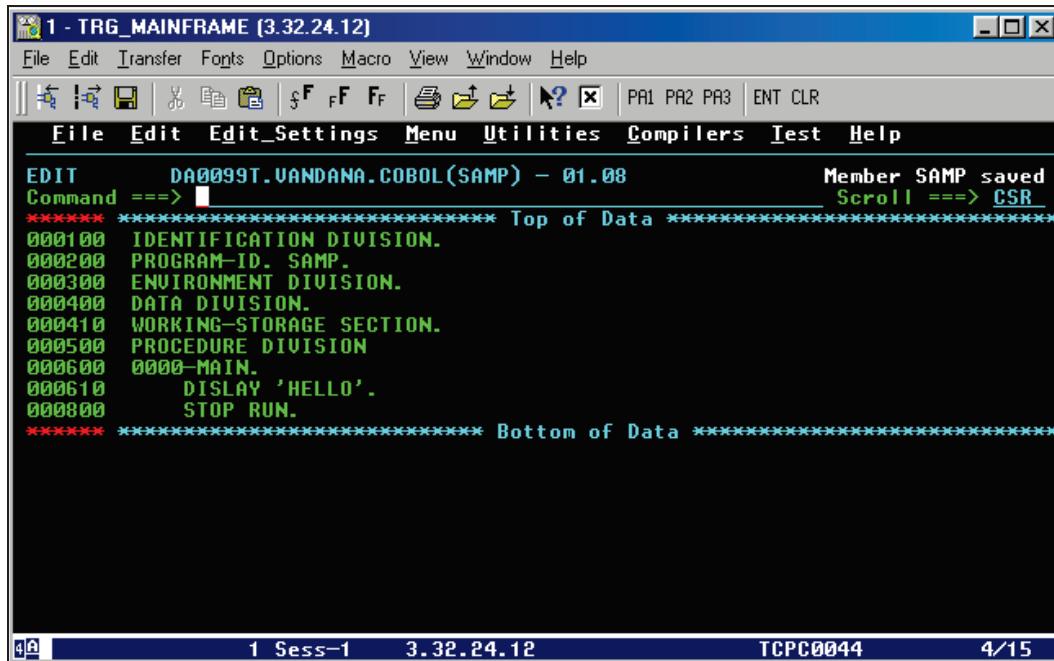
29 SYSPUT SPOOL KBYTES

48 1 Sess-1 3.32.24.12 TCPC0044 § 2/21

Figure 10: Statistics of COMPLINK jcl

If there is any compilation error, then **LKED** step is not shown. In that case, go to the **COB** step by pressing the **TAB** key. You will get the next panel that shows the COBOL source listing. On the command line, type **BOTTOM**, where you see all the compilation error along with the line numbers. Correct all the COBOL errors and repeat the above steps until you get the return code zero.

Following is a program with some errors, wherein DISPLAY is mentioned instead of DISPLAY and period is missing in the PROCEDURE DIVISION.



```

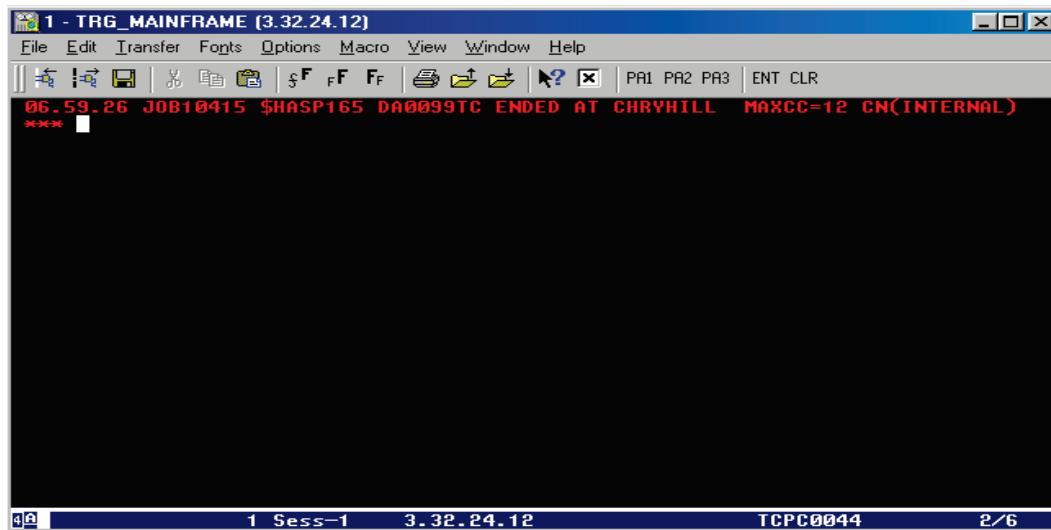
1 - TRG_MAINFRAME (3.32.24.12)
File Edit Transfer Fonts Options Macro View Window Help
File Edit Edit_Settings Menu Utilities Compilers Test Help
EDIT DA0099T.VANDANA.COBOL(SAMP) - 01.08 Member SAMP saved
Command ==> [ ] Scroll ==> CSR
***** **** Top of Data ****
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. SAMP.
000300 ENVIRONMENT DIVISION.
000400 DATA DIVISION.
000410 WORKING-STORAGE SECTION.
000500 PROCEDURE DIVISION.
000600 0000-MAIN.
000610     DISPLAY 'HELLO'.
000800     STOP RUN.
***** **** Bottom of Data ****

```

Figure 11: Sample COBOL program

Repeat the same step as discussed earlier to compile and link edit the program **SUB:=S.ST** and then press **ENTER**.

On pressing **ENTER**, you will get the following panel, which indicates the status of your job, which is **successful** or **unsuccessful**. In this case non-zero return code of 12 means there is some syntax errors in the COBOL program.

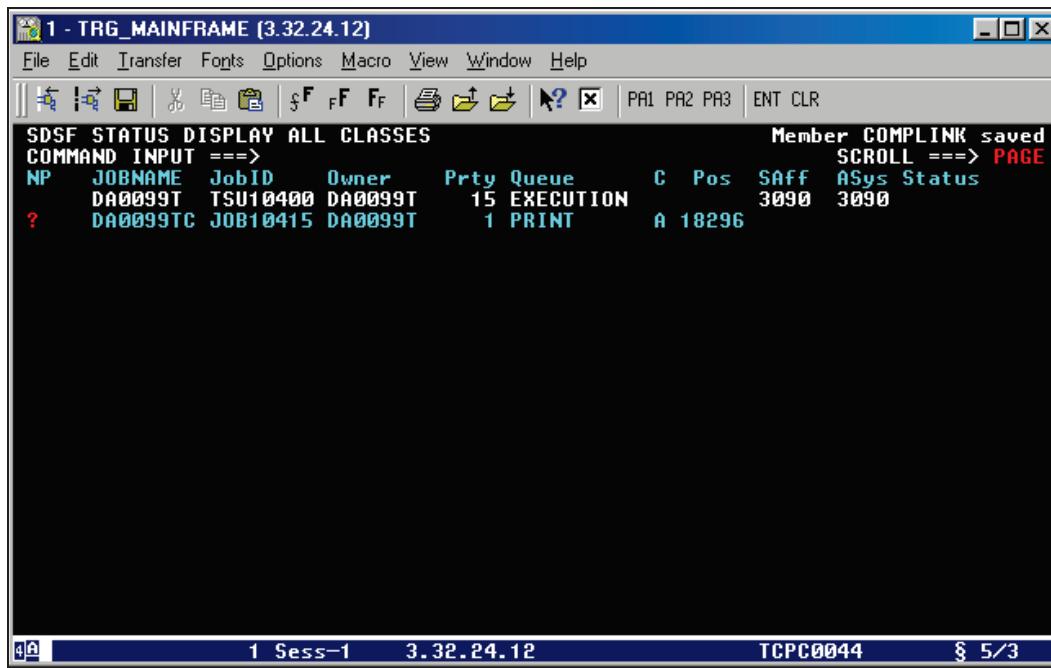


```
06.59.26 JOB10415 $HASP165 DA0099TC ENDED AT CHRYHILL MAXCC=12 CN(INTERNAL)
***
```

Figure 14: Notification message of COMPLINK JCL

Again, press **ENTER**, and the next panel will be displayed.

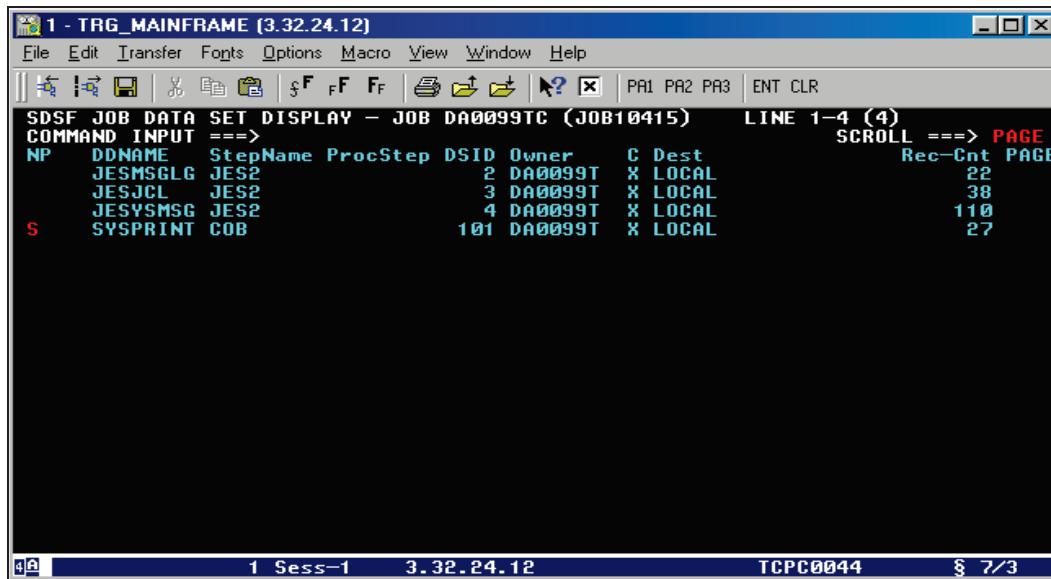
On this panel, position the cursor and type **?** as shown below, then press **ENTER**.



```
SDSF STATUS DISPLAY ALL CLASSES                               Member COMPLINK saved
COMMAND INPUT ==>                                         SCROLL ==> PAGE
NP   JOBNAME    JobID     Owner      Prtv Queue      C Pos  SAff  ASys Status
     DA0099T    TSU10400  DA0099T    15 EXECUTION  C 3090  3090
?     DA0099TC   JOB10415  DA0099T    1 PRINT      A 18296
```

Figure 15: Execution status of COMPLINK JCL

You get the next panel and type **S** as shown below and then press **ENTER**.



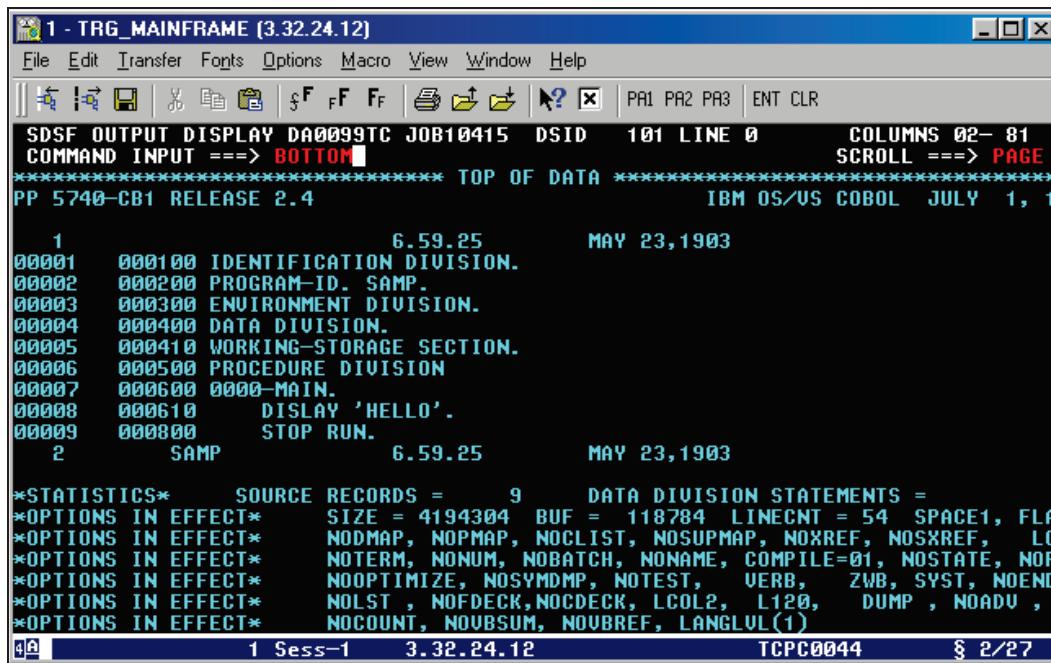
The screenshot shows a terminal window titled "1 - TRG_MAINFRAME (3.32.24.12)". The menu bar includes File, Edit, Transfer, Fonts, Options, Macro, View, Window, Help. The toolbar includes icons for file operations like Open, Save, Print, and a question mark. The main display area shows a table of job steps:

NP	DDNAME	StepName	ProcStep	DSID	Owner	C	Dest	SCROLL	Rec-Cnt	PAGE
	JESMSGLG	JES2		2	DA0099T	X	LOCAL		22	
	JESJCL	JES2		3	DA0099T	X	LOCAL		38	
	JESYMSG	JES2		4	DA0099T	X	LOCAL		110	
S	SYSPRINT	COB		101	DA0099T	X	LOCAL		27	

At the bottom of the screen, there are status indicators: 1 Sess-1, 3.32.24.12, TCPC0044, § 7/3.

Figure 16: List of files created after COMPLINK operation

After you get the next panel, type **BOTTOM** and then press **ENTER**.



The screenshot shows a window titled "1 - TRG_MAINFRAME (3.32.24.12)" displaying the output of a COBOL compilation. The window has a menu bar with File, Edit, Transfer, Fonts, Options, Macro, View, Window, Help. Below the menu is a toolbar with various icons. The main area displays the following text:

```

SDSF OUTPUT DISPLAY DA0099TC JOB10415 DSID 101 LINE 0 COLUMNS 02- 81
COMMAND INPUT ==> BOTTOM SCROLL ==> PAGE
***** TOP OF DATA *****
PP 5740-CB1 RELEASE 2.4 IBM OS/VSE COBOL JULY 1, 1

1 6.59.25 MAY 23,1903
00001 000100 IDENTIFICATION DIVISION.
00002 000200 PROGRAM-ID. SAMP.
00003 000300 ENVIRONMENT DIVISION.
00004 000400 DATA DIVISION.
00005 000410 WORKING-STORAGE SECTION.
00006 000500 PROCEDURE DIVISION.
00007 000600 0000-MAIN.
00008 000610 DISPLAY 'HELLO'.
00009 000800 STOP RUN.
2 SAMP 6.59.25 MAY 23,1903

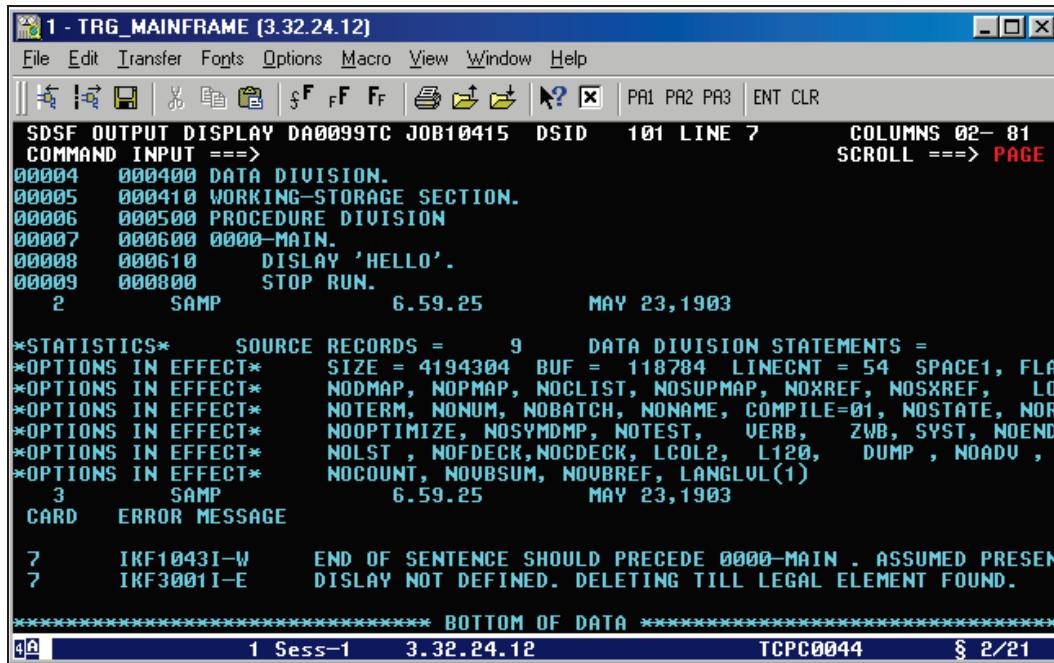
*STATISTICS* SOURCE RECORDS = 9 DATA DIVISION STATEMENTS =
*OPTIONS IN EFFECT* SIZE = 4194304 BUF = 118784 LINECNT = 54 SPACE1, FLA
*OPTIONS IN EFFECT* NODMAP, NOPMAP, NOCLIST, NOSUPMAP, NOXREF, NOSXREF, LO
*OPTIONS IN EFFECT* NOTERM, NONUM, NOBATCH, NONAME, COMPILE=01, NOSTATE, NOR
*OPTIONS IN EFFECT* NOOPTIMIZE, NOSVMDMP, NOTEST, VERB, ZWB, SYST, NOEND
*OPTIONS IN EFFECT* NOLST, NOFDECK, NOCDECK, LCOL2, L120, DUMP, NOADV ,
*OPTIONS IN EFFECT* NOCOUNT, NOUBSUM, NOUBREF, LANGLVL(1)

4@ 1 Sess-1 3.32.24.12 TCPC0044 § 2/27

```

Figure 17: Compilation Errors of sample COBOL program

You get to the bottom of screen where you can see the error listing with line number, which enables the user to debug the error.



```

1 - TRG_MAINFRAME (3.32.24.12)
File Edit Transfer Fonts Options Macro View Window Help
SDSF OUTPUT DISPLAY DA0099TC JOB10415 DSID 101 LINE 7 COLUMNS 02- 81
COMMAND INPUT ==> SCROLL ==> PAGE
00004 000400 DATA DIVISION.
00005 000410 WORKING-STORAGE SECTION.
00006 000500 PROCEDURE DIVISION
00007 000600 0000-MAIN.
00008 000610      DISPLAY 'HELLO'.
00009 000800      STOP RUN.
2      SAMP          6.59.25      MAY 23,1903

*STATISTICS*      SOURCE RECORDS = 9      DATA DIVISION STATEMENTS =
*OPTIONS IN EFFECT*      SIZE = 4194304  BUF = 118784  LINECNT = 54  SPACE1, FLA
*OPTIONS IN EFFECT*      NODMAP, NOPMAP, NOCLIST, NOSUPMAP, NOXREF, NOSXREF, LO
*OPTIONS IN EFFECT*      NOTERM, NONUM, NOBATCH, NONAME, COMPILE=01, NOSTATE, NOR
*OPTIONS IN EFFECT*      NOOPTIMIZE, NOSYMDMP, NOTEST, VERB, ZWB, SYST, NOEND
*OPTIONS IN EFFECT*      NOLST, NOFDECK, NOCDECK, LCOL2, L120, DUMP, NOADU,
*OPTIONS IN EFFECT*      NOCOUNT, NOVBSUM, NOUBREF, LANGLVL(1)
3      SAMP          6.59.25      MAY 23,1903
CARD  ERROR MESSAGE

7      IKF1043I-W      END OF SENTENCE SHOULD PRECEDE 0000-MAIN . ASSUMED PRESEN
7      IKF3001I-E      DISPLAY NOT DEFINED. DELETING TILL LEGAL ELEMENT FOUND.

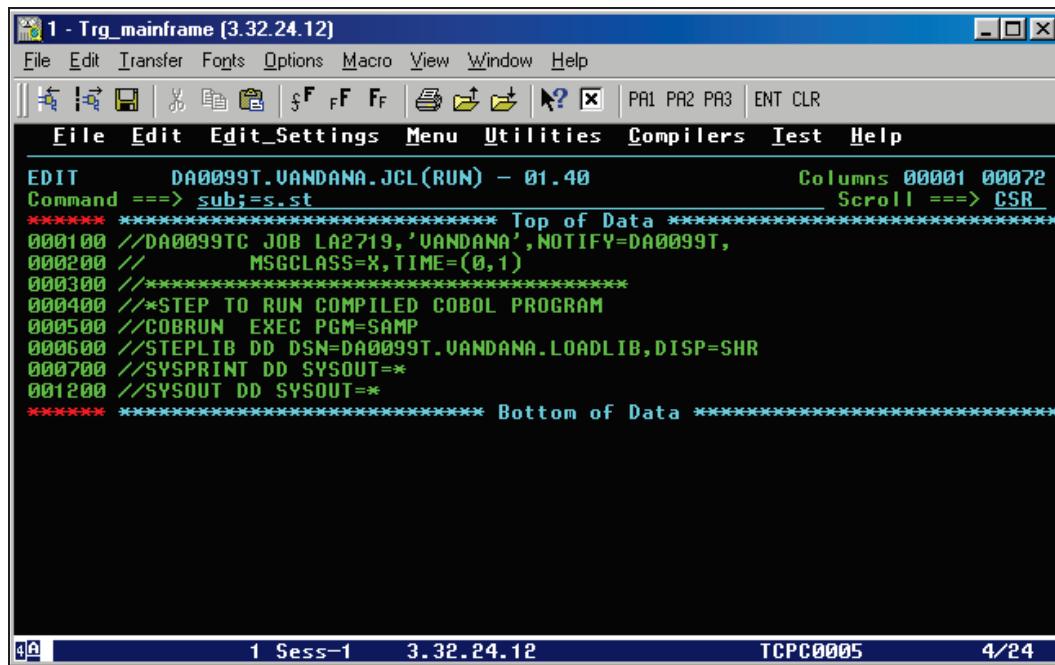
***** BOTTOM OF DATA *****
4@ 1 Sess-1 3.32.24.12 TCPC0044 § 2/21

```

Figure 18: Compilation Errors of sample COBOL program (Cont...)

Once you have debugged your COBOL program, repeat the above steps for COMPLINK viz., **SUB; =S.ST**

Once the compilation and linking is successful, execute the COBOL program. For this open the **RUN** jcl and repeat the same above steps, namely, **SUB;=S.ST**.

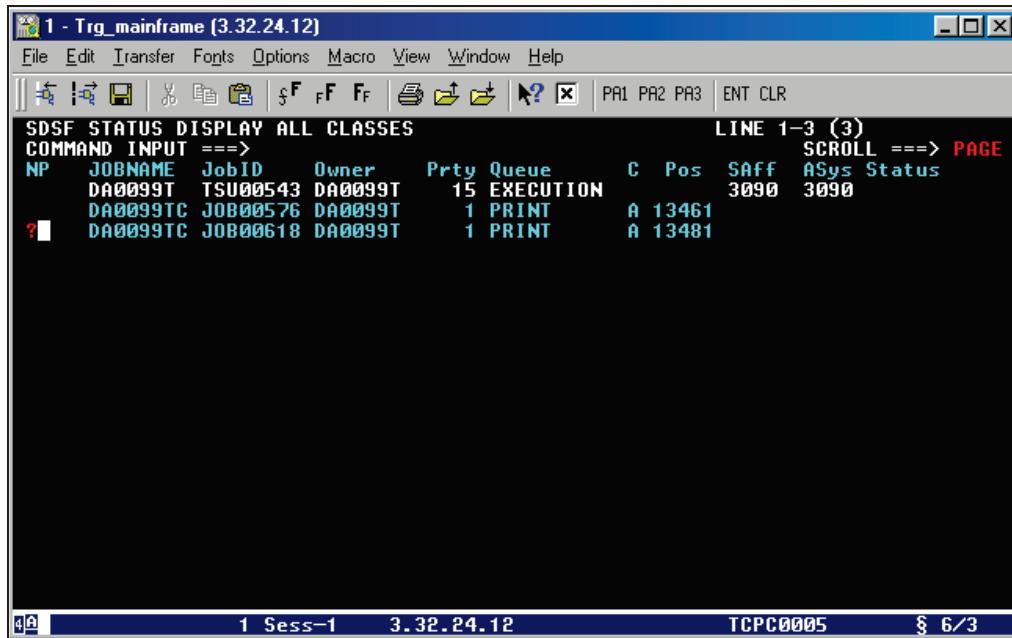


The screenshot shows a mainframe terminal window titled "1 - Trg_mainframe (3.32.24.12)". The window has a menu bar with "File", "Edit", "Transfer", "Fonts", "Options", "Macro", "View", "Window", and "Help". Below the menu is a toolbar with icons for file operations like Open, Save, Print, and Cut/Paste. The main area displays JCL (Job Control Language) code:

```
EDIT      DA0099T.VANDANA.JCL(RUN) - 01.40          Columns 00001 00072
Command ==> sub:=s.st                               Scroll ==> CSR
***** **** Top of Data ****
000100 //DA0099TC JOB LA2719,'VANDANA',NOTIFY=DA0099T,
000200 //           MSGCLASS=X,TIME=(0,1)
000300 //*****
000400 //**STEP TO RUN COMPILED COBOL PROGRAM
000500 //COBRUN EXEC PGM=SAMP
000600 //STEPLIB DD DSN=DA0099T.VANDANA.LOADLIB,DISP=SHR
000700 //SYSPRINT DD SYSOUT=*
001200 //SYSOUT DD SYSOUT=*
***** **** Bottom of Data ****
```

The bottom status bar shows "1 Sess-1 3.32.24.12", "TCP/C0005", and "4/24".

Figure 19: Sample Run JCL



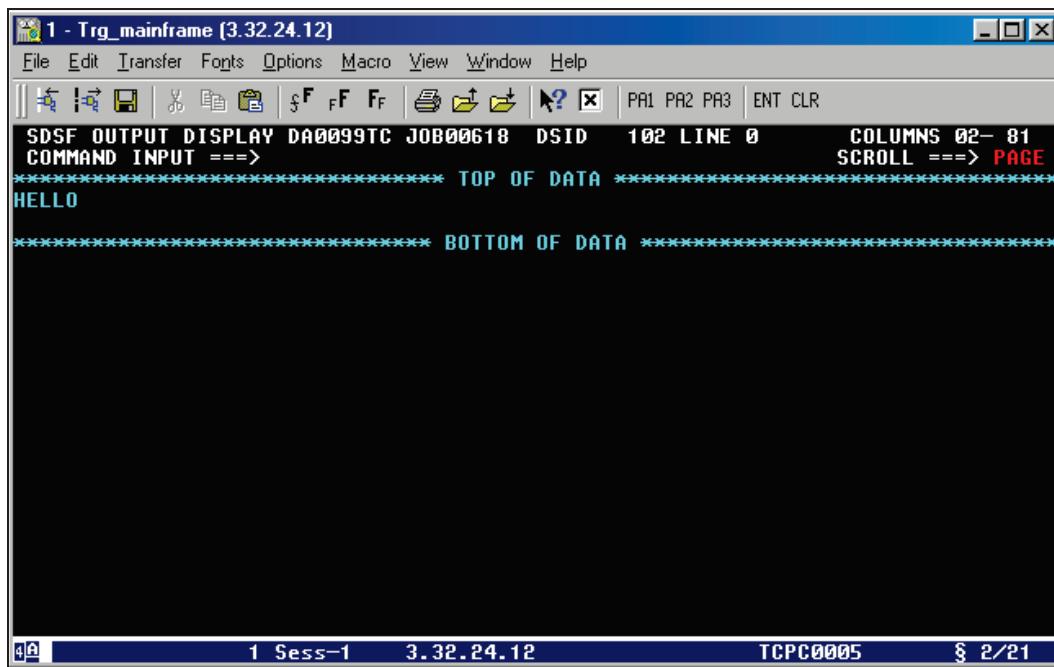
```

1 - Trg_mainframe [3.32.24.12]
File Edit Transfer Fonts Options Macro View Window Help
SDSF STATUS DISPLAY ALL CLASSES LINE 1-3 (3)
COMMAND INPUT ==> SCROLL ==> PAGE
NP  JOBNAME JobID Owner Prty Queue C Pos SAFF ASys Status
DA0099T TSU00543 DA0099T 15 EXECUTION 3090 3090
DA0099TC JOB00576 DA0099T 1 PRINT A 13461
? DA0099TC JOB00618 DA0099T 1 PRINT A 13481

```

40 1 Sess-1 3.32.24.12 TCP/C0005 § 6/3

Figure 21: Execution status of Run JCL



```

1 - Trg_mainframe [3.32.24.12]
File Edit Transfer Fonts Options Macro View Window Help
SDSF OUTPUT DISPLAY DA0099TC JOB00618 DSID 102 LINE 0 COLUMNS 02- 81
COMMAND INPUT ==> SCROLL ==> PAGE
***** TOP OF DATA *****
HELLO
***** BOTTOM OF DATA *****

```

40 1 Sess-1 3.32.24.12 TCP/C0005 § 2/21

Figure 22: Output of Run JCL

(The output as it appears in SYSOUT COBRUN.)

Map the IBM MVS Environment into Windows 95 or Windows NT

Introduction:

“CA-Realia for JCL” enables you to use **IBM MVS Job Control Language** as an execution control language in the PC environment. With “CA-Realia for JCL”, you define a relationship between the constructs of MVS JCL and MVS file system and Windows 95 or Windows NT files.

MVS Datasets versus Windows 95 or Windows NT Files:

In IBM MVS, uncataloged datasets are uniquely defined by their fully qualified dataset name and the volume where they reside.

For example:

VOLUME=SER=BS3007,DSN=DA0001T.PATNI.TESTFILE

It identifies a dataset name **DA0001T.PATNI.TESTFILE** on volume **BS3007**. The same dataset name on a different volume is a different dataset. The different dataset names on different volumes are different datasets.

In Windows 95 or Windows NT, a file is identified by drive letter, directory, filename, and extension.

For example:

D:\DA0001T.PATNI.TESTFILE.IBM

It is a Windows 95 or Windows NT file on drive D: in directory \DA0001T\PATNI with filename TESTFILE and extension .IBM.

You must define an association between IBM MVS dataset names and Windows 95 or Windows NT full filenames. The association of the IBM MVS file VOLUME=SER=BS3007,DSN=DA0001T.PATNI.TESTFILE and Windows 95 or Windows NT file D:\DA0001T.PATNI.TESTFILE.IBM is fairly obvious:

- IBM MVS volume BS3007 corresponds to Windows 95 or Windows NT drive D:
- IBM MVS name level DA0001T corresponds to Windows 95 or Windows NT directory \DA0001T
- IBM MVS name level PATNI corresponds to Windows 95 or Windows NT sub-directory \PATNI
- IBM MVS name level TESTFILE corresponds to Windows 95 or Windows NT filename TESTFILE with .IBM extension in the directory \DA0001T\PATNI

CA-Realia for JCL permits a flexible mapping of IBM MVS datasets into Windows 95 or Windows NT files by generalizing the type of relationship.

VOLUME NAME MAPPING

The Windows 95 or Windows NT drive/directory names is specified by a &VOLUME directive in the CA-Realia for JCL configuration file. It possesses a structure as shown below:

```
/*&VOLUME:ibm-mvs-volume-ser/ibm-mvs-unit;dos-drive-directory
```

where:

ibm-mvs-volume-ser is the IBM volume serial number you specify in the VOLUME=SER parameter in the DD card.

ibm-mvs-unit is the IBM mainframe device-type of this volume as referenced in a UNIT=parameter in a DD statement.

dos-drive-directory is the Windows 95 or Windows NT identifier for the drive and directory to be associated with the IBM MVS volume.

Following are valid specifications of a &VOLUME directive:

```
/*&VOLUME:SYS000/3350;E:\VSY0  
/*&VOLUME:TESTA/3350;E:TEST\A  
/*&VOLUME:BS3007/3390;D:\
```

The last one corresponds to the volume mapping used in the example shown above. With these definitions, volumes SYS000 and TESTA both reside on the same Windows 95 or Windows NT drive D:.

DATASET NAME MAPPING:

After you map volume serial numbers to their corresponding Windows 95 or Windows NT directories, any uncataloged datasets can be mapped to Windows 95 or Windows NT files.

Simple Datasets:

For simple IBM MVS datasets, CA-Realia for JCL converts each qualification level name in the IBM MVS dataset name to a Windows 95 or Windows NT directory, except for the last level name. The last level name is converted into the filename. CA-Realia adds an extension to the filename. The default extension is .IBM and it will be used unless there is &DSNAME directive specifying an extension.

Using the &VOLUME directive examples above to define the volumes SYS000, TESTA, and BS3007, here are some examples of IBM MVS dataset name conversion to Windows 95 or Windows NT filenames.

IBM MVS	Windows 95 or Windows NT
VOLUME=SER=SYS000, DSN=SYS1.DEF.WORKFILE	E:\VSY0\SYS1\DEF\WORKFILE.IBM
VOLUME=SER=TESTA, DSN=F045J7.AR.TESTFILE	E:\TEST\A\F045J7\AR\TESTFILE.IBM
VOLUME=SER=BS3007, DSN=DA0001T.PATNI.TESTFILE	D:\DA0001T\PATNI\TESTFILE.IBM
VOLUME=SER=BS3007, DSN=DA0001T.EMPPFILE	D:\DA0001T\EMPPFILE.IBM

Partitioned Datasets:

For partitioned datasets (PDS), each member of the PDS must be converted into a separate Windows or Windows NT file. CA-Realia for JCL converts each qualification level name in the IBM MVS PDS name to a Windows 95 or Windows NT directory name,

including the last level name. We append the extension **.PDS**, to the last directory name. The PDS member name is converted into the filename.

Many partitioned datasets are used to hold the JCL, procedures, source programs, and executable programs. For these, Windows 95 or Windows NT requires a specific extension.

For example: Executable programs must have an extension of .EXE or .COM; and COBOL source programs have extension of .COB or .CBL. To do this, you can use &DSNAME directive as shown below:

```
/*&DSNAME:pds-data-set-name(*);dos-directory*.dos-ext
```

where:

pds-data-set-name is the IBM MVS PDS dataset name with no member.

dos-directory is the path for the files corresponding to pds-data-set-name.

dos-ext is the Windows 95 or Windows NT filename extension required for this dataset.

For example, you can specify the following for two PDSs:

```
/*DSN:SYS1.PROCLIB(*);SYS1\PROCLIB.PDS\*.JCL
```

```
/*DSN:AR.JOBLIB(*);AR\JOBLIB.PDS\*.EXE
```

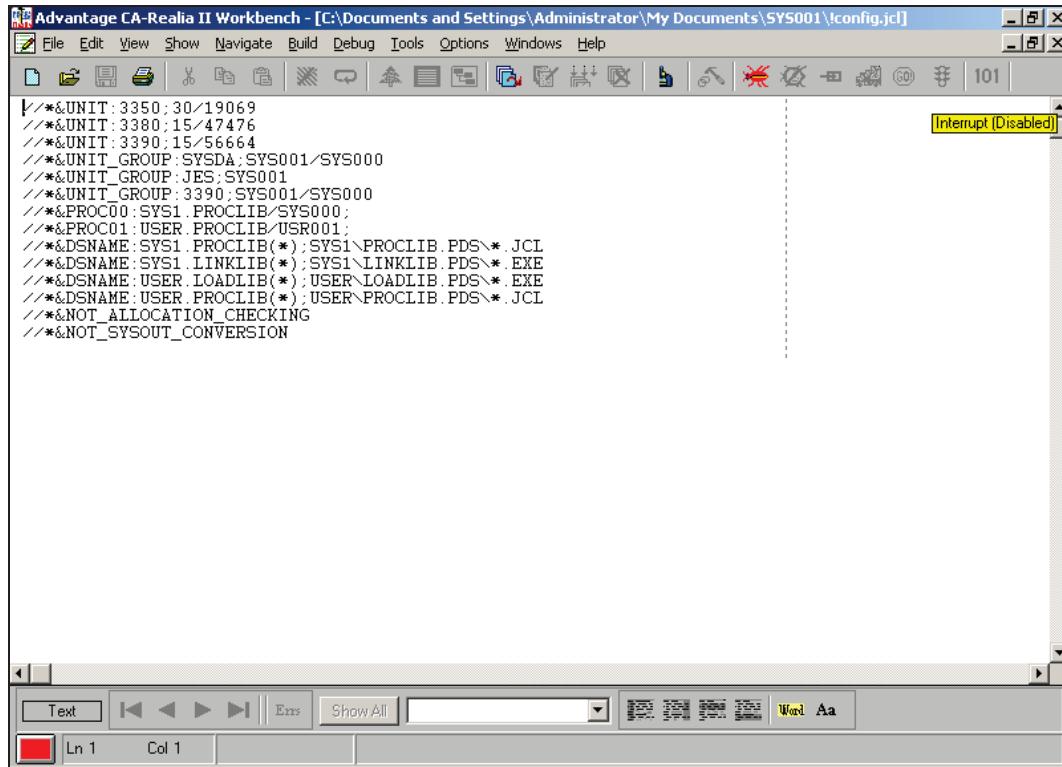
```
/*DSN:TEST.COBLIB(*);TEST\COBLIB.PDS\*.JCL
```

IBM MVS	Windows 95 or Windows NT
VOLUME=SER=SYS000, DSN=SYS1.PROCLIB(TRS06)	E:\VSY0\SYS1\PROCLIB.PDS\TRS06.JCL
VOLUME=SER=TESTA, DSN=AR.JOBLIB(BETA01)	E:\TESTA\JOBLIB.PDS\BETA01.EXE
VOLUME=SER=BS3007, DSN=DA0001T.PATNI.SRCLIB(ASS1)	D:\DA0001T\PATNI\SRCLIB.PDS\ASS1.CBL
VOLUME=SER=BS3007, DSN=DA0001T.PATNI.OBJLIB(ASS1)	D:\DA0001T\PATNI\OBJLIB.PDS\ASS1.OBJ
VOLUME=SER=BS3007, DSN=DA0001T.PATNI.PROCLIB(IEBGEN)	D:\DA0001T\PATNI\PROCLIB.PDS\IEBGEN.JCL
VOLUME=SER=BS3007, DSN=DA0001T.PATNI.LINKLIB(ASS1)	D:\DA0001T\PATNI\SRCLIB.PDS\ASS1.EXE

Configuration:

Configuration File:

The configuration file in the installation directory !CONFIG.JCL, determines how the IBM MVS system is mapped into Windows 95 or Windows NT on your computer. It contains a number of configuration directives.



```

Advantage CA-Realia II Workbench - [C:\Documents and Settings\Administrator\My Documents\SYS001\!config.jcl]
File Edit View Show Navigate Build Debug Tools Options Windows Help
File Explorer | Find | Open | Save | Print | Copy | Paste | Cut | Delete | Insert | Properties | Refresh | Exit | Help | Interrupt (Disabled) | 101 |
/*&UNIT:3350;30/19069
/*&UNIT:3380;15/47476
/*&UNIT:3390;15/56664
/*&UNIT_GROUP:SYSDA:SYS001/SYS000
/*&UNIT_GROUP:JES:SYS001
/*&UNIT_GROUP:3390:SYS001/SYS000
/*&PROC00:SYS1.PROCLIB/SYS000;
/*&PROC01:USER PROCLIB/USR001;
/*&DSNAME:SYS1.PROCLIB(*);SYS1\PROCLIB.PDS\*.* JCL
/*&DSNAME:SYS1.LINKLIB(*);SYS1\LINKLIB.PDS\*.* EXE
/*&DSNAME:USER LOADLIB(*);USER\LOADLIB.PDS\*.* EXE
/*&DSNAME:USER PROCLIB(*);USER\PROCLIB.PDS\*.* JCL
/*&NOT_ALLOCATION_CHECKING
/*&NOT_SYSOUT_CONVERSION

```

Figure 87: RealJCL !config.jcl (Screen Shot)

Run the **setupjcl.bat** from the folder where Ca-Realia is installed (For example: E:\CAWB31).The following **!config.jcl** is created as a result of this.

```

/*&UNIT:3350;30/19069
/*&UNIT:3380;15/47476
/*&UNIT:3390;15/56664
/*&VOLUME:SYS000/3390;E:\cawb31\SYS000
/*&VOLUME:SYS001/3390;D:\DOCUME~1\koleshva\MYDOCU~1\SYS001
/*&VOLUME:USR001/3390;D:\DOCUME~1\koleshva\MYDOCU~1\SYS001

```

```
/*&UNIT_GROUP:SYSDA;SYS001/SYS000
/*&UNIT_GROUP:JES;SYS001
/*&UNIT_GROUP:3390;SYS001/SYS000
/*&PROC00:SYS1.PROCLIB/SYS000;
/*&PROC01:USER.PROCLIB/USR001;
/*&DSNAME:SYS1.PROCLIB(*);SYS1\PROCLIB.PDS\*.JCL
/*&DSNAME:SYS1.LINKLIB(*);SYS1\LINKLIB.PDS\*.EXE
/*&DSNAME:USER.LOADLIB(*);USER\LOADLIB.PDS\*.EXE
/*&DSNAME:USER.PROCLIB(*);USER\PROCLIB.PDS\*.JCL
/*&NOT_ALLOCATION_CHECKING
/*&NOT_SYSOUT_CONVERSION
```

Example 1: RealJCL !config.jcl

Note: The volume specification may differ as per the installation. Also the setupjcl.bat has to be run only once initially. If the settings as mentioned above are already present then the setupjcl.bat need not be run.

Configuring the Configuration file

Follow the steps given below to configure the **!config.jcl** file in order to customize the Windows environment to IBM mainframe environment:

Step 1: Create **JCL.PDS** folder in your own path. (for example: C:\ramu\jcl)

Step 2: Copy the **!config.jcl** to **JCL.PDS**.

Step 3: Customize the JCL configuration file (**!config.jcl**) as follows:

```
/*&UNIT:3350;30/19069
/*&UNIT:3380;15/47476
/*&UNIT:3390;15/56664
/*&VOLUME:SYS000/3390;C:\cawb31\SYS000
/*&VOLUME:SYS001/3390;C:\DOCUME~1\ramuthiy\MYDOCU~1\SYS001
/*&VOLUME:USR001/3390;C:\DOCUME~1\ramuthiy\MYDOCU~1\SYS001
/*&VOLUME:BS3007/3390;C:\RAMU\JCL
/*&UNIT_GROUP:SYSDA;BS3007/SYS001/SYS000
/*&UNIT_GROUP:JES;BS3007/SYS001
/*&UNIT_GROUP:3390;BS3007/SYS001/SYS000
/*&PROC00:DA0001T.RAMU.PROCLIB/BS3007;
/*&PROC01:SYS1.PROCLIB/SYS000;
/*&PROC02:USER.PROCLIB/USR001;
/*&DSNAME:SYS1.PROCLIB(*);SYS1\PROCLIB.PDS\*.JCL
/*&DSNAME:SYS1.LINKLIB(*);SYS1\LINKLIB.PDS\*.EXE
/*&DSNAME:USER.LOADLIB(*);USER\LOADLIB.PDS\*.EXE
/*&DSNAME:USER.PROCLIB(*);USER\PROCLIB.PDS\*.JCL
/*&DSNAME:DA0001T.RAMU.COBOL(*);JCL.PDS\*.COB
/*&DSNAME:DA0001T.RAMU.JCL(*);JCL.PDS\*.JCL
/*&DSNAME:DA0001T.RAMU.PROCLIB(*);JCL.PDS\*.JCL
```

```
/*&DSNAME:DA0001T.RAMU.LOADLIB(*);JCL.PDS\*.EXE  
/*&DSNAME:DA0001T.RAMU.OBJLIB(*);JCL.PDS\*.OBJ  
/*&DSNAME:DA0001T.EMPFILE(*);JCL.PDS\EMPFILE.IBM  
/*&DSNAME:DA0001T.FILE.;JCL.PDS\*.IBM  
/*&NOT_ALLOCATION_CHECKING  
/*&NOT_SYSOUT_CONVERSION
```

Example 2: Customized !config.jcl

Step 4: Create all your source code, object code, executable files and data files in the same JCL.PDS.

Step 5: Make **JCL.PDS** as your working directory.

Program Execution

Programs can be in one of the three places to be executed by CA-Realia for JCL:

Lab 1. SYS1.LINKLIB (Path:

C:\DOCUME~1\ramuthiy\MYDOCU~1\SYS001\LINKLIB.PDS)

Lab 2. A user-defined JOBLIB (Path: C:\ramu\jcl\JCL.PDS, if you customize your !config.jcl as above)

Lab 3. A user-defined STEPLIB (Path: C:\ramu\jcl\JCL.PDS, if you customize your !config.jcl as above)

Your development environment must know the Windows 95 or Windows NT name of this drive/directory. You can use the rules for conversion from IBM dataset names to Windows 95 or Windows NT files names via the information in the configuration file.

Executing JCL Using CA-Realia II workbench

Introduction

This section describes how to execute job step programs that call Windows 95 or Windows NT programs using the CA-Realia Workbench.

This section discusses the following:

- Defining an execution profile
- Creating or editing CA-Realia for JCL session options

Defining an Execution Profile

Before you execute your JCL using CA-Realia Workbench, you must first define an execution profile. To define an execution profile, follow the steps given below:

Step 1: To open a JCL file, select the **File** menu and choose **Open**.

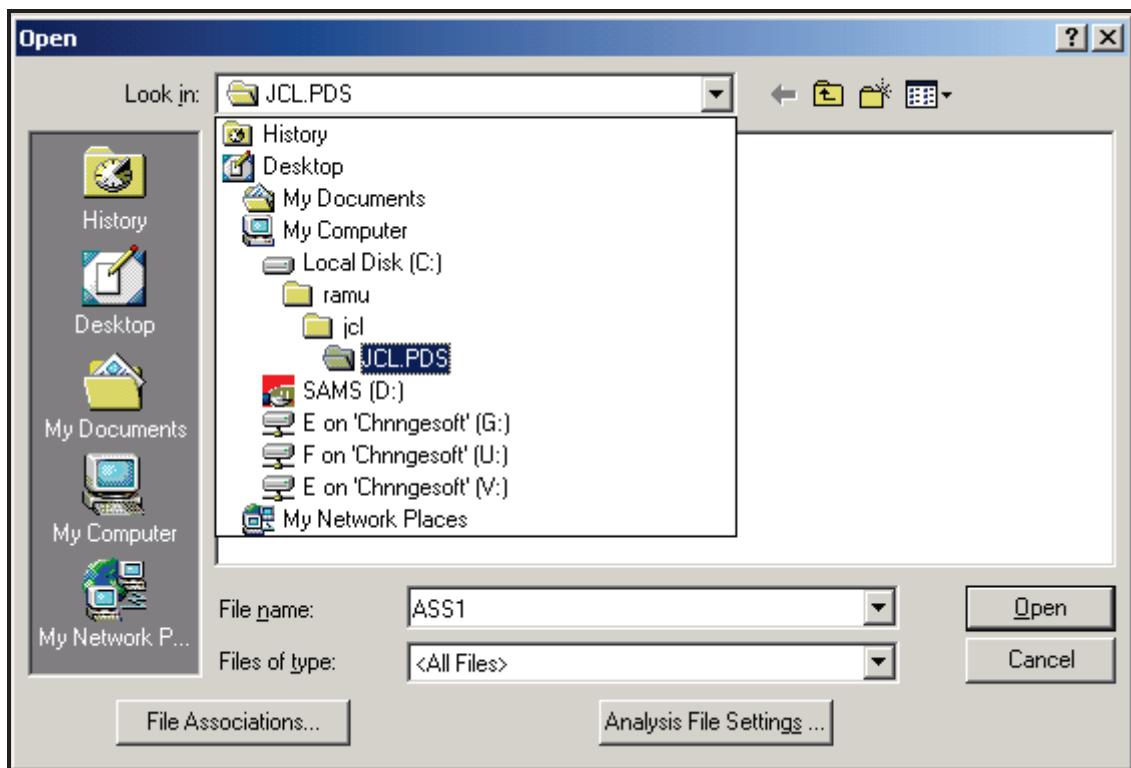
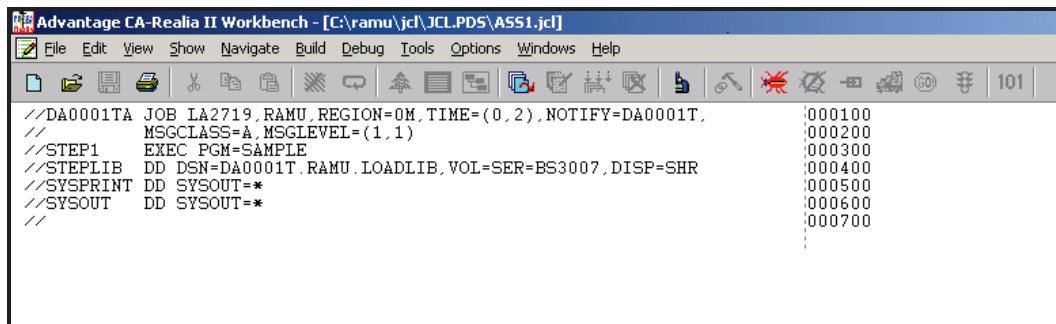


Figure 88: Open Menu for an existing JCL file

Step 2: The Run jcl file is displayed.



```
//DA0001TA JOB LA2719, RAMU, REGION=0M, TIME=(0,2), NOTIFY=DA0001T,
//          MSGCLASS=A, MSGLEVEL=(1,1)
//STEP1    EXEC PGM=SAMPLE
//STEPLIB  DD DSN=DA0001T.RAMU.LOADLIB, VOL=SER=BS3007, DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//
```

Figure 89: Sample Run JCL file (ASS1.jcl)

Step 3: Select the **Build** menu, and choose **Options**.

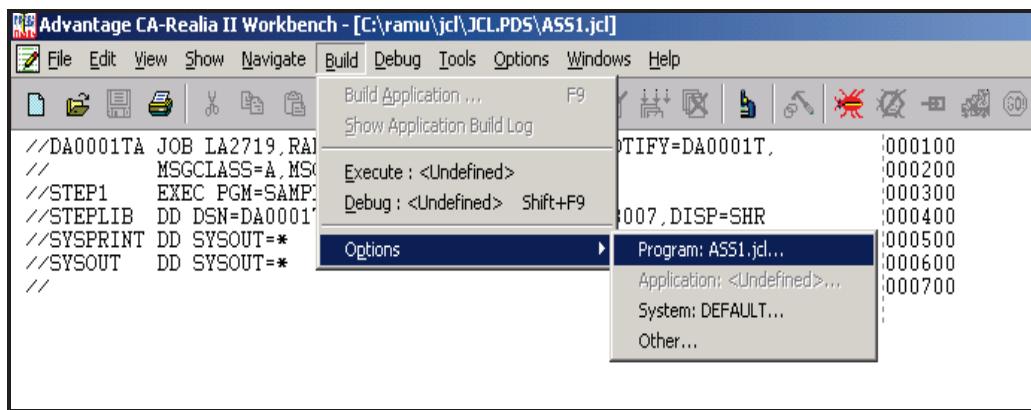


Figure 90: Creation of session options for Run JCL

Step 4: If your program is not yet defined, choose **Program: <Undefined>**. The **Select Application/Program Type** dialog will be displayed. Select **JCL Model** and click **OK**.

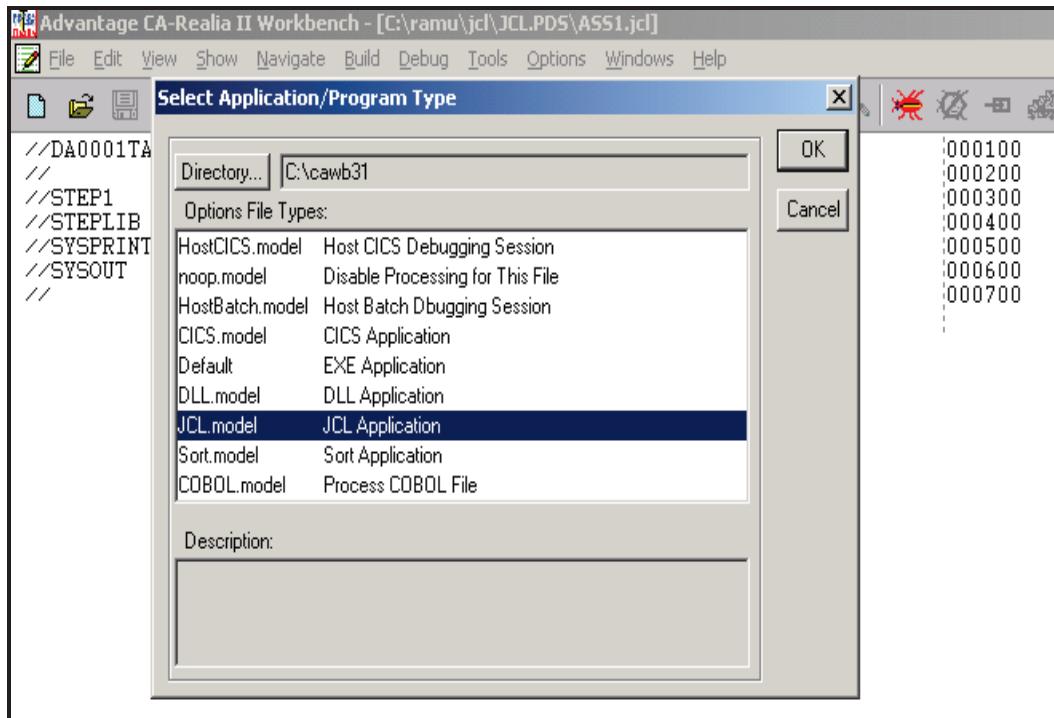


Figure 91: Selection of JCL Model

Step 5: If your program is already defined, choose Program: *yourprogrammname* from the **Build/Options** dialog. The **RealJCL Jobrun Options Notebook** will be displayed. Here you can specify whether you are creating a new profile or editing an existing profile.

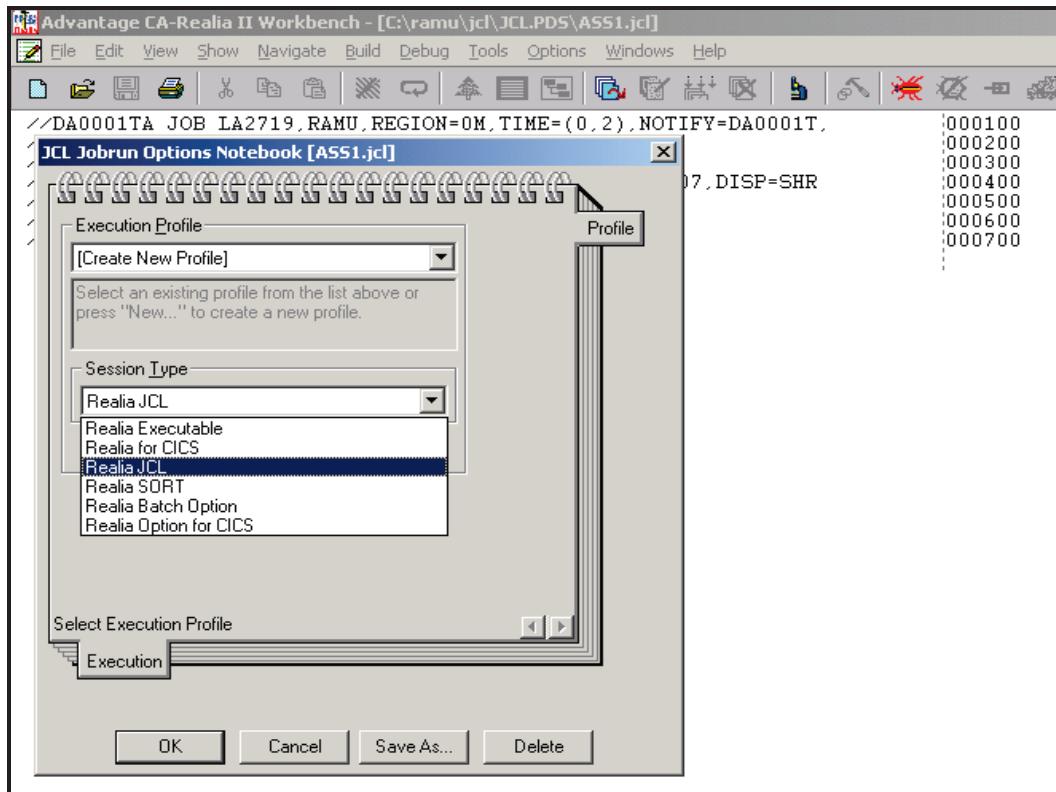


Figure 92: Selection of Session Type

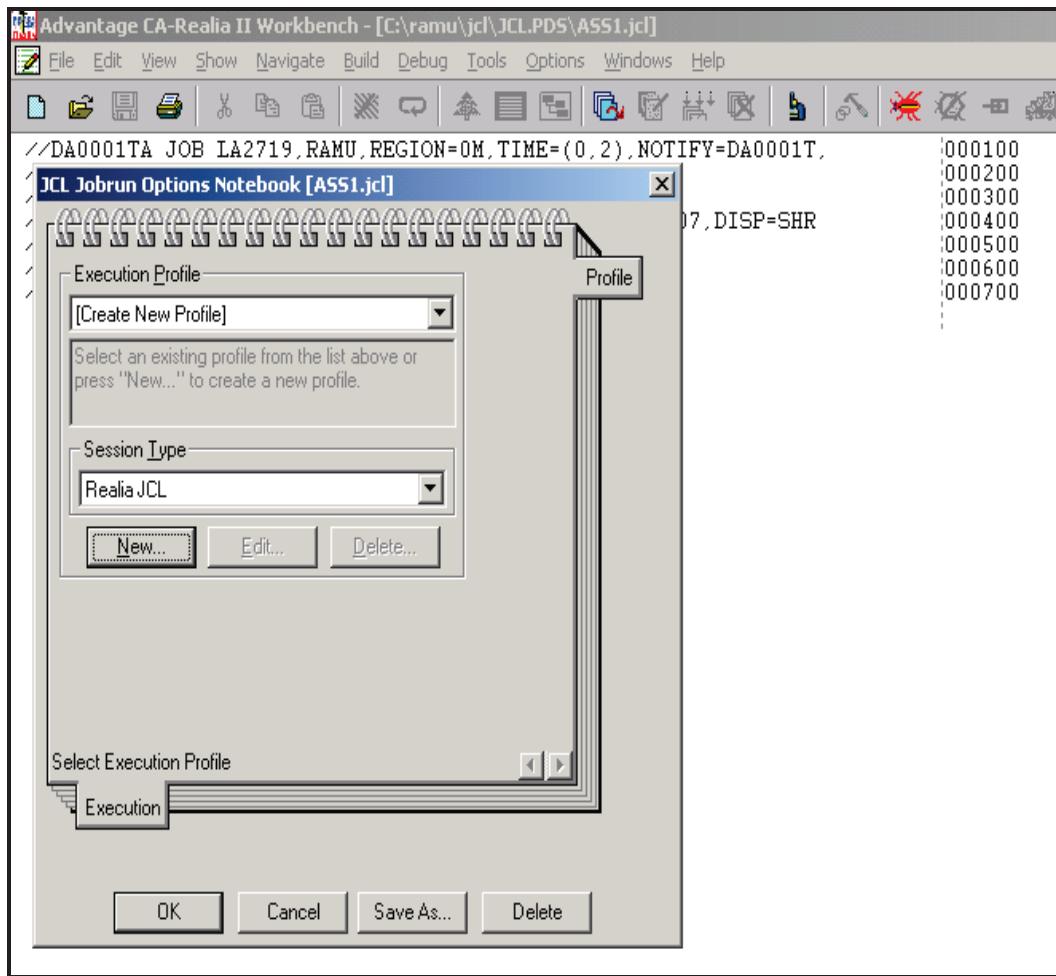


Figure 93: Creation of new Profile

You will see the following elements in the **JCL Jobrun Options Notebook**:

- **Execution Profile:** This drop-down combination box lists all existing execution profiles.
- **New:** Click this button to define a new execution profile.
- **Edit:** Click this button to change an existing execution profile.
- **Delete:** Click this button to delete the highlighted execution profile.

- **Session Type:** This field identifies the type of application to debug. Select Realia JCL for this debug session if it is not already selected.

Step 6: Click **New** to create session options for a new profile or **Edit** to edit session options for an existing profile.

Creating or Editing CA-Realia for JCL Session Options:

When you click **New** or **Edit** on the **RealJCL Jobrun Options Notebook** dialog, the **Realia for JCL Session Options Notebook** is displayed. Here you define your session options to your system.

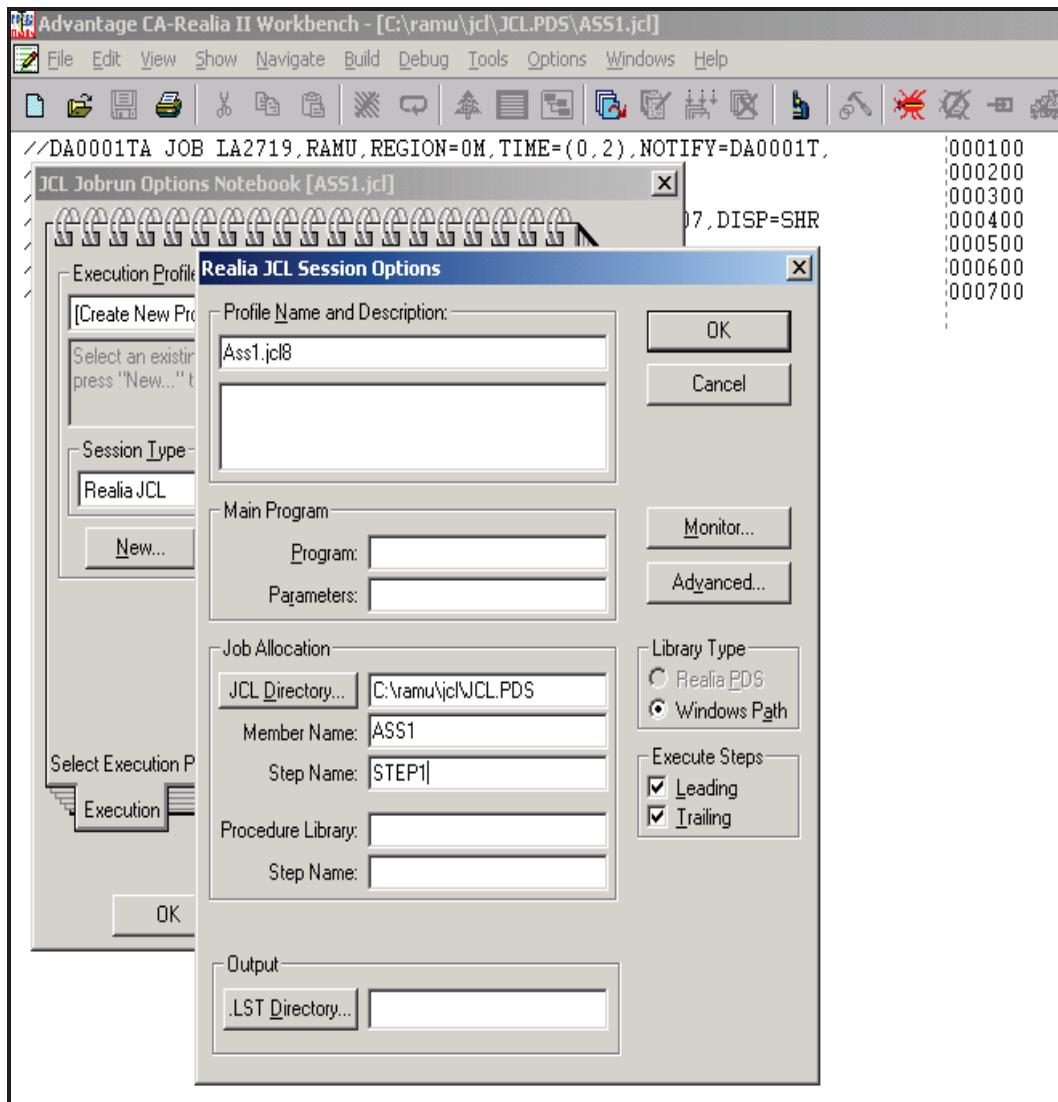


Figure 94: Creation or Editing Session Options

The **JCL directory** in this case is **C:\ramu\jcl\jcl.pds**. The drive name may be different as per the installation.

You will see the following elements on the Realia:

- **Profile Name and Description:** In this field, you can enter a name for the session options profile you are creating. You can then enter description of the profile.

- **Program:** In this field, you can enter the name of the main program that should be executed when this profile is selected for the debug session. This is the name that appears in an EXEC PGM= JCL statement or the JCL step name identified in this dialog. This field is required for debugging only. It is ignored if you are not debugging a job step.
- **Parameters:** In this field, you can enter parameters to override parameters on the JCL EXEC statement. If you do not supply any parameters here, then the parameters on JCL EXEC statement are used. This field is required for debugging only.
- **JCL Directory:** In this field, you can enter the name of Windows drive and directory that contain the JCL member that defines the data set allocations to be performed when the application is executed. This field is required.
- **Member Name:** In this field, you can enter the name of the member within the JCL directory that is going to be executed. An extension of .JCL is assumed. This field is required.
- **Step Name:** In this field, you can enter the name of the step within the JCL member that should be executed. The step name must correspond the program name shown in the Program field. This field is required for debugging only.
- **Procedure Library:** In this field, you can specify the procedure library that contains the external procedure invoked by specified steps within the JCL member. If the JCL step invokes the main program directly, Procedure Library and Step Name must be left blank. This field is required for debugging only.
- **Step Name:** In this field, you can specify the name of the step within the procedure that is going to be executed, if any. The step name must correspond to the program name shown in the Program field. This field is required for debugging only.
- **Library Type - Windows Path:** This option indicates the drive and directory where the execution profile resides. Realia PDS is reserved for future use. This field is required for debugging only.
- **Execute Steps – Leading:** Select this check box if the JCL member contains steps that you want to be executed before the step has to be debugged. This field is required for debugging only.
- **Execute Steps – Trailing:** Select this check box if the JCL member contains steps that you want to be executed following the step to be debugged. This field is required for debugging only.

Advanced Options

To modify the execution of CA-Realia for JCL for the current run, click **Advanced** on the **Realia for JCL Session Options**. A dialog box will be displayed.

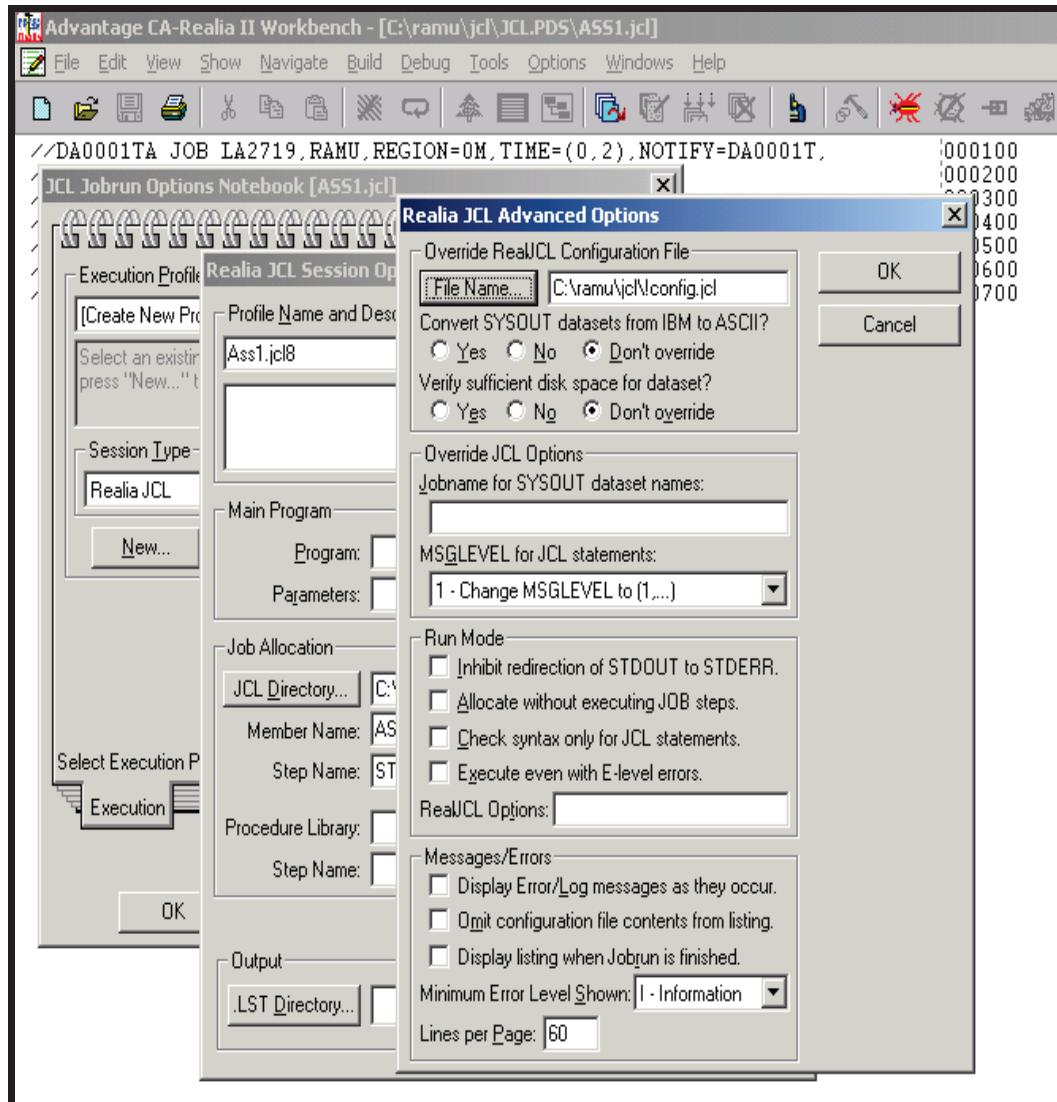


Figure 95: Advanced Options

File Name: CA-Realia for JCL normally uses the configuration file !CONFIG.JCL in the installation directory where it finds the REALJCL.EXE module. This field lets you point to an alternate configuration file. Here you select the customized **!config.jcl** file so that your path and volume will be selected for execution.

Running and Debugging a Program

To proceed with running your job, select **Build** and then **Execute** while your profile is displayed. When the execution completes, you can debug your job by selecting **Build** and then **Debug**.

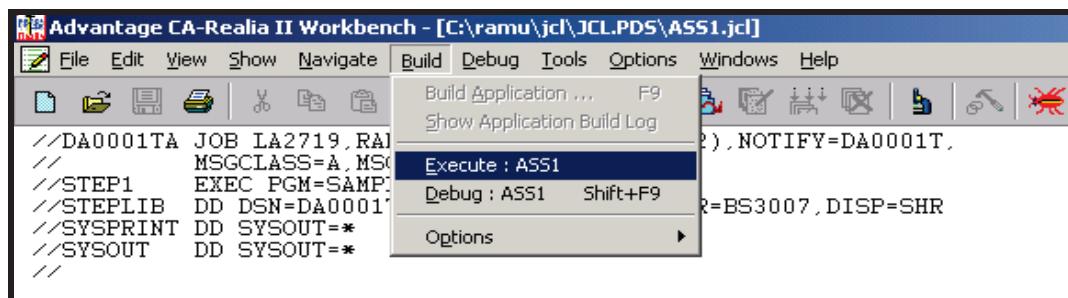


Figure 96: Execution of Run JCL file

Executing procedures:

A) Cataloged procedures:

To create and invoke a cataloged procedure, proceed with the following steps.

Step 1: Write down a COBOL program, compile, link and create an executable. Store the executable in D:\cawb31\SAMPLES\jcl.pds.

Following is the COBOL program which is invoked from the procedure.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. SAMPLE.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-PC.
OBJECT-COMPUTER. IBM-PC.

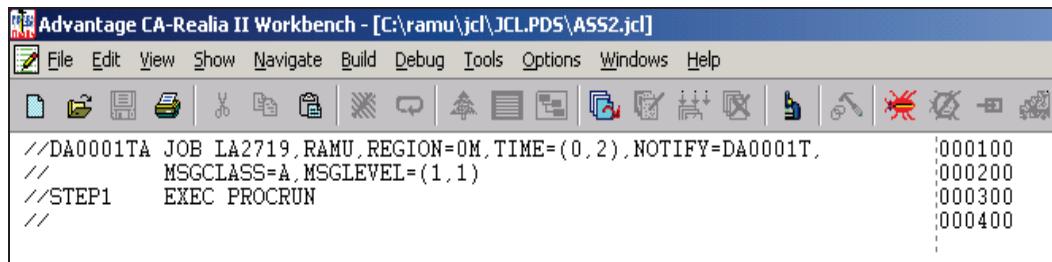
DATA DIVISION.
WORKING-STORAGE SECTION.
01 W01-DUMMY PIC 9(02) VALUE ZEROS.

PROCEDURE DIVISION.
0000-MAIN.
DISPLAY 'MY FIRST COBOL PROGRAM'.
ACCEPT W01-DUMMY.

```

STOP RUN.

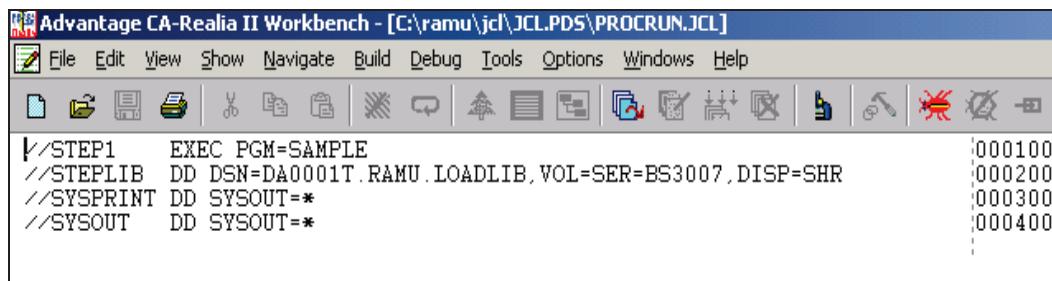
Step 2: Create a cataloged procedure and store it in the same **JCL.pds**.



```
//DA0001TA JOB LA2719, RAMU, REGION=0M, TIME=(0,2), NOTIFY=DA0001T,      000100
//          MSGCLASS=A, MSGLEVEL=(1,1)           000200
//STEP1    EXEC PROCRUN                   000300
//                                         000400
```

Figure 97: Cataloged procedure

Step 3: Create a JCL which invokes the above cataloged procedure and store it in the same **JCL.PDS**.



```
!STEP1    EXEC PGM=SAMPLE             000100
//STEPLIB  DD DSN=DA0001T.RAMU, LOADLIB, VOL=SER=BS3007, DISP=SHR 000200
//SYSPRINT DD SYSOUT=*                 000300
//SYSOUT   DD SYSOUT=*                 000400
```

Figure 98: Run JCL that executes the Cataloged procedure

Step 4: Select **Build → options → program → new** and make the entries as given in the screen mentioned below.

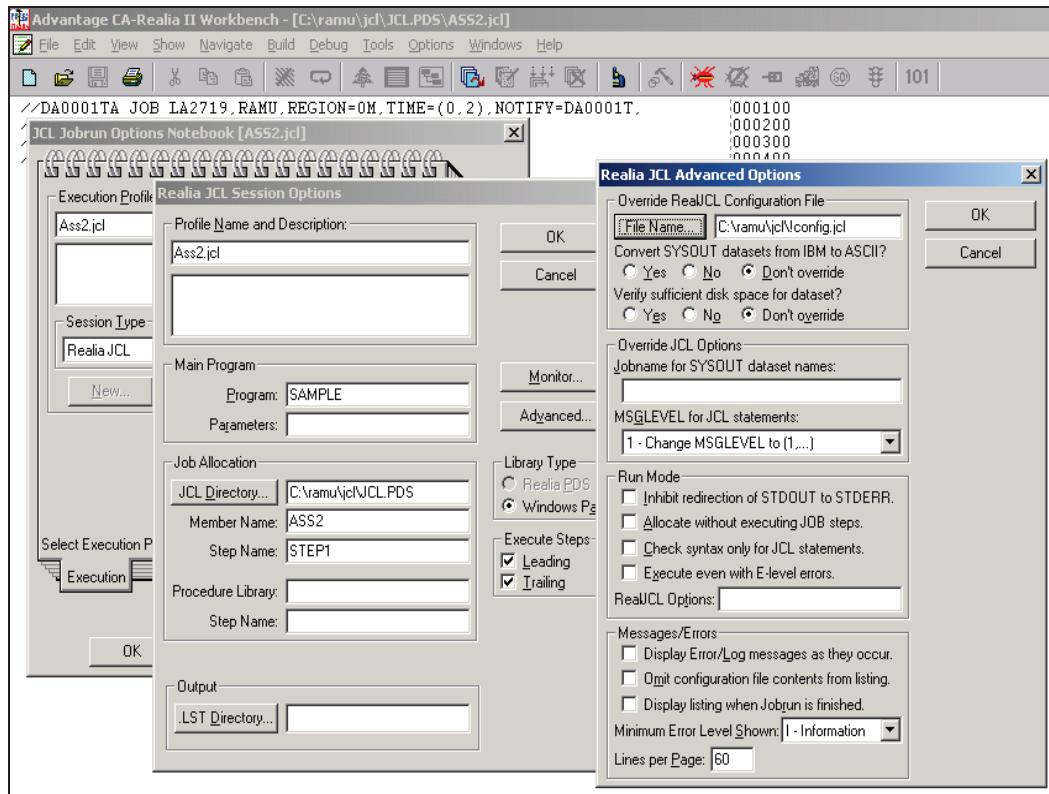
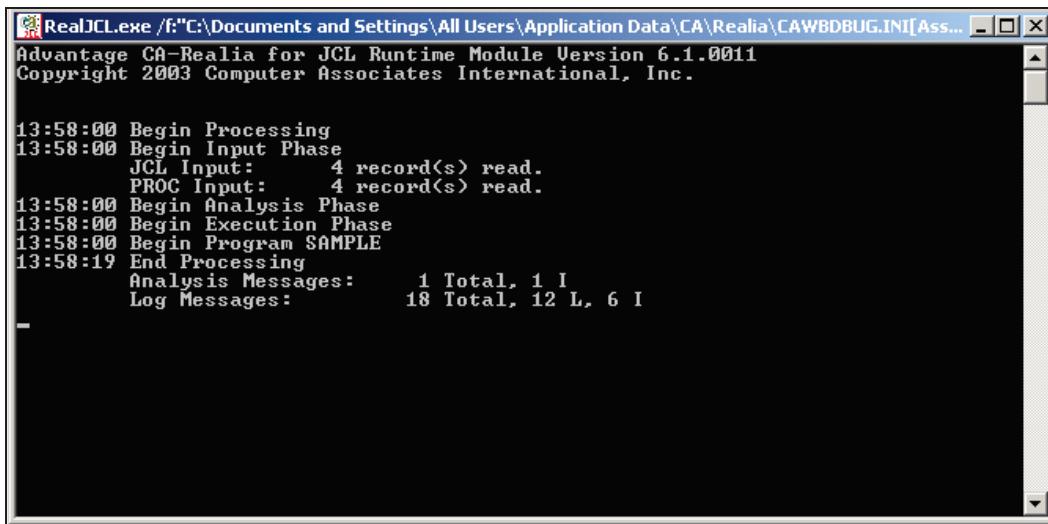


Figure 99: Creation of new profile for execution of cataloged procedure

Step 5: Execute the JCL. The output will be displayed as follows:



```

RealJCL.exe /F:"C:\Documents and Settings\All Users\Application Data\CA\Realia\CAWBDBUG.INI[Ass...
Advantage CA-Realia for JCL Runtime Module Version 6.1.0011
Copyright 2003 Computer Associates International, Inc.

13:58:00 Begin Processing
13:58:00 Begin Input Phase
    JCL Input:      4 record(s) read.
    PROC Input:     4 record(s) read.
13:58:00 Begin Analysis Phase
13:58:00 Begin Execution Phase
13:58:00 Begin Program SAMPLE
13:58:19 End Processing
    Analysis Messages:      1 Total, 1 I
    Log Messages:           18 Total, 12 L, 6 I
-
```

Figure 100: After Execution of Run JCL

To see the message printed by the COBOL program, open the **JES** folder in the same path, that is, **C:\ramu\jcl** and find the folder with the name as same as your job name. Open the folder with the job name and you can find two text files starting with the character that ends in your job name. One of the files will contain the message printed by the COBOL program (that is SYSOUT) and the other file is for SYSPRINT which would not contain any output.

Note: If you have only one SYSOUT statement in your run JCL, then only one text file will be created in the folder and that contains the output message.

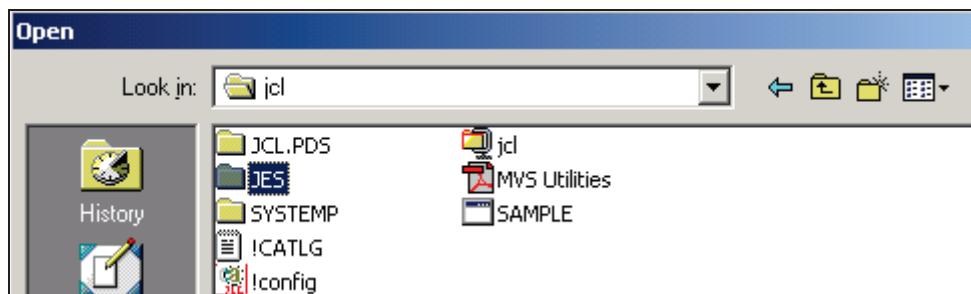


Figure 101: Open JES folder



Figure 102: Open the folder named with your job name

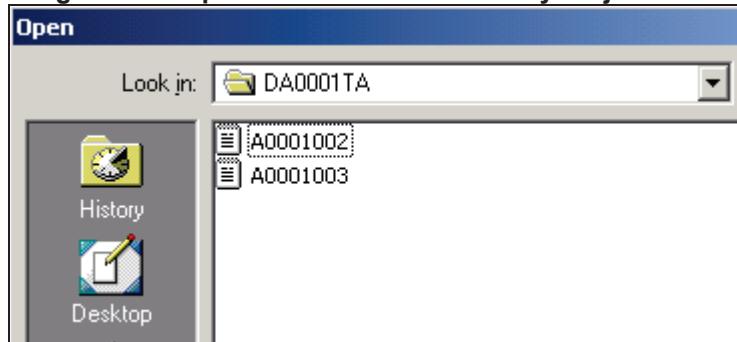


Figure 103: Two text files – First one for SYSPRINT and Second for SYSOUT

Note: This may differ according to the order you code these two statements.

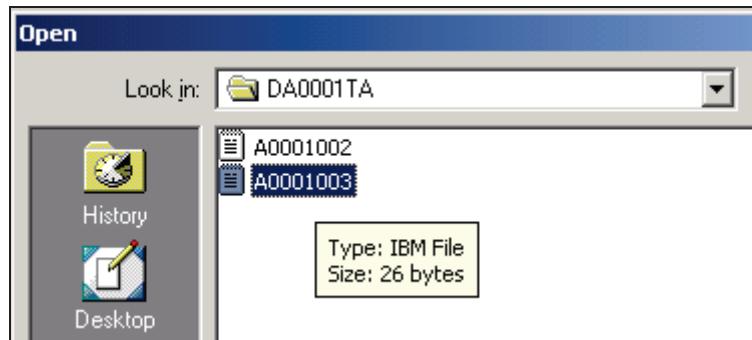


Figure 104: Open the SYSOUT text file for the output message

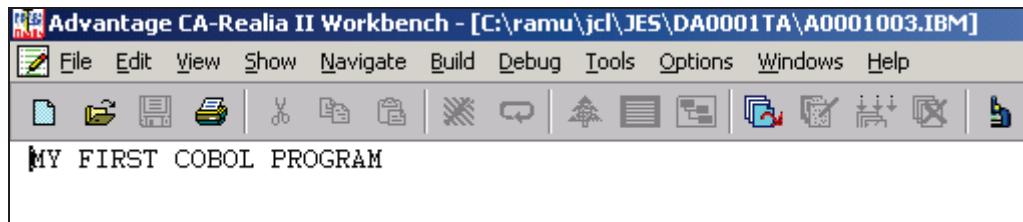
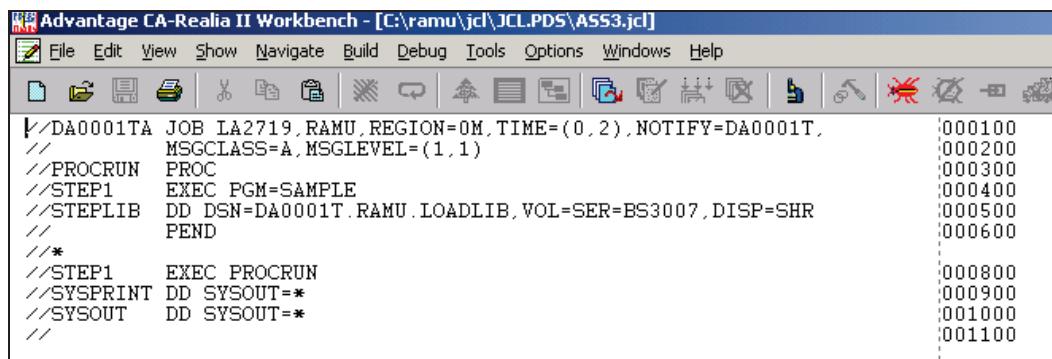


Figure 105: Output message printed by the COBOL program named SAMPLE

B) Instream procedure:

Step 1: Create an **instream** procedure and store it in your **jcl.pds**.


A screenshot of the Advantage CA-Realia II Workbench interface showing JCL code. The code defines a job step DA0001TA with various parameters and steps. It includes a PROCRUN step, an EXEC PGM=SAMPLE step, and a STEPLIB DD DSN=DA0001T.RAMU.LOADLIB step. It also includes SYSPRINT and SYSOUT DD statements. The right side of the screen shows a column of binary values (000100, 000200, etc.) corresponding to the JCL statements.

```

//DA0001TA JOB LA2719, RAMU, REGION=0M, TIME=(0,2), NOTIFY=DA0001T,      000100
//          MSGCLASS=A, MSGLEVEL=(1,1)                                     000200
//PROCRUN  PROC                                         000300
//STEP1    EXEC PGM=SAMPLE                                      000400
//STEPLIB  DD DSN=DA0001T.RAMU.LOADLIB, VOL=SER=BS3007, DISP=SHR   000500
//          PEND                                         000600
//*
//STEP1    EXEC PROCRUN                                     000800
//SYSPRINT DD SYSOUT=*                                     000900
//SYSOUT   DD SYSOUT=*                                     001000
//                                                 001100

```

Figure 106: Instream procedure

Step 2: Select **Build → options → program → new** and make the entries as given in the screen mentioned below.

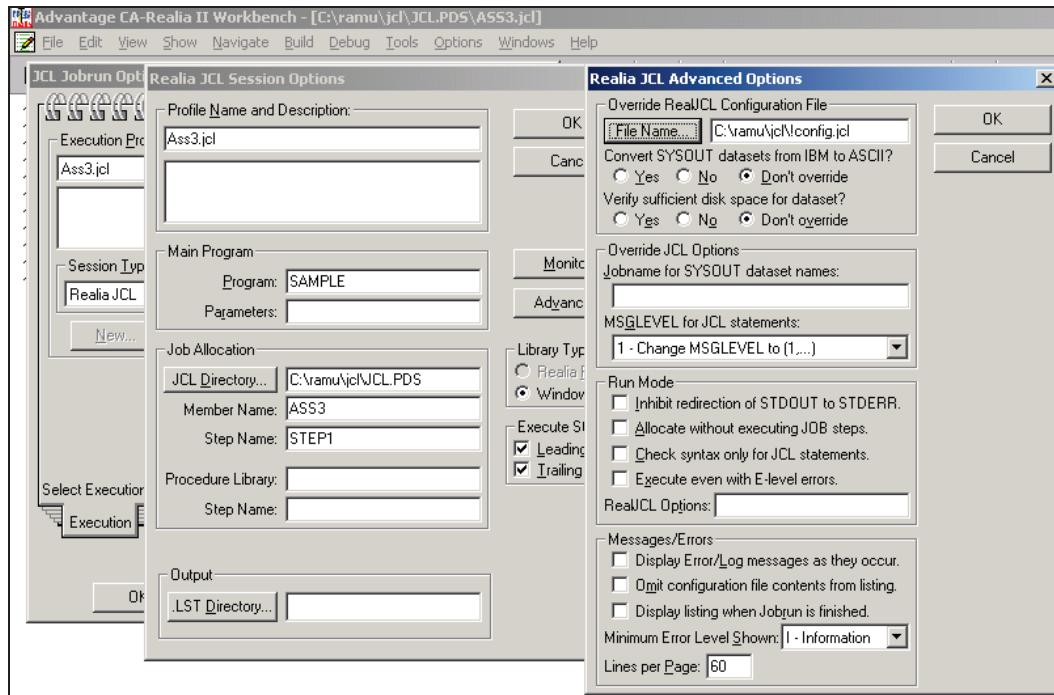
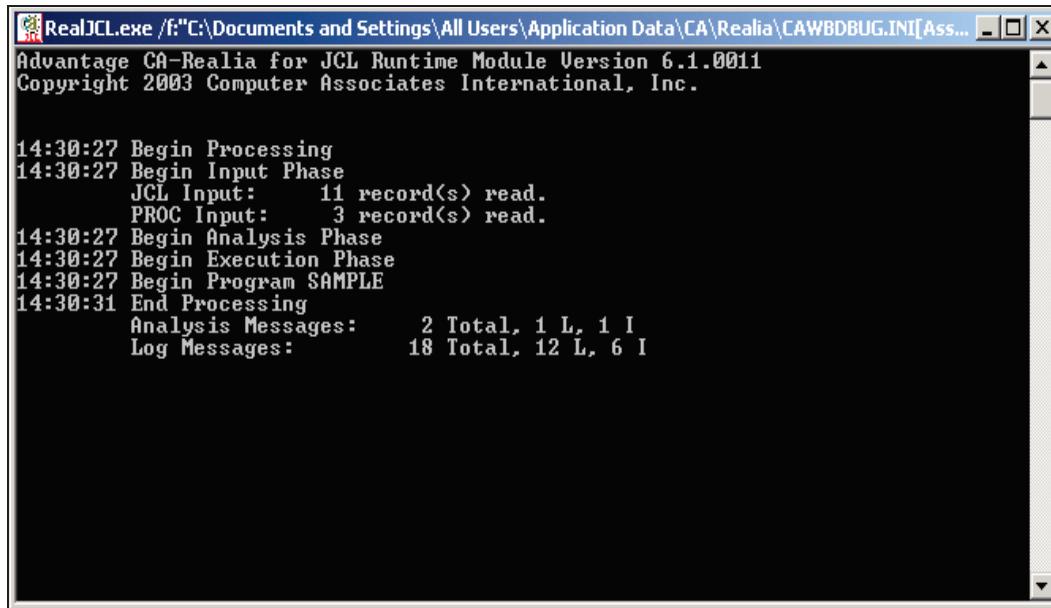


Figure 107: Creation of new profile for execution of instream procedure

Step 3: Execute the JCL. The output will be as shown below:



```
RealJCL.exe /f:"C:\Documents and Settings\All Users\Application Data\CA\Realia\CAWBDBUG.INI[Ass...
Advantage CA-Realia for JCL Runtime Module Version 6.1.0011
Copyright 2003 Computer Associates International, Inc.

14:30:27 Begin Processing
14:30:27 Begin Input Phase
    JCL Input:    11 record(s) read.
    PROC Input:   3 record(s) read.
14:30:27 Begin Analysis Phase
14:30:27 Begin Execution Phase
14:30:27 Begin Program SAMPLE
14:30:31 End Processing
    Analysis Messages:    2 Total, 1 L, 1 I
    Log Messages:        18 Total, 12 L, 6 I
```

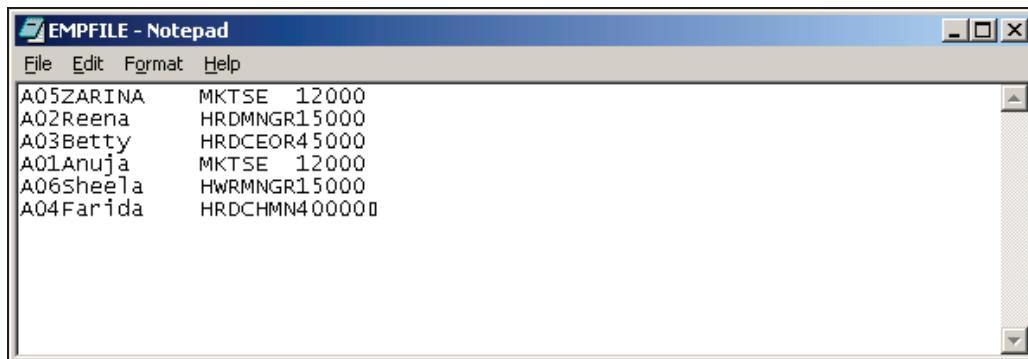
Figure 108: Execution of Run JCL that invokes the instream procedure

To see the message printed by the COBOL program, follow the same procedure shown in **Fig: 4.14** to **Fig: 4.18**.

Sorting:

Step 1: Create a file through **IEFBR14** (Ref. Page no. 28) or by executing a COBOL program.

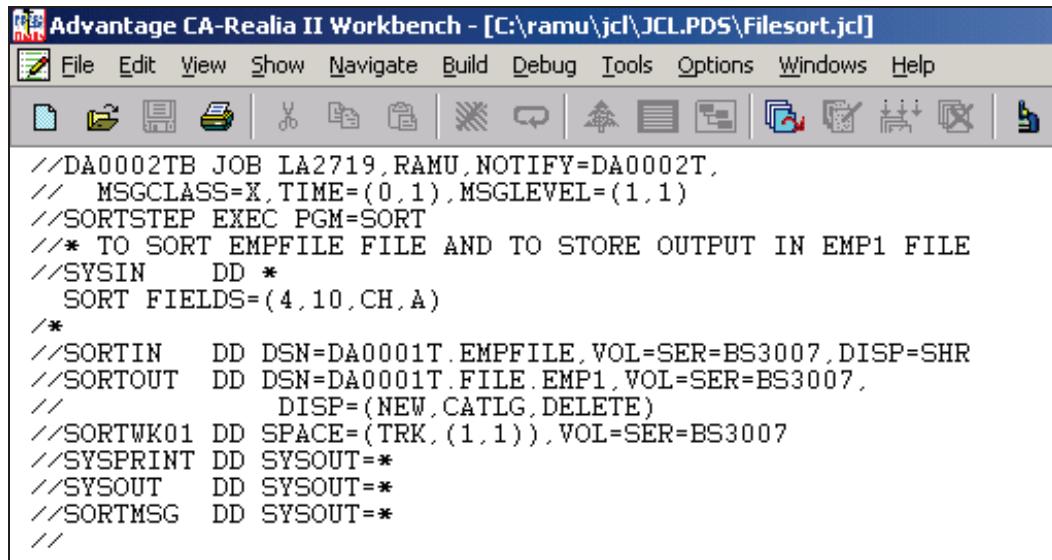
Step 2: Store the following records in this file. This is the file which will be sorted. Store the file in **JCL.PDS**.



EMPFILE - Notepad		
File	Edit	Format
A05ZARINA	MKTSE	12000
A02Reena	HRDMngr1	5000
A03Betty	HRDCEOR4	5000
A01Anuja	MKTSE	12000
A06Sheela	HWRMngr1	5000
A04Farida	HRDCHMN4	00000

Figure 109: Input file for Sorting

Step 3: Create the following JCL and store it in **jcl.pds**.



```
//DA0002TB JOB LA2719, RAMU, NOTIFY=DA0002T,
//  MSGCLASS=X, TIME=(0,1), MSGLEVEL=(1,1)
//SORTSTEP EXEC PGM=SORT
//* TO SORT EMPFILE FILE AND TO STORE OUTPUT IN EMP1 FILE
//SYSIN    DD *
   SORT FIELDS=(4,10,CH,A)
/*
//SORTIN    DD DSN=DA0001T.EMPFILE, VOL=SER=BS3007, DISP=SHR
//SORTOUT   DD DSN=DA0001T.FILE.EMP1, VOL=SER=BS3007,
//            DISP=(NEW,CATLG,DELETE)
//SORTWK01  DD SPACE=(TRK,(1,1)), VOL=SER=BS3007
//SYSPRINT  DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//SORTMSG   DD SYSOUT=*
//
```

Figure 110: Code to sort EMPFILE and to store the sorted output in EMP1 file

Step 4: Select **Build → options → program → new** and make the entries as given in the screen shown below:

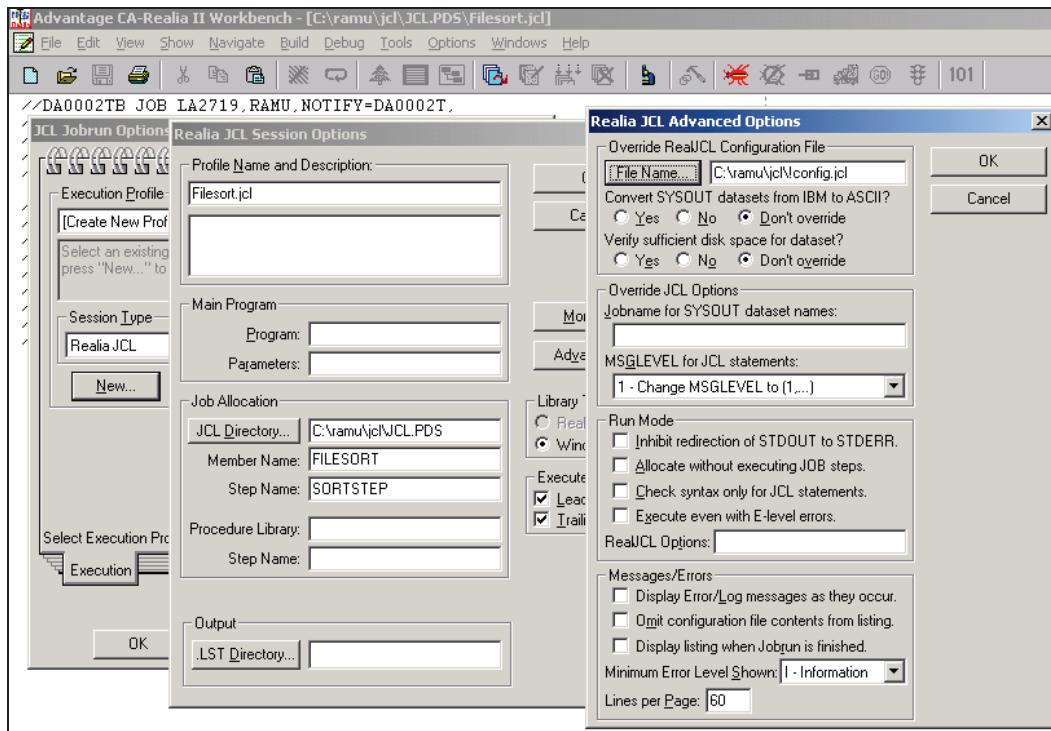


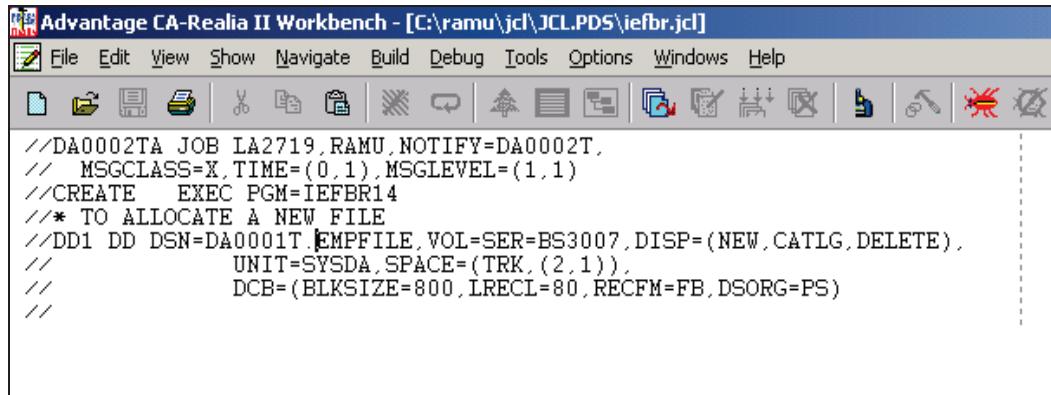
Figure 111: Creation of new profile for FILESORT run JCL

Step 5: Execute the JCL. A file named **EMP1** gets created in your **JCL.PDS**. This file is sorted on the employee name.

The contents of EMP1 are as shown below:

Advantage CA-Realia II Workbench - [C:\ramu\jcl\JCL.PDS\EMP1.IBM]		
File Edit View Show Navigate Build Debug Tools Options Windows Help		
A01Anuja	MKTSE	12000
A03Betty	HRDCEOR	45000
A04Farida	HRDCHMN	40000
A02Reena	HRDMNGR	15000
A06Sheela	HWRMNGR	15000
A05ZARINA	MKTSE	12000

Figure 112: Contents of EMP1 file

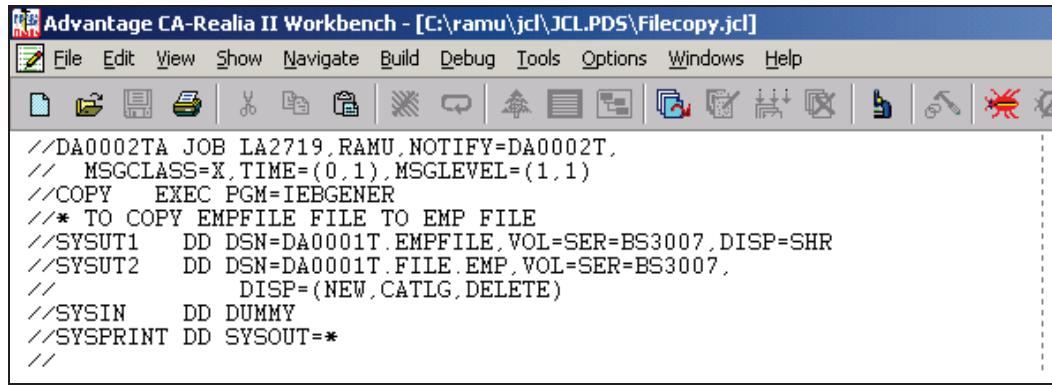
Creating a file through IEFBR14:**Step 1:** Write the following program in **jcl.pds**.

```
//DA0002TA JOB LA2719, RAMU, NOTIFY=DA0002T,
// MSGCLASS=X, TIME=(0,1), MSGLEVEL=(1,1)
//CREATE EXEC PGM=IEFBR14
//** TO ALLOCATE A NEW FILE
//DD1 DD DSN=DA0001T.EMPFILE, VOL=SER=BS3007, DISP=(NEW,CATLG,DELETE),
//          UNIT=SYSDA, SPACE=(TRK,(2,1)),
//          DCB=(BLKSIZE=800,LRECL=80,RECFM=FB,DSORG=PS)
//
```

Figure 113: Code to create sequential file named EMPFILE in JCL.PDS**Step 2:** Execute the JCL. This will create a file by the name **EMPFILE** in your **JCL.PDS**. Also, The corresponding entry for the file is also made in the **!catlg.ibm**. Since **DA0001T.EMPFILE** is mapped to your **JCL.PDS** in customized **!config.jcl**, this file will be created in your **JCL.PDS** folder.

Copying a file through IEBGENER:

Step 1: Write the following program.



```
//DA0002TA JOB LA2719, RAMU, NOTIFY=DA0002T,  
// MSGCLASS=X, TIME=(0,1), MSGLEVEL=(1,1)  
//COPY EXEC PGM=IEBGENER  
//* TO COPY EMPFILE FILE TO EMP FILE  
//SYSUT1 DD DSN=DA0001T.EMPFILE, VOL=SER=BS3007, DISP=SHR  
//SYSUT2 DD DSN=DA0001T.FILE.EMP, VOL=SER=BS3007,  
// DISP=(NEW,CATLG,DELETE)  
//SYSIN DD DUMMY  
//SYSPRINT DD SYSOUT=*  
//
```

Figure 114: Code to copy one file to another file in JCL.PDS

Step 2: Execute the JCL. This will create a file by the name **EMP** in your **JCL.PDS** since dataset name **DA0001T.FILE** is mapped to **JCL.PDS**. Also, the corresponding entry for the file is also made in **!catlg.ibm**.

The contents of **!catlg.ibm** are as shown below:

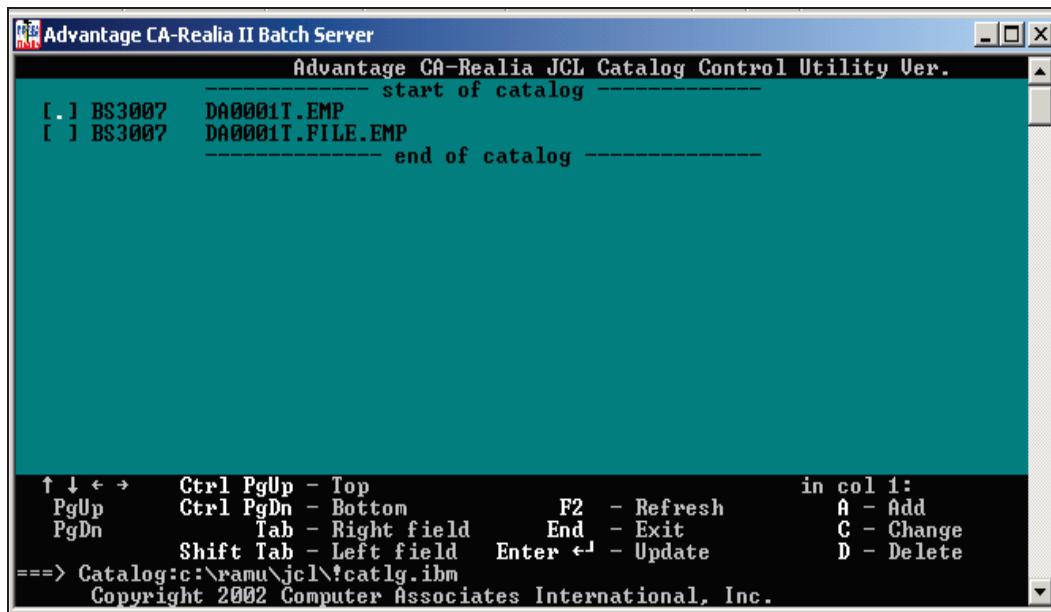


Figure 115: Catalog contents

The output message of the **FILECOPY.JCL** is stored in the **JES** folder under the directory named with the job name, **DA0002TA** as shown below.

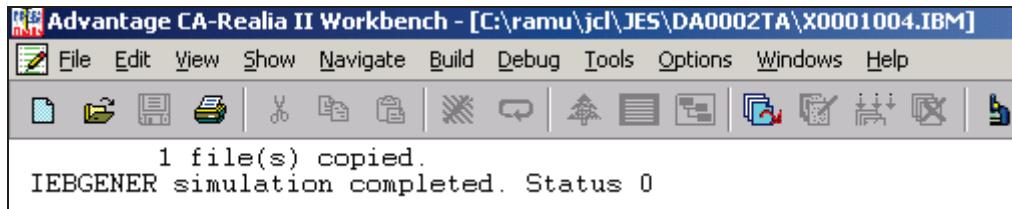


Figure 116: Output Message of IEBGENER after COPY operatio

Operators Procedure to enquire the details of tape using CA1

Use Operators Procedure to enquire the details of tape using CA1

Solution:

Step 1: Get tape number or name of the dataset.

Procedure to get tape volume number:

Enter 3.4 at ISPF Command Line.

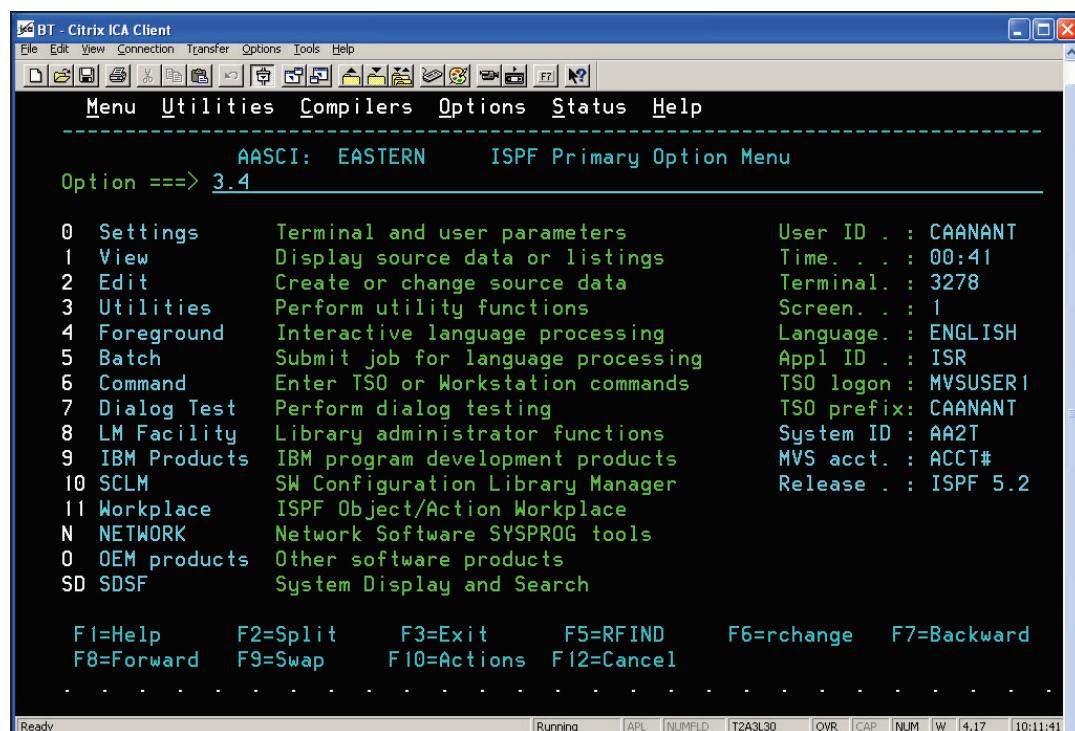
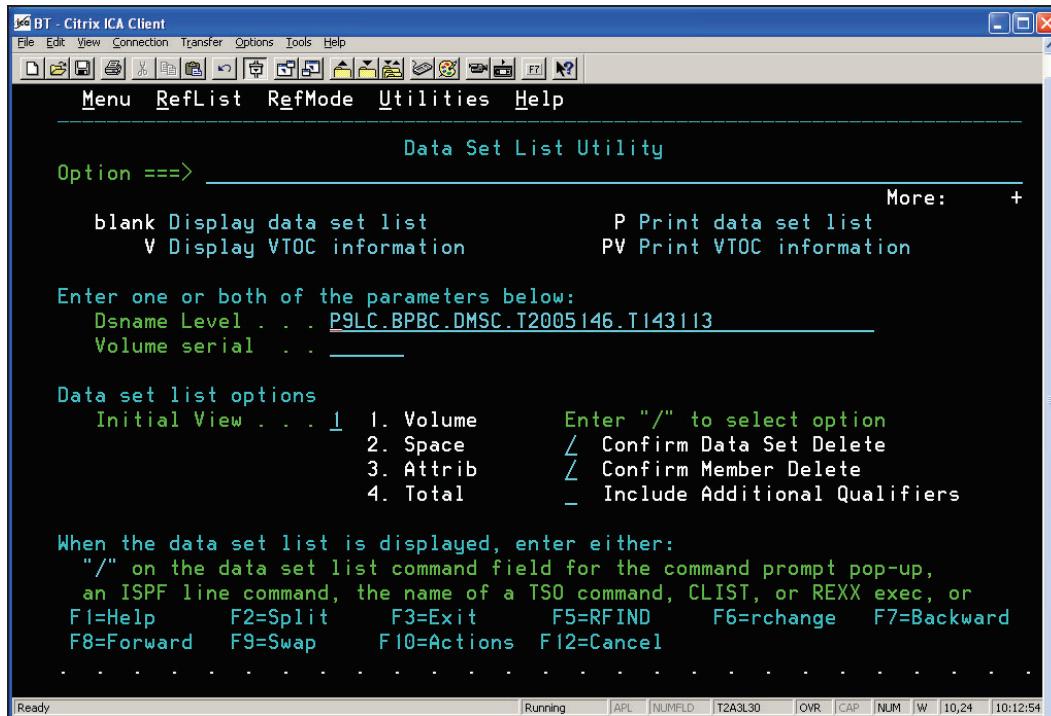


Figure 23: Output

Step 2: Type Dataset Name and press **ENTER**.



```

BT - Citrix ICA Client
File Edit View Connection Transfer Options Tools Help
Menu RefList RefMode Utilities Help

Data Set List Utility
Option ==> _____ More: +
blank Display data set list          P Print data set list
V Display VTOC information         PV Print VTOC information

Enter one or both of the parameters below:
Dsname Level . . . P9LC.BPBC.DMSC.T2005146.T143113
Volume serial . . .

Data set list options
Initial View . . . 1 1. Volume      Enter "/" to select option
                  2. Space        / Confirm Data Set Delete
                  3. Attrib        / Confirm Member Delete
                  4. Total         _ Include Additional Qualifiers

When the data set list is displayed, enter either:
"/" on the data set list command field for the command prompt pop-up,
an ISPF line command, the name of a TSO command, CLIST, or REXX exec, or
F1=Help   F2=Split   F3=Exit    F5=RFIND   F6=rchange  F7=Backward
F8=Forward  F9=Swap   F10=Actions F12=Cancel

```

Figure 24: Output

Step3: Note down the **Tape volume Number**.

Note: If it shows + next to volume number, then it indicates that Dataset is on multiple Tapes. Volume number is Numeric for tapes.

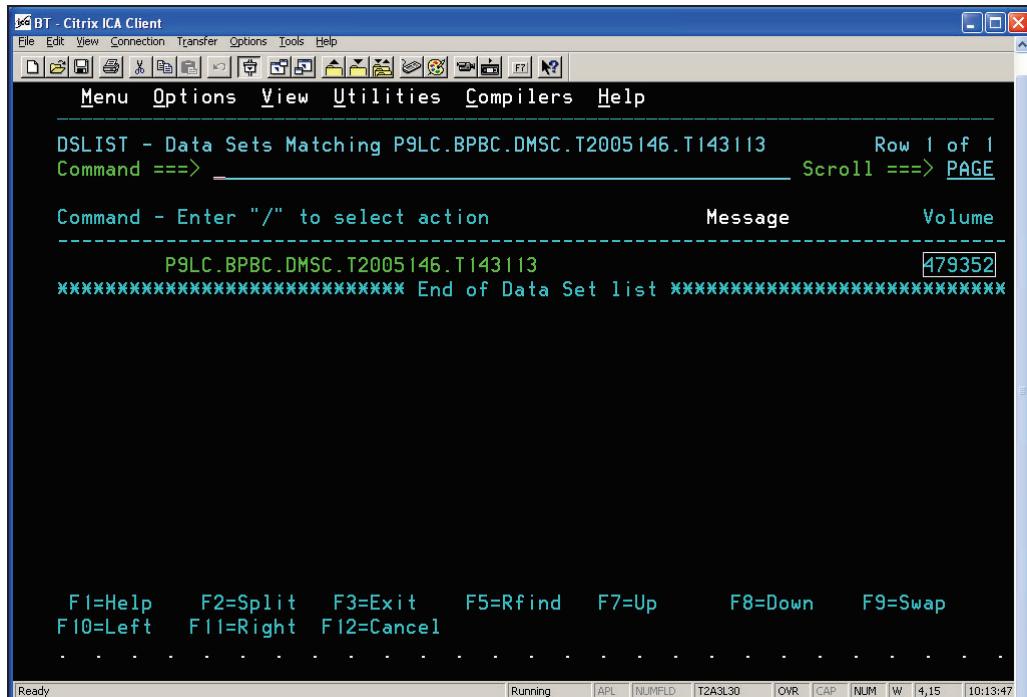


Figure 25: Output

Step 4: Get tape details using **CA1**. Type **CA1** and press **ENTER**

Note: Find out where the **CA1** option exists for your application. Sometimes you see this option in **ISPF** menu directly, sometimes residing as part of other products option.

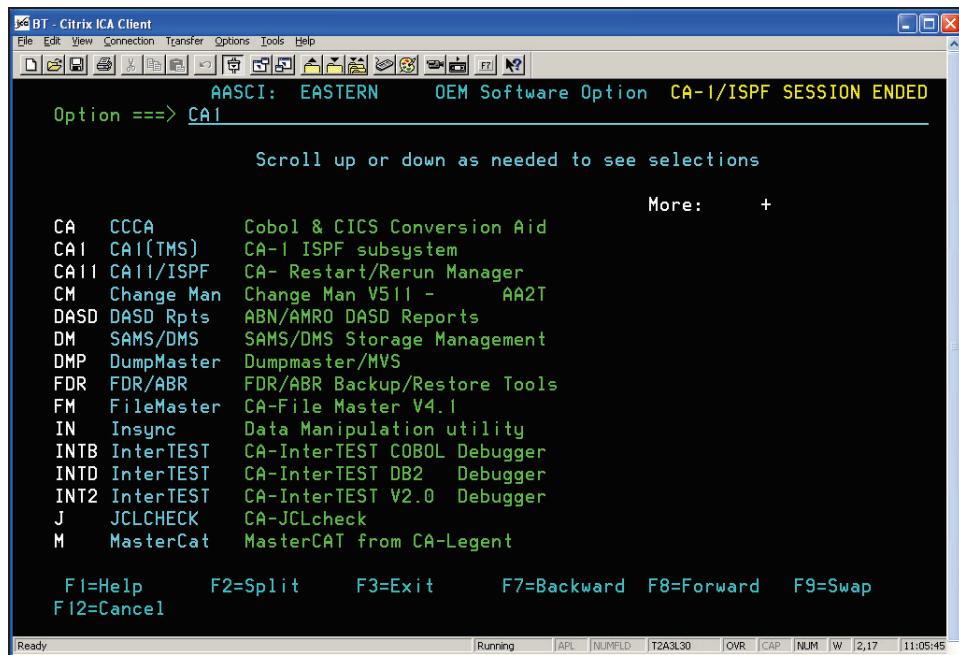


Figure 26: Output

Step 5: Type 1 and press **ENTER**.

Note: Password is required if dataset is secured by CA1. Otherwise default is blank for the password.

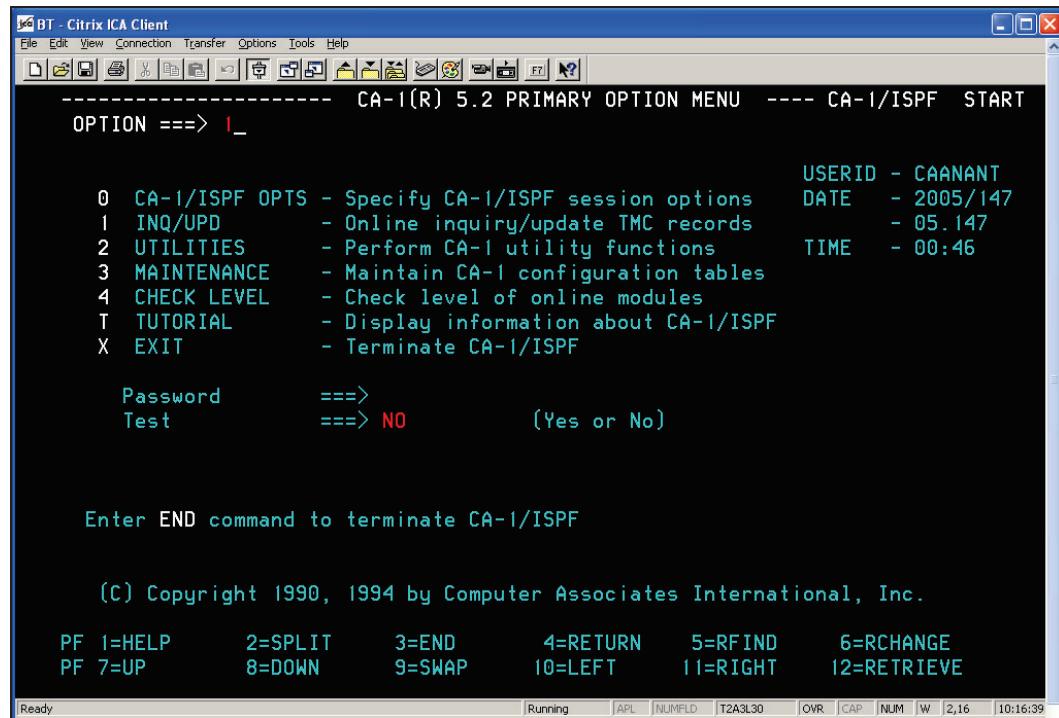
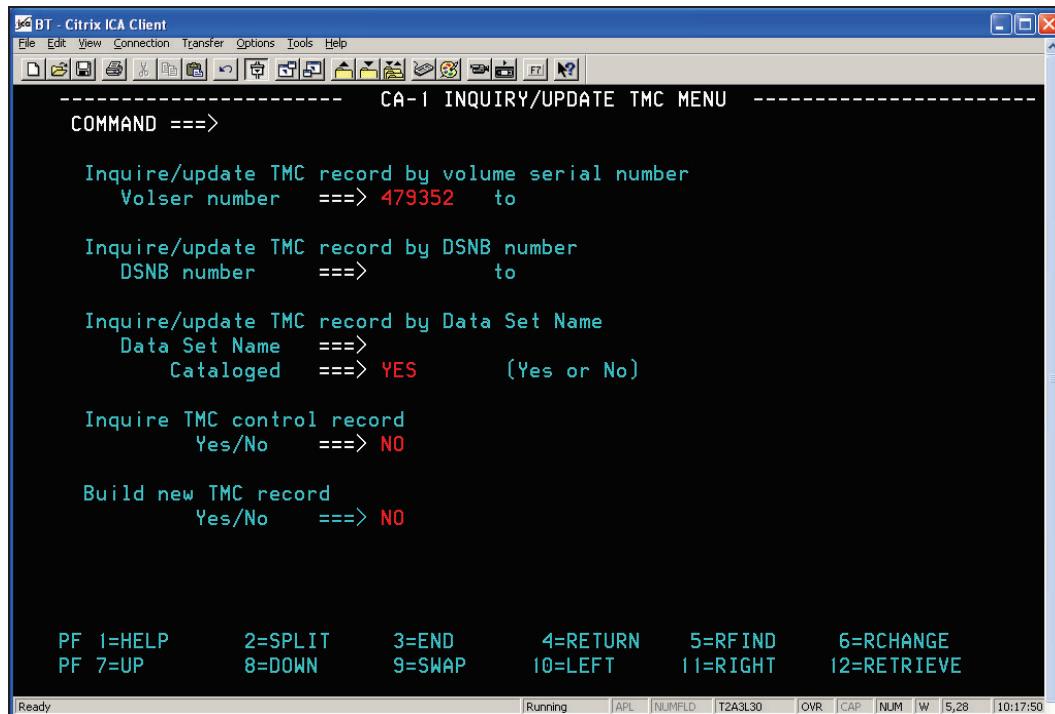


Figure 27: Output

Step6: Get tape details using CA1.

If you have volume number of the tape from Step1, enter here.



```

----- CA-1 INQUIRY/UPDATE TMC MENU -----
COMMAND ==>

Inquire/update TMC record by volume serial number
Volser number ==> 479352 to

Inquire/update TMC record by DSNB number
DSNB number ==> to

Inquire/update TMC record by Data Set Name
Data Set Name ==>
Cataloged ==> YES (Yes or No)

Inquire TMC control record
Yes/No ==> NO

Build new TMC record
Yes/No ==> NO

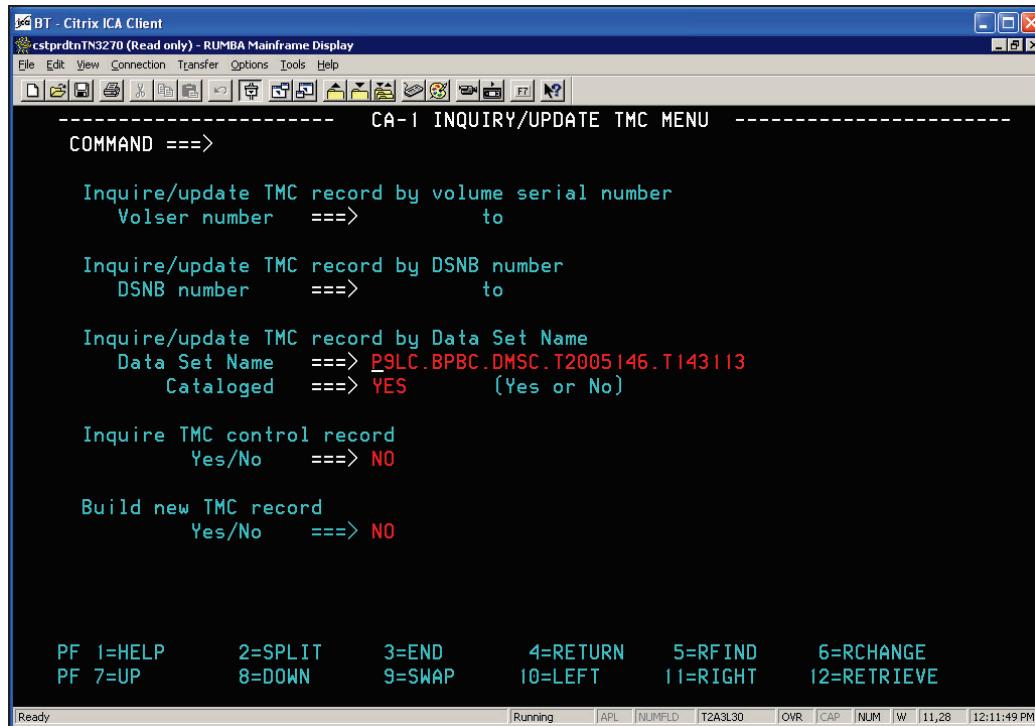
PF 1=HELP      2=SPLIT      3=END       4=RETURN     5=RIND      6=RCHANGE
PF 7=UP        8=DOWN       9=SWAP      10=LEFT     11=RIGHT    12=RETRIEVE

```

Figure 28: Output

Step7: Get tape details using CA1.

If you do not have volume number, then you can enter data set name here.



```

BT - Citrix ICA Client
cstrprtnTN3270 (Read only) - RUMBA Mainframe Display
File Edit View Connection Transfer Options Tools Help
----- CA-1 INQUIRY/UPDATE TMC MENU -----
COMMAND ==>

Inquire/update TMC record by volume serial number
Volser number ==> to

Inquire/update TMC record by DSNB number
DSNB number ==> to

Inquire/update TMC record by Data Set Name
Data Set Name ==> P9LC.BPBC.DMSC.T2005146.T143113
Cataloged ==> YES (Yes or No)

Inquire TMC control record
Yes/No ==> NO

Build new TMC record
Yes/No ==> NO

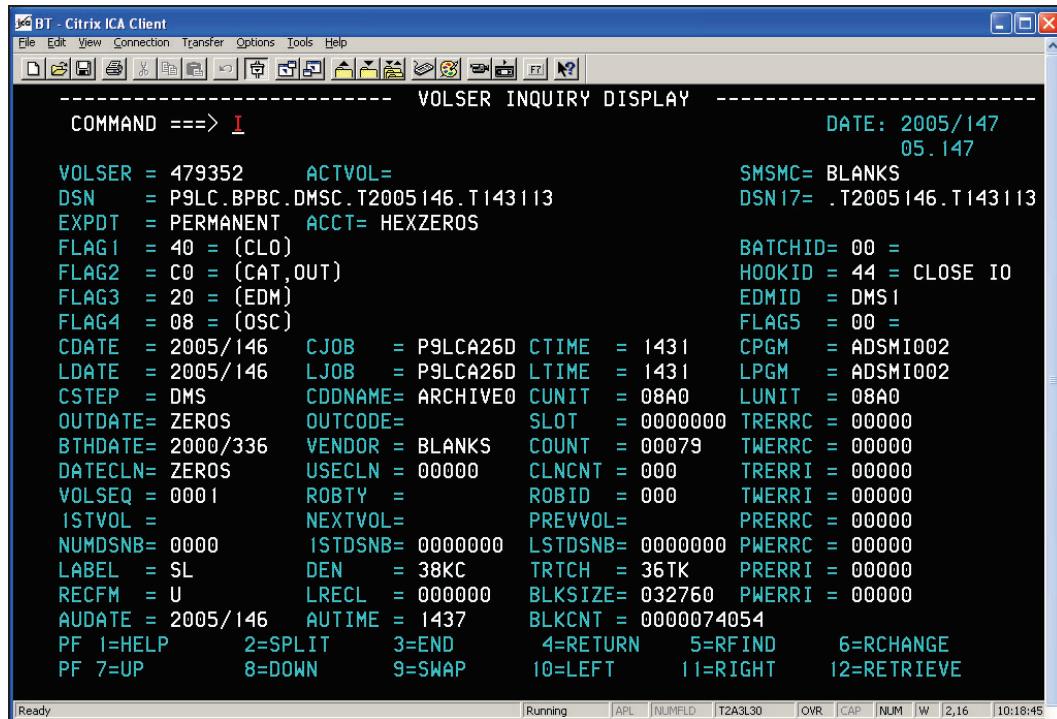
PF 1=HELP      2=SPLIT      3=END       4=RETURN     5=RFIND      6=RCHANGE
PF 7=UP        8=DOWN       9=SWAP      10=LEFT      11=RIGHT     12=RETRIEVE

```

Figure 29: Output

Step 8: Get tape details using CA1.

Note: Please see next panels to see the details of some important fields. Rest of the details can be found by pressing **PF1** when you are at CA1 screen.



```

----- VOLSER INQUIRY DISPLAY -----
COMMAND ==> I                               DATE: 2005/147
                                                05.147
VOLSER = 479352    ACTVOL=                 SMSMC= BLANKS
DSN   = P9LC.BPBC.DMSC.T2005146.T143113  DSN17= .T2005146.T143113
EXPDT = PERMANENT ACCT= HEXZEROS
FLAG1 = 40 = (CLO)                         BATCHID= 00 =
FLAG2 = C0 = (CAT,OUT)                      HOOKID = 44 = CLOSE IO
FLAG3 = 20 = (EDM)                          EDMID  = DMS1
FLAG4 = 08 = (OSC)                          FLAG5 = 00 =
CDATE = 2005/146   CJOB   = P9LCA26D CTIME = 1431   CPGM   = ADSMI002
LDATE = 2005/146   LJOB   = P9LCA26D LTIME = 1431   LPGM   = ADSMI002
CSTEP  = DMS     CDDNAME= ARCHIVE0 CUNIT = 08A0   LUNIT = 08A0
OUTDATE= ZEROS  OUTCODE=                  SLOT   = 0000000 TRERRC = 00000
BTHDATE= 2000/336 VENDOR = BLANKS   COUNT  = 00079  TWERRC = 00000
DATECLN= ZEROS  USECLN = 00000   CLNCNT = 000   TRERRI = 00000
VOLSEQ = 0001    ROBTY =                   ROBID  = 000   TWERRI = 00000
ISTVOL =          NEXTVOL=                  PREVVOL= PRERRC = 00000
NUMDSNB= 0000    1STDSNB= 0000000 LSTDNB= 0000000 PWERRC = 00000
LABEL  = SL      DEN    = 38KC    TRTCN = 36TK    PRERRI = 00000
RECFM  = U       LRECL  = 000000  BLKSIZE= 032760  PWERRI = 00000
AUDATE = 2005/146 AUTIME = 1437   BLKCNT = 0000074054
PF 1=HELP    2=SPLIT   3=END     4=RETURN   5=RFIND    6=RCHANGE
PF 7=UP      8=DOWN    9=SWAP    10=LEFT    11=RIGHT   12=RETRIEVE

```

Figure 30: Output

Step 9: Get tape details using CA1

VOLSER Volume serial number

DSN Data set name

EXPDT Expiration date

(**Note:** This date can be changed by using Utilities option 2 in CA1 but authorization is required to do this.)

CJOB Creating job name

CDATE Creation date

CTIME Creation time

CSTEP Creating step name

CPGM Creating Program

LJOB Name of the last job to use the volume

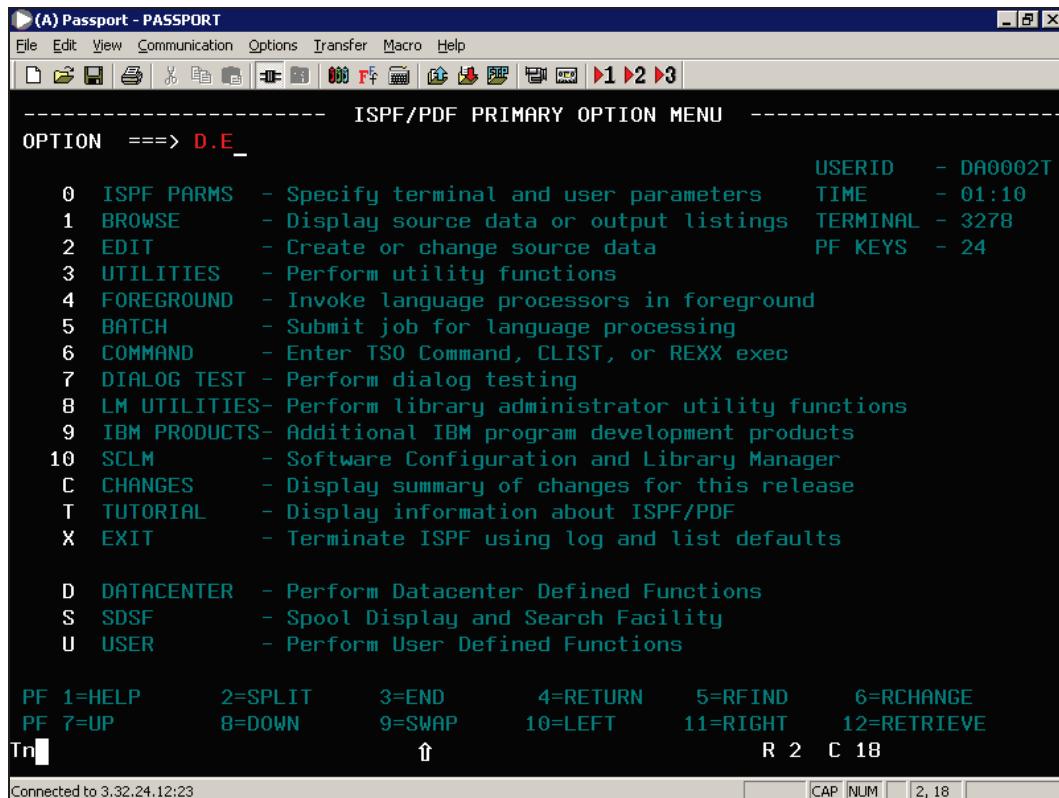
LPGM Last Used Program

CDDNAME	DD name of the creating step
TRTCH	Recording technique (tracks)
RECFM	Record format
LRECL	Logical record length
BLKSIZE	Block size
BLKCNT	Number of blocks
1STVOL	First volume serial of a multi-volume data set
NEXTVOL	Next volume serial of a multi-volume data set
(Note: 1ST VOL and NEXTVOL will be populated if the dataset is stored in multiple tapes. In this case, other volume details can also be found by entering "V" and pressing ENTER on this screen).	
VOLSEQ	Volume sequence

Xpeditor

Solution:

Step 1: To logon to Xpediter, type **D.E.**



```

(A) Passport - PASSPORT
File Edit View Communication Options Transfer Macro Help
[Icons] 1 2 3
----- ISPF/PDF PRIMARY OPTION MENU -----
OPTION ===> D.E_
0 ISPF PARMS - Specify terminal and user parameters      USERID - DA0002T
1 BROWSE - Display source data or output listings        TIME   - 01:10
2 EDIT - Create or change source data                   TERMINAL - 3278
3 UTILITIES - Perform utility functions
4 FOREGROUND - Invoke language processors in foreground
5 BATCH - Submit job for language processing
6 COMMAND - Enter TSO Command, CLIST, or REXX exec
7 DIALOG TEST - Perform dialog testing
8 LM UTILITIES - Perform library administrator utility functions
9 IBM PRODUCTS - Additional IBM program development products
10 SCLM - Software Configuration and Library Manager
C CHANGES - Display summary of changes for this release
T TUTORIAL - Display information about ISPF/PDF
X EXIT - Terminate ISPF using log and list defaults

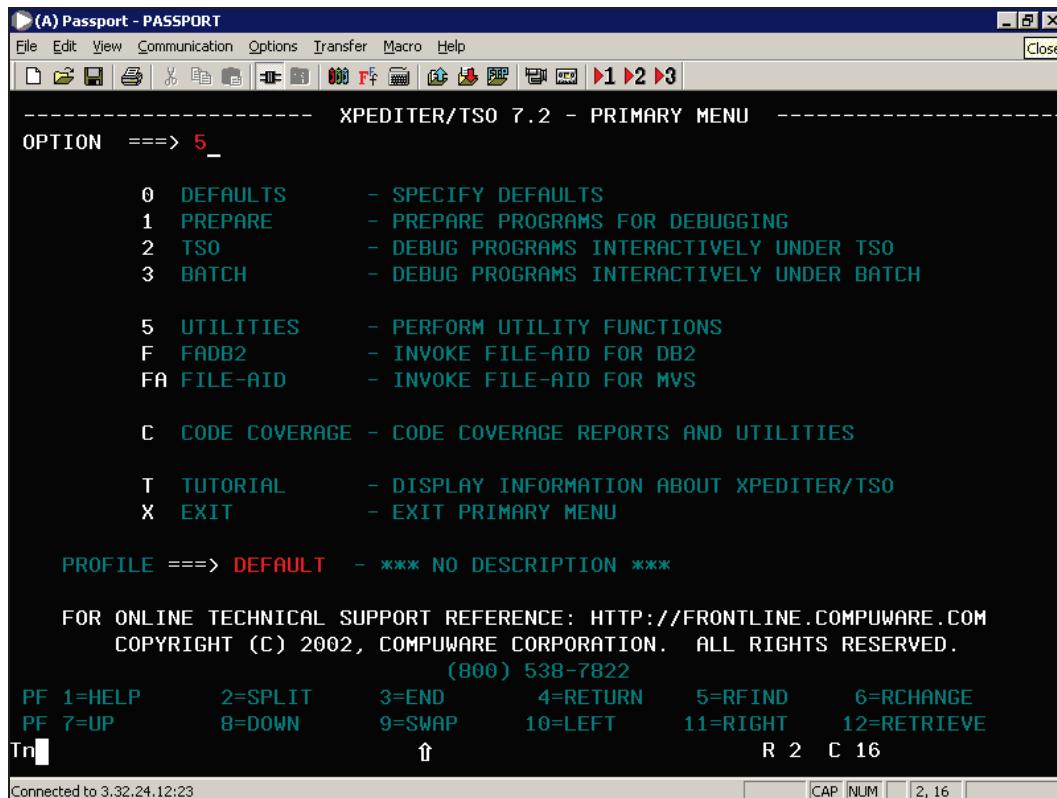
D DATACENTER - Perform Datacenter Defined Functions
S SDSF - Spool Display and Search Facility
U USER - Perform User Defined Functions

PF 1=HELP      2=SPLIT      3=END       4=RETURN     5=RFIND      6=RCHANGE
PF 7=UP        8=DOWN       9=SWAP      10=LEFT      11=RIGHT    12=RETRIEVE
Tn [ ]          ↑           R 2 C 18
Connected to 3.32.24.12:23
[CAP NUM] [2, 18]

```

Figure 31: Output

Step 2: Type 5, to enter the utilities menu.



```

(A) Passport - PASSPORT
File Edit View Communication Options Transfer Macro Help
Close
----- XPEDITER/TSO 7.2 - PRIMARY MENU -----
OPTION ===> 5

  0  DEFAULTS      - SPECIFY DEFAULTS
  1  PREPARE       - PREPARE PROGRAMS FOR DEBUGGING
  2  TSO            - DEBUG PROGRAMS INTERACTIVELY UNDER TSO
  3  BATCH          - DEBUG PROGRAMS INTERACTIVELY UNDER BATCH

  5  UTILITIES     - PERFORM UTILITY FUNCTIONS
F  FADB2          - INVOKE FILE-AID FOR DB2
FA FILE-AID       - INVOKE FILE-AID FOR MVS

  C  CODE COVERAGE - CODE COVERAGE REPORTS AND UTILITIES

  T  TUTORIAL      - DISPLAY INFORMATION ABOUT XPEDITER/TSO
  X  EXIT           - EXIT PRIMARY MENU

PROFILE ==> DEFAULT - *** NO DESCRIPTION ***

FOR ONLINE TECHNICAL SUPPORT REFERENCE: HTTP://FRONTLINE.COMPUWARE.COM
COPYRIGHT (C) 2002, COMPUWARE CORPORATION. ALL RIGHTS RESERVED.
(800) 538-7822
PF 1=HELP          2=SPLIT          3=END            4=RETURN         5=RFIND          6=RCHANGE
PF 7=UP            8=DOWN           9=SWAP           10=LEFT          11=RIGHT         12=RETRIEVE
Tr                         ↑                           R 2   C 16
Connected to 3.32.24.12:23
CAP NUM 2, 16

```

Figure 32: Output

Step 3: Create a **DDIO** file. This is a one-time job. Once a **DDIO** file is created, then you need not create it every time you use **Xpediter**. Type **3** in the **OPTION** to enter the **DDIO** file facility.

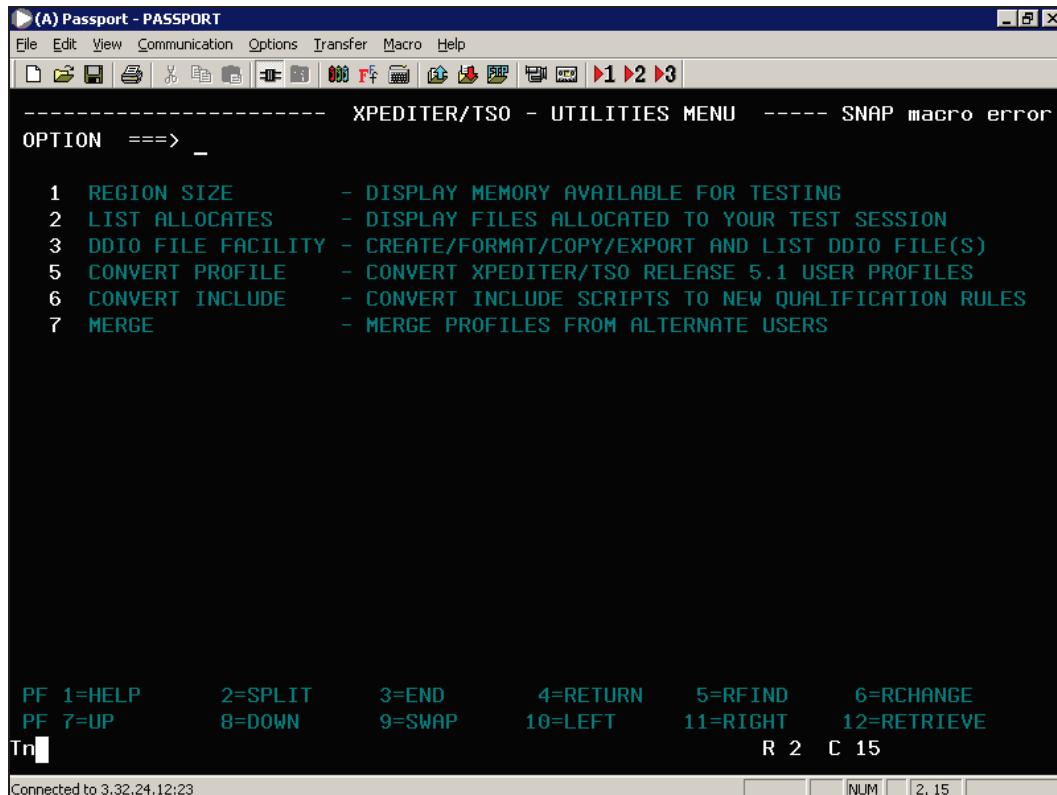
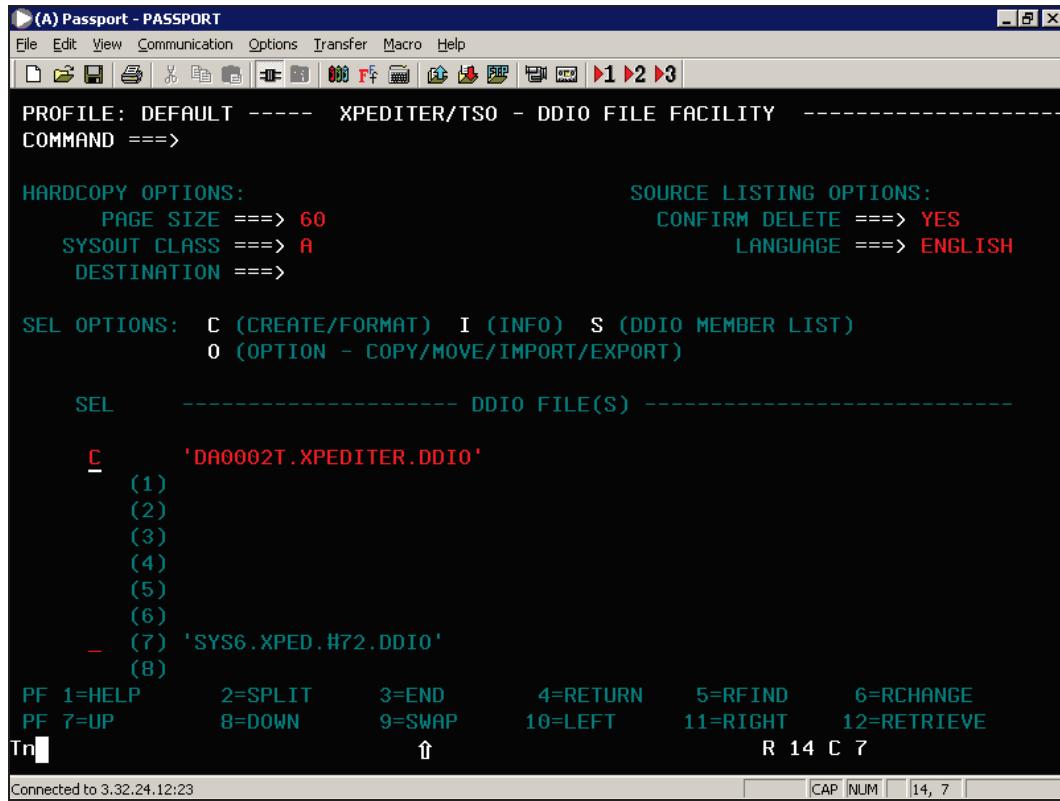


Figure 33: Output

Step 4: Type **C** to create a file. Also type the name of the **DDIO** file in single quotes.



```

(A) Passport - PASSPORT
File Edit View Communication Options Transfer Macro Help
[Icons] PROFILE: DEFAULT ----- XPEDITER/TSO - DDIO FILE FACILITY -----
COMMAND ===>

HARDCOPY OPTIONS:                               SOURCE LISTING OPTIONS:
    PAGE SIZE ===> 60                         CONFIRM DELETE ===> YES
    SYSOUT CLASS ===> A                        LANGUAGE ===> ENGLISH
    DESTINATION ===>

SEL OPTIONS:  C (CREATE/FORMAT)  I (INFO)  S (DDIO MEMBER LIST)
              0 (OPTION - COPY/MOVE/IMPORT/EXPORT)

SEL      ----- DDIO FILE(S) -----
C      'DA0002T.XPEDITER.DDIO'
(1)
(2)
(3)
(4)
(5)
(6)
- (7) 'SYS6.XPED.#72.DDIO'
(8)

PF 1=HELP      2=SPLIT      3=END      4=RETURN      5=RFIND      6=RCHANGE
PF 7=UP        8=DOWN       9=SWAP      10=LEFT       11=RIGHT     12=RETRIEVE
Tr [ ]          ↑           R 14 C 7

Connected to 3.32.24.12:23
[CAP NUM] [14, 7]

```

Figure 34: Output

Step 5: Change the preparation to **BATCH** and File Type to **VSAM**. Give the appropriate values for **Space Units**, **Primary Quantity**, and **Number of Members**. After specifying values for all the parameters hit **Enter** key. This will submit a job to create the **DDIO** file.

(A) Passport - PASSPORT

File Edit View Communication Options Transfer Macro Help

----- XPEDITOR/TSO - DDIO CREATE/FORMAT FACILITY -----

COMMAND ==>

COMMANDS: SETUP (SETUP PANEL) D (DELETE FILE)

DDIO FILE ==> 'DA0002T.XPEDITER.DDIO'

PREPARATION ==> BATCH (BATCH/EDITJCL/BACKGROUND)
FILE TYPE ==> VSAM (VSAM/SEQUENTIAL)

STORAGE CLASS ==> (OPTIONAL)
MANAGEMENT CLASS ==> (OPTIONAL)
DATA CLASS ==> (OPTIONAL)
VOLUME ==> (OPTIONAL)
UNIT ==> (OPTIONAL)

SPACE UNITS ==> TRACKS (BLOCKS/TRACKS/CYLINDERS)
PRIMARY QUANTITY ==> 10
NUMBER OF MEMBERS ==> 20_ (1 TO 32767 <= REPTCOUNT)

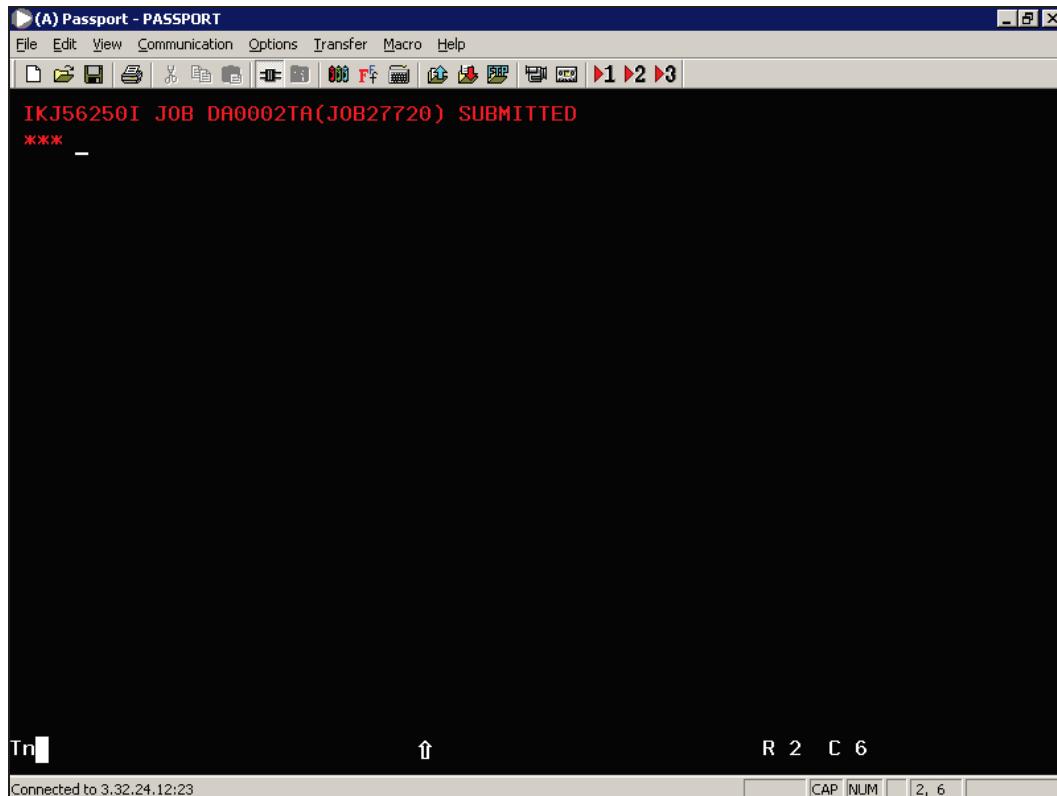
PF 1=HELP PF 7=UP 2=SPLIT B=DOWN 3=END 9=SWAP 4=RETURN 10=LEFT 5=RFINID 11=RIGHT 6=RCHANGE 12=RETRIEVE

Tn [] ↑ R 19 C 29

Connected to 3.32.24.12:23 CAP NUM 19, 29

Figure 35: Output

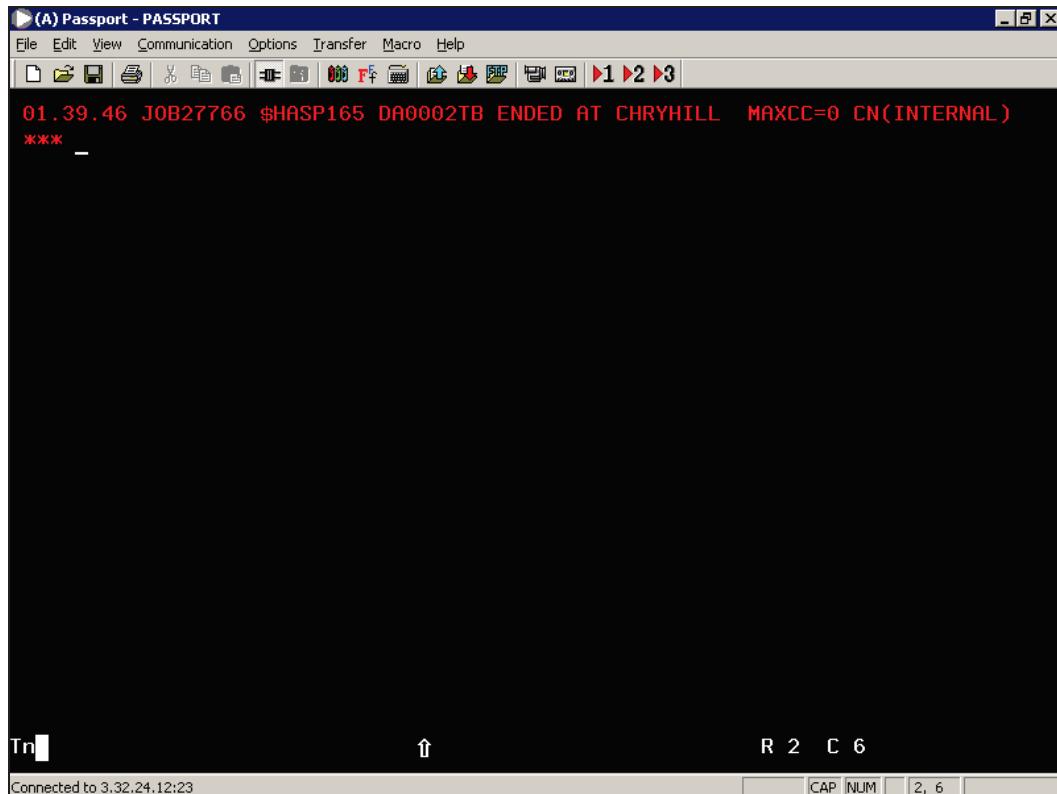
Step 6: Observe the output message saying that the job is submitted for creating the DDIO file.



```
IKJ56250I JOB DA0002TA(JOB27720) SUBMITTED
***-
```

Figure 36: Output

Step 7: Now observe the output message saying that the job has ended successfully and the DDIO file is created.



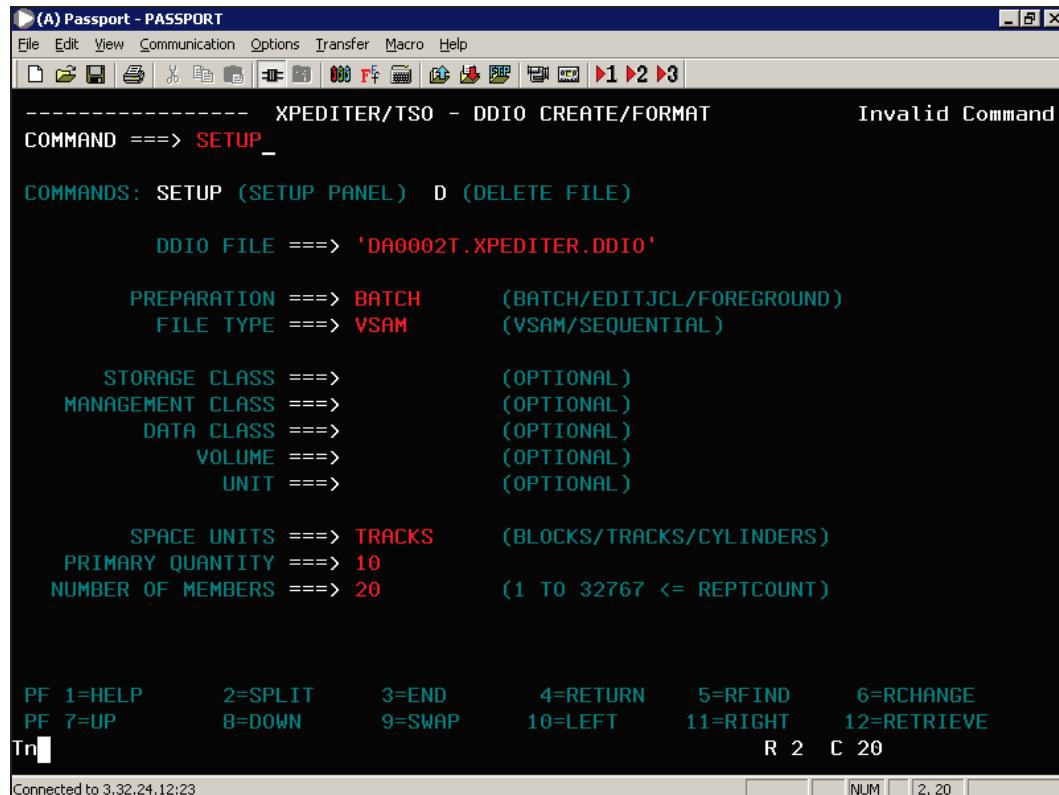
The screenshot shows a terminal window titled '(A) Passport - PASSPORT'. The menu bar includes File, Edit, View, Communication, Options, Transfer, Macro, and Help. The toolbar contains various icons for file operations like Open, Save, Print, and Cut/Copy/Paste. The main text area displays the following output message in red text:

```
01.39.46 JOB27766 $HASP165 DA0002TB ENDED AT CHRYHILL MAXCC=0 CN(INTERNAL)
*** -
```

At the bottom of the window, there are status indicators: 'Tr' on the left, an upward arrow in the center, 'R 2 C 6' on the right, and a footer bar showing 'Connected to 3.32.24.12:23' and a numeric keypad area with CAP, NUM, and 2, 6.

Figure 37: Output

Step 8: When you submit the job and if you get any JCL Error, you need to setup the Process. To do this setup, enter **SETUP** in the command prompt as shown in the figure given below.

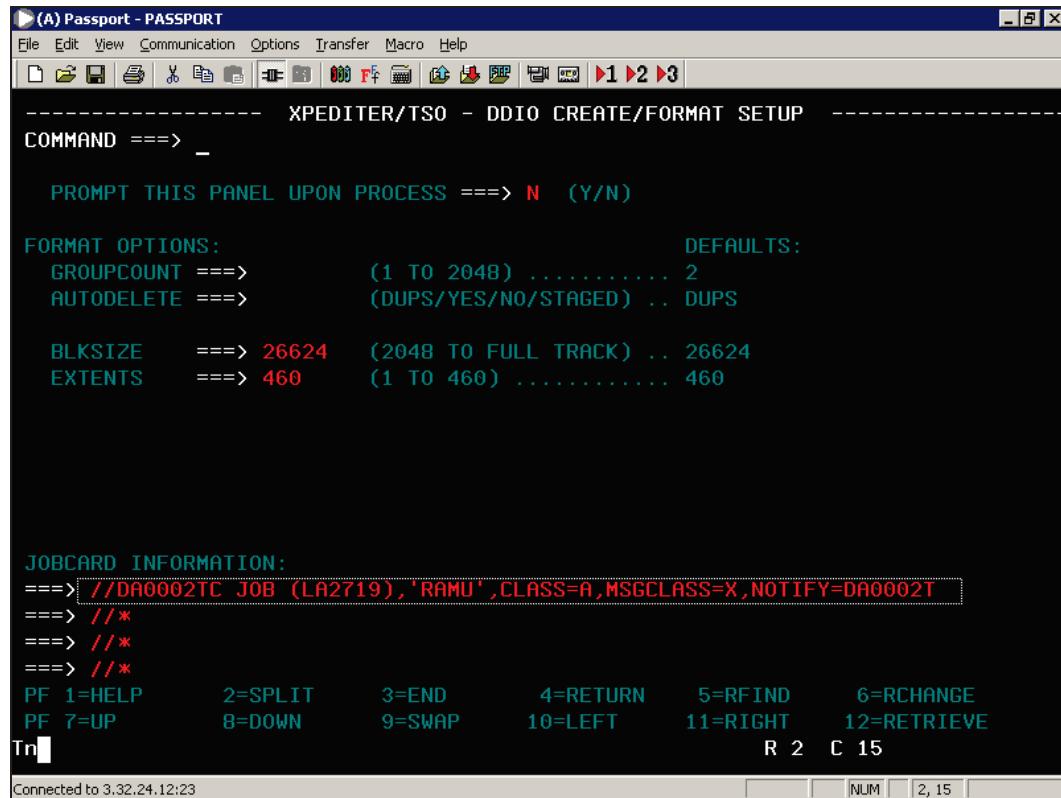


```
(A) Passport - PASSPORT
File Edit View Communication Options Transfer Macro Help
File Edit View Communication Options Transfer Macro Help
----- XPDITER/TSO - DDIO CREATE/FORMAT           Invalid Command
COMMAND ===> SETUP -
COMMANDS: SETUP (SETUP PANEL) D (DELETE FILE)
DDIO FILE ===> 'DA0002T.XPDITER.DDIO'
PREPARATION ===> BATCH      (BATCH/EDITJCL/BACKGROUND)
FILE TYPE ===> VSAM        (VSAM/SEQUENTIAL)
STORAGE CLASS ===>          (OPTIONAL)
MANAGEMENT CLASS ===>       (OPTIONAL)
DATA CLASS ===>             (OPTIONAL)
VOLUME ===>                 (OPTIONAL)
UNIT ===>                   (OPTIONAL)
SPACE UNITS ===> TRACKS    (BLOCKS/TRACKS/CYLINDERS)
PRIMARY QUANTITY ===> 10
NUMBER OF MEMBERS ===> 20   (1 TO 32767 <= REPTCOUNT)

PF 1=HELP      2=SPLIT      3=END       4=RETURN     5=RFIND      6=RCHANGE
PF 7=UP        8=DOWN       9=SWAP      10=LEFT      11=RIGHT     12=RETRIEVE
Fn [ ]          R 2          C 20
Connected to 3.32.24.12:23
```

Figure 38: Output

Step 9: Enter account information, name, class, msgclass, and so on under the **JOBCARD INFORMATION** for setting up the DDIO file creation job. This is also one time process and this has to be done only if you get any JCL error while you create the DDIO file.



```

(A) Passport - PASSPORT
File Edit View Communication Options Transfer Macro Help
[Icons] ----- XPEDITER/TSO - DDIO CREATE/FORMAT SETUP -----
COMMAND ===> _

PROMPT THIS PANEL UPON PROCESS ===> N (Y/N)

FORMAT OPTIONS:                               DEFAULTS:
GROUPCOUNT ===> (1 TO 2048) ..... 2
AUTODELETE ===> (DUPS/YES/NO/STAGED) .. DUPS

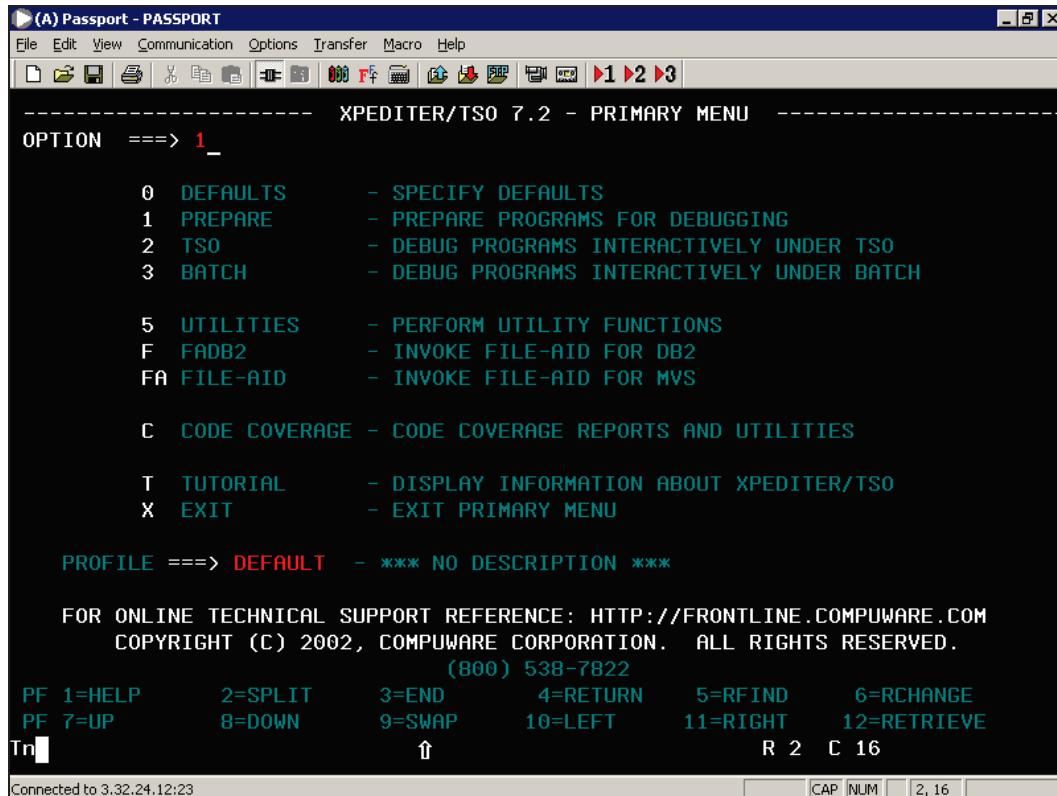
BLKSIZE    ===> 26624   (2048 TO FULL TRACK) ... 26624
EXTENTS    ===> 460      (1 TO 460) ..... 460

JOBCARD INFORMATION:
==> //DA0002TC JOB (LA2719),'RAMU',CLASS=A,MSGCLASS=X,NOTIFY=DA0002T
==> /**
==> /**
==> /**
PF 1=HELP     2=SPLIT     3=END      4=RETURN    5=RFIND     6=RCHANGE
PF 7=UP       8=DOWN      9=SWAP     10=LEFT     11=RIGHT    12=RETRIEVE
Tn[ ]                                     R 2 C 15
Connected to 3.32.24.12:23
[Buttons] [NUM] [2, 15]

```

Figure 39: Output

Step 10: Select 1 to prepare the program for debugging.



```

(A) Passport - PASSPORT
File Edit View Communication Options Transfer Macro Help
----- XPEDITER/TSO 7.2 - PRIMARY MENU -----
OPTION ===> 1

  0  DEFAULTS      - SPECIFY DEFAULTS
  1  PREPARE       - PREPARE PROGRAMS FOR DEBUGGING
  2  TSO           - DEBUG PROGRAMS INTERACTIVELY UNDER TSO
  3  BATCH          - DEBUG PROGRAMS INTERACTIVELY UNDER BATCH

  5  UTILITIES     - PERFORM UTILITY FUNCTIONS
F   FADB2         - INVOKE FILE-AID FOR DB2
FA  FILE-AID      - INVOKE FILE-AID FOR MVS

  C  CODE COVERAGE - CODE COVERAGE REPORTS AND UTILITIES

  T  TUTORIAL      - DISPLAY INFORMATION ABOUT XPEDITER/TSO
  X  EXIT          - EXIT PRIMARY MENU

PROFILE ==> DEFAULT - *** NO DESCRIPTION ***

FOR ONLINE TECHNICAL SUPPORT REFERENCE: HTTP://FRONTLINE.COMPUWARE.COM
COPYRIGHT (C) 2002, COMPUWARE CORPORATION. ALL RIGHTS RESERVED.
(800) 538-7822
PF 1=HELP        2=SPLIT      3=END        4=RETURN     5=RFIND      6=RCHANGE
PF 7=UP          8=DOWN       9=SWAP       10=LEFT      11=RIGHT    12=RETRIEVE
Tr[ ]                         ↑                         R 2  C 16
Connected to 3.32.24.12:23

```

Figure 40: Output

Step 11: Select 1 to convert the **Compile JCL**.

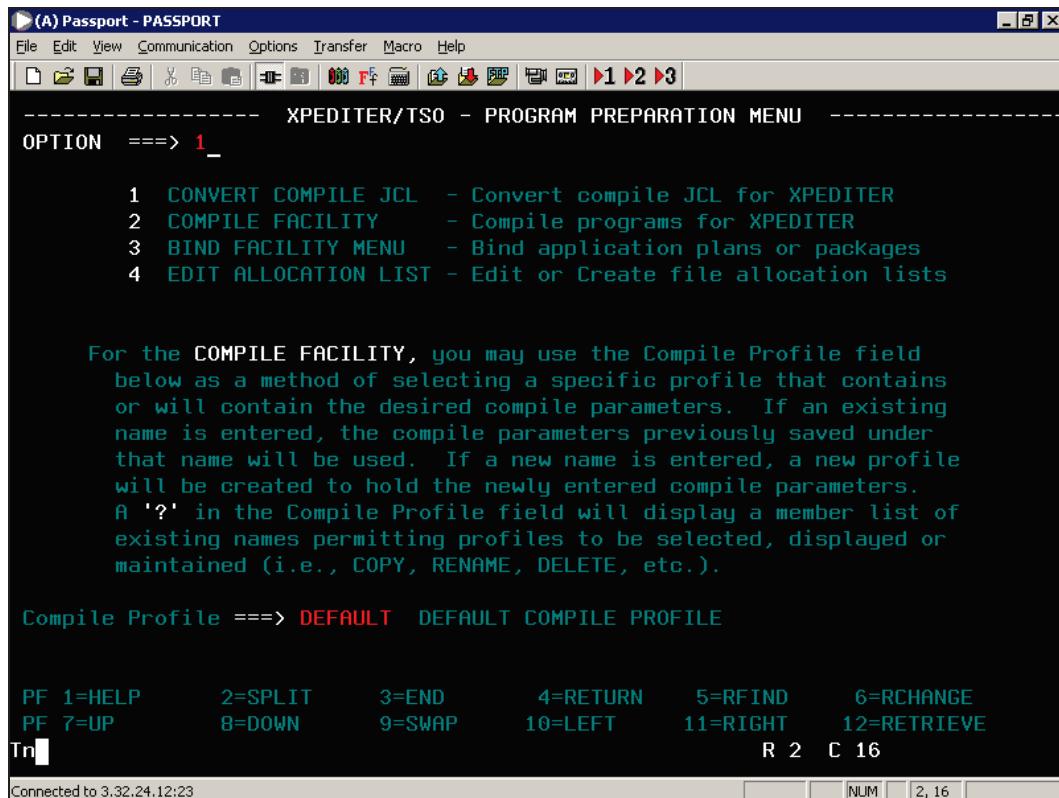
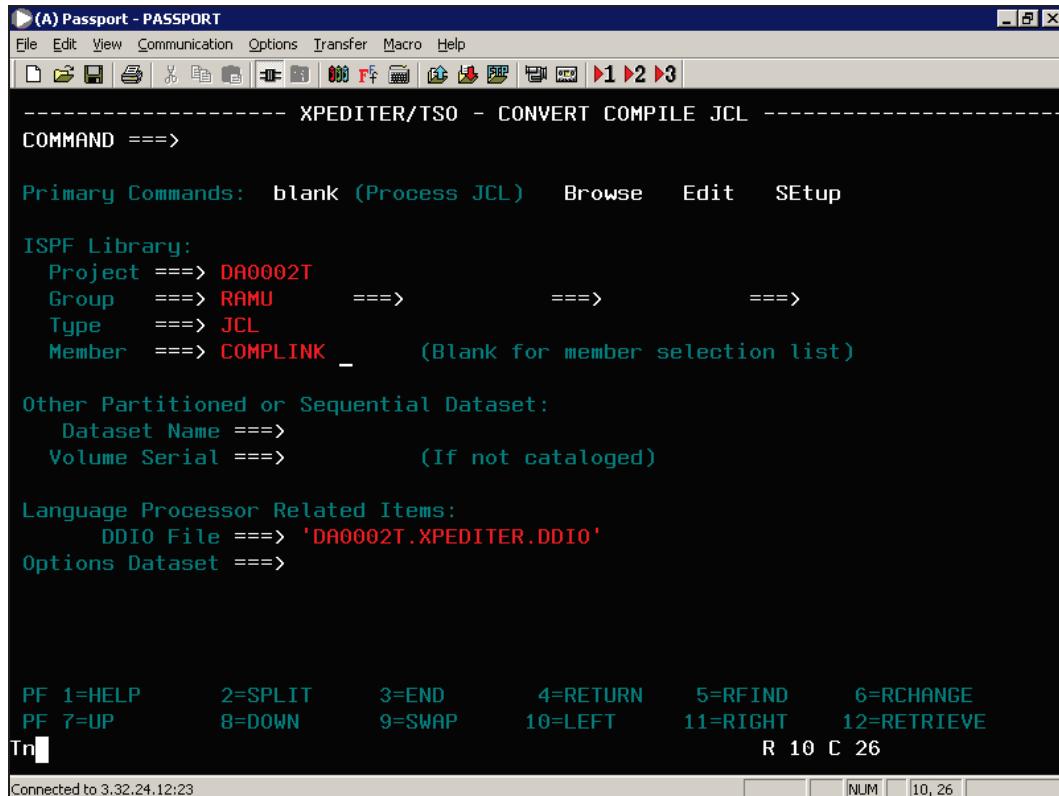


Figure 41: Output

Step 12: Enter the Member, DDIO file name, and so on and press **ENTER**.



```

(A) Passport - PASSPORT
File Edit View Communication Options Transfer Macro Help
----- XPDITER/TSO - CONVERT COMPILE JCL -----
COMMAND ===>

Primary Commands: blank (Process JCL) Browse Edit SSetup

ISPF Library:
Project ===> DA0002T
Group ===> RAMU ===> ===> ===>
Type ===> JCL
Member ===> COMPLINK - (Blank for member selection list)

Other Partitioned or Sequential Dataset:
Dataset Name ===>
Volume Serial ===> (If not catalogued)

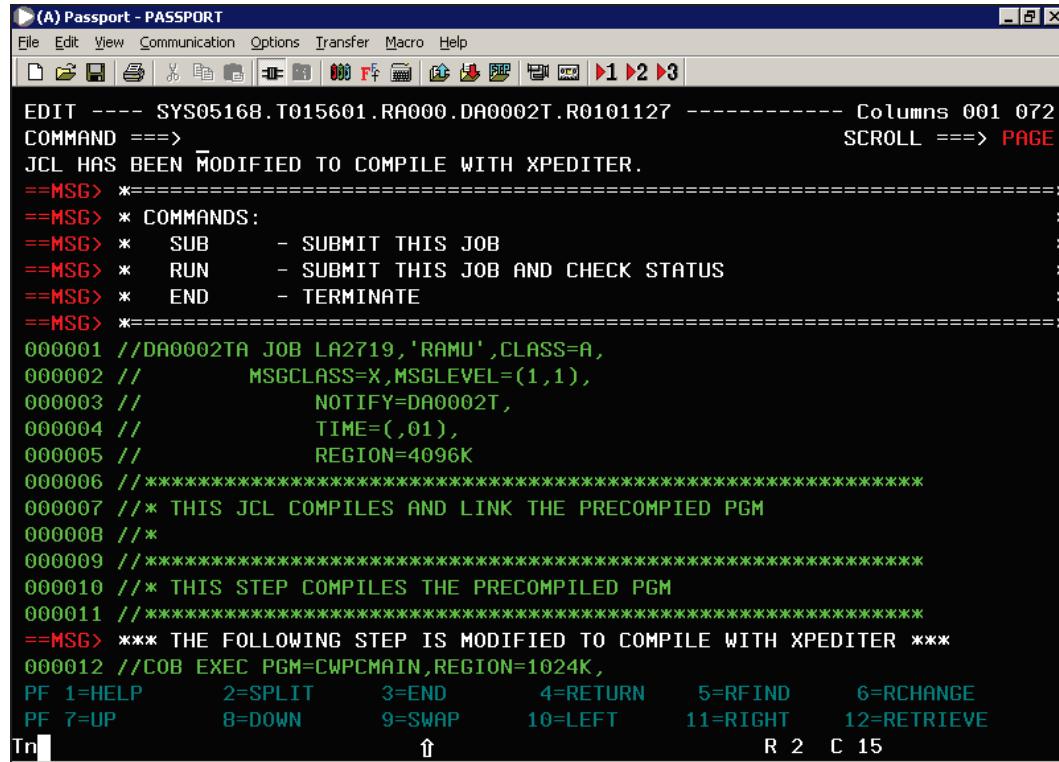
Language Processor Related Items:
DDIO File ===> 'DA0002T.XPDITER.DDIO'
Options Dataset ===>

PF 1=HELP      2=SPLIT      3=END      4=RETURN      5=RFIND      6=RCHANGE
PF 7=UP        8=DOWN       9=SWAP      10=LEFT       11=RIGHT     12=RETRIEVE
Tr [ ]          R 10 C 26
Connected to 3.32.24.12:23
[ ] [ ] NUM [ ] 10, 26 [ ]

```

Figure 42: Output

Step 13: This modifies the JCL to compile with **Xpediter**. Change the time parameter as **TIME=(,30)**.



```

EDIT ---- SYS05168.T015601.RA000.DA0002T.R0101127 ----- Columns 001 072
COMMAND ===> SCROLL ==> PAGE
JCL HAS BEEN MODIFIED TO COMPILE WITH XPEDITER.
==MSG> *=====
==MSG> * COMMANDS:
==MSG> *   SUB      - SUBMIT THIS JOB
==MSG> *   RUN      - SUBMIT THIS JOB AND CHECK STATUS
==MSG> *   END      - TERMINATE
==MSG> *=====
000001 //DA0002TA JOB LA2719,'RAMU',CLASS=A,
000002 //           MSGCLASS=X,MSGLEVEL=(1,1),
000003 //           NOTIFY=DA0002T,
000004 //           TIME=(,01),
000005 //           REGION=4096K
000006 //*****
000007 //** THIS JCL COMPILES AND LINK THE PRECOMPILED PGM
000008 //**
000009 //*****
000010 //** THIS STEP COMPILES THE PRECOMPILED PGM
000011 //*****
==MSG> *** THE FOLLOWING STEP IS MODIFIED TO COMPILE WITH XPEDITER ***
000012 //COB EXEC PGM=CWPCMAIN,REGION=1024K,
PF 1=HELP     2=SPLIT    3=END      4=RETURN    5=RFIND      6=RCHANGE
PF 7=UP       8=DOWN     9=SWAP      10=LEFT     11=RIGHT    12=RETRIEVE

```

Figure 43: Output

Step 14: Type **SUB** in the command prompt to submit the job.

(A) Passport - PASSPORT

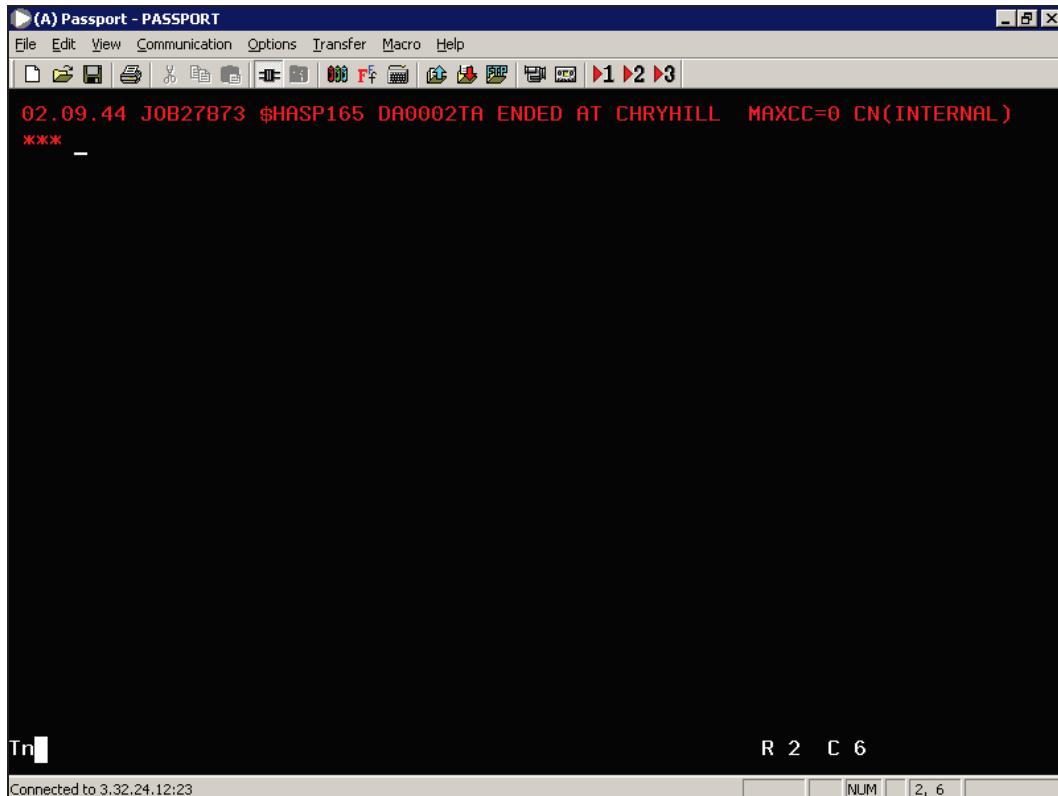
File Edit View Communication Options Transfer Macro Help

Columns 001 072
SCROLL ==> PAGE

```
EDIT ---- SYS05168.T015601.RA000.DA0002T.R0101127 -----
COMMAND ==>
000020 //SYSLIN DD DSN=88TEMP,DISP=(NEW,PASS),
000021 // UNIT=SYSALDDA,SPACE=(TRK,(40,40))
000022 //*****
000023 //SYSUT1 DD UNIT=SYSALDDA,SPACE=(TRK,(6,1))
000024 //SYSUT2 DD UNIT=SYSALDDA,SPACE=(CYL,(6,1))
000025 //SYSUT3 DD UNIT=SYSALDDA,SPACE=(CYL,(6,1))
000026 //SYSUT4 DD UNIT=SYSALDDA,SPACE=(CYL,(6,1))
000027 //** *****
000028 //** HERE YOU SPECIFY PRECOMPILED PGM MEMBER
000029 /**
000030 //SYSIN DD DSN=DA0002T.RAMU.COBOL(ASS03),DISP=SHR
000031 //*****
000032 // STEP TO LINK THE COBOL PROGRAM
000033 //*****
000034 //STEPLIB DD DISP=SHR,DSN=SYS6.COMPECC.MLCX800.SLCXLOAD
000035 //XOPTIONS DD DISP=SHR,DSN=SYS6.XPED.H72.XOPTIONS
000036 //CWPDDIO DD DISP=SHR,DSN=DA0002T.XPEDITER.DDIO
000037 //CWPPRMO DD *
000038 COBOL(OUTPUT(PRINT,DDIO))
000039 PROCESSOR(OUTPUT(NOPRINT,NODDIO),TEXT(NONE))
PF 1=HELP      2=SPLIT      3=END        4=RETURN      5=RFINDD      6=RCHANGE
PF 7=UP       8=DOWN       9=SWAP       10=LEFT       11=RIGHT      12=RETRIEVE
Tn [ ]          ↑           R 2 C 15
Connected to 3.32.24.12:23
```

Figure 44: Output

Step 15: Check for **MAXCC = 0** to ensure that the job is completed successfully.



The screenshot shows a terminal window titled '(A) Passport - PASSPORT'. The window has a menu bar with File, Edit, View, Communication, Options, Transfer, Macro, and Help. Below the menu is a toolbar with various icons. The main text area displays the following message in red text:
02.09.44 JOB27873 \$HASP165 DA0002TA ENDED AT CHRYHILL MAXCC=0 CN(INTERNAL)

Below the text area, there is a status bar with the text 'Connected to 3.32.24.12:23' on the left and 'R 2 C 6' on the right. At the bottom, there is a numeric keypad and a status indicator showing 'NUM' and '2, 6'.

Figure 45: Output

Step 16: Again, to logon to **Xpediter**, type **D.E.**

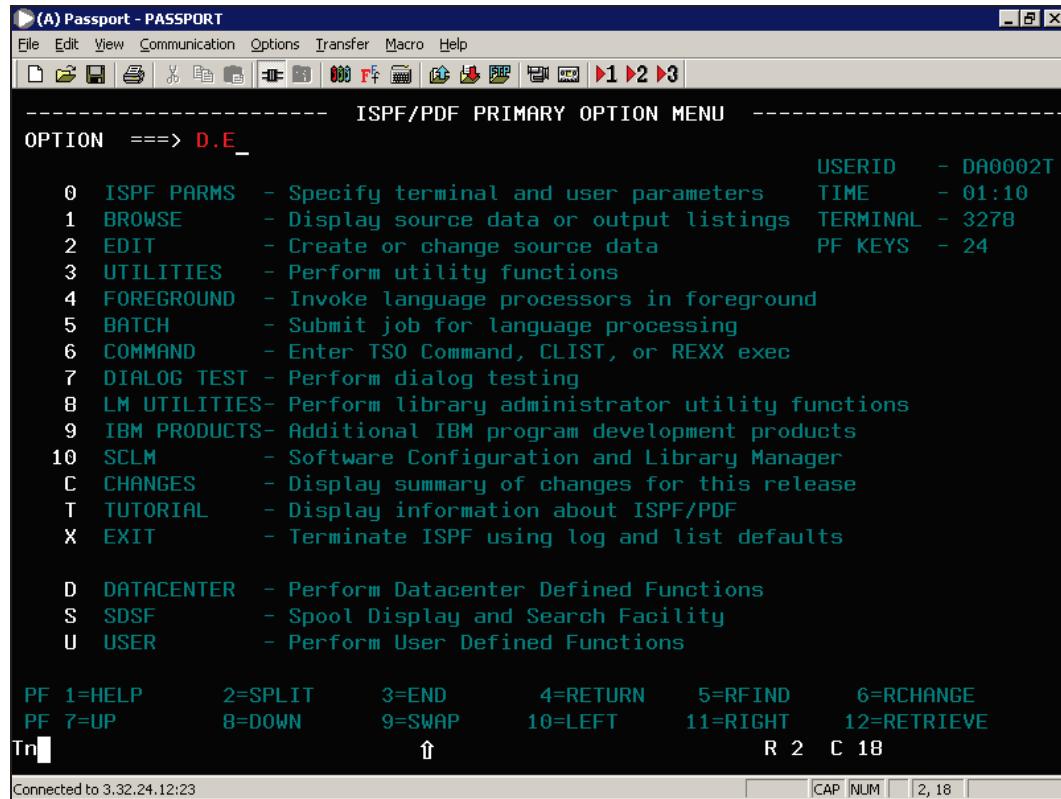


Figure 46: Output

Step 17: From the **Xpediter Menu**, choose **2** to debug programs interactively under TSO. If you are entering the **TSO Debugging menu** for the first time, this will open the **Environments menu**.

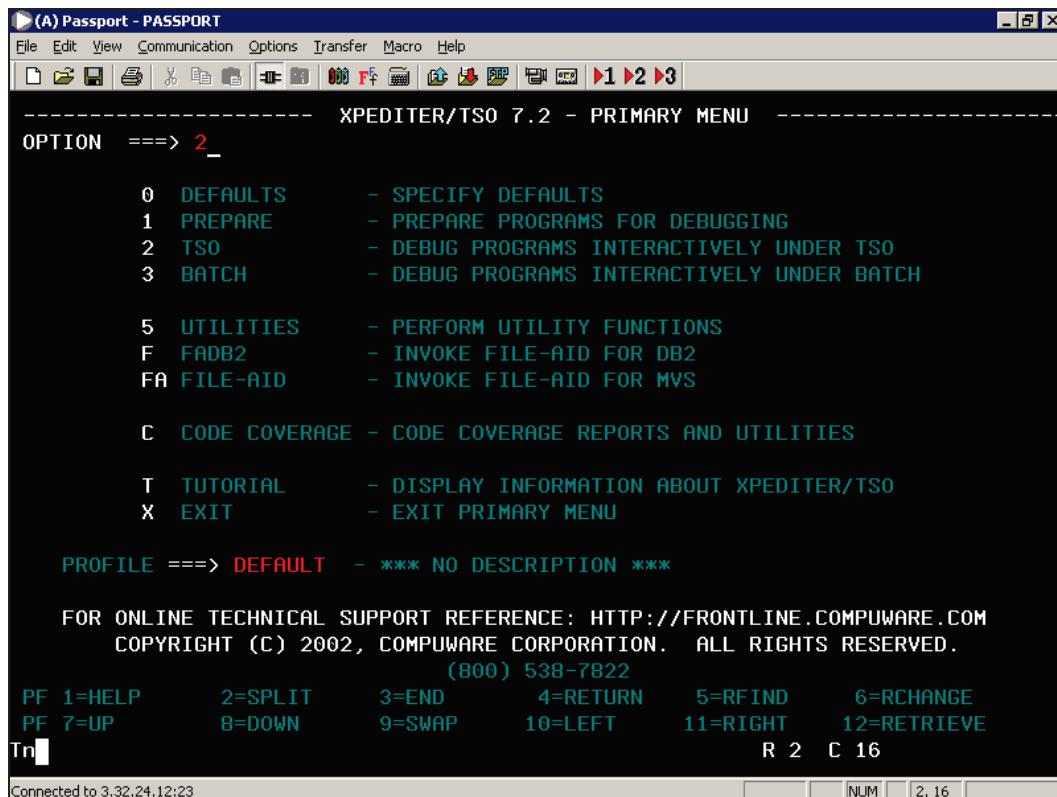


Figure 47: Output

Step 18: From the **Environments Menu**, choose **1** to test the program with no special environment services.

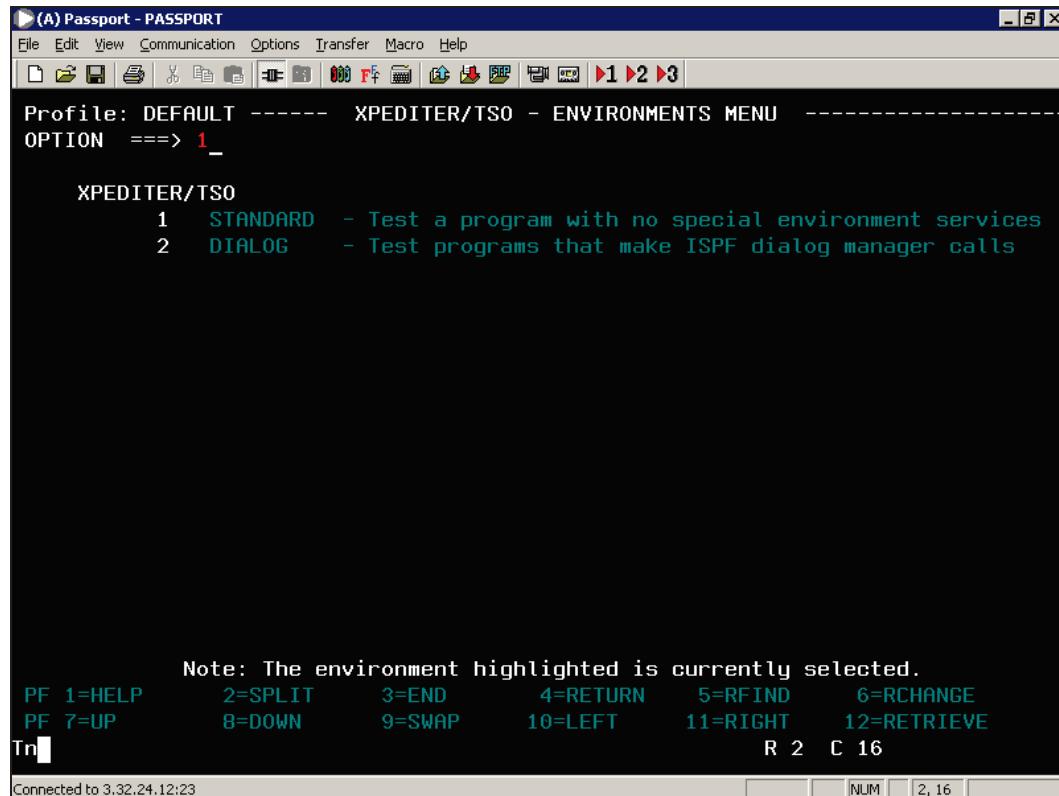


Figure 48: Output

Step 19: Under the **Test Selection Criteria**, enter the name of the COBOL program, which needs to be debugged. Also enter the name of the **run JCL**, which will be used to execute the program. Type **SE** to enter the **Setup Menu**.

(A) Passport - PASSPORT

File Edit View Communication Options Transfer Macro Help

Profile: DEFAULT ----- XPEDITER/TSO - STANDARD (2.1) -----
COMMAND ==> SE_

COMMANDS: SETUP (DISPLAY SETUP MENU)
PROFILE (DISPLAY PROFILE LIST)

TEST SELECTION CRITERIA:

PROGRAM ==> ASS03
ENTRY POINT ==>
LOAD MODULE ==>

INITIAL SCRIPT ==>
POST SCRIPT ==>

PARM (CAPS = YES) ==>

FILE LIST/JCL MEMBER ==> 'DA0002T.RAMU.JCL(RUN3)'
PREVIEW FILES? ==> NO
CODE COVERAGE TEST? ==> NO
IS THIS A DB2 TEST? ==> NO PLAN ==> SYSTEM ==>

PRESS ENTER TO PROCESS OR ENTER END COMMAND TO TERMINATE

PF 1=HELP	2=SPLIT	3=END	4=RETURN	5=RFINDD	6=RCHANGE
PF 7=UP	8=DOWN	9=SWAP	10=LEFT	11=RIGHT	12=RETRIEVE

Tn █ ↑ R 2 C 17

Connected to 3.32.24.12:23

Figure 49: Output

Step 20: From the **Setup Menu**, select **0 (Zero)** to go to the **Environments Menu**.

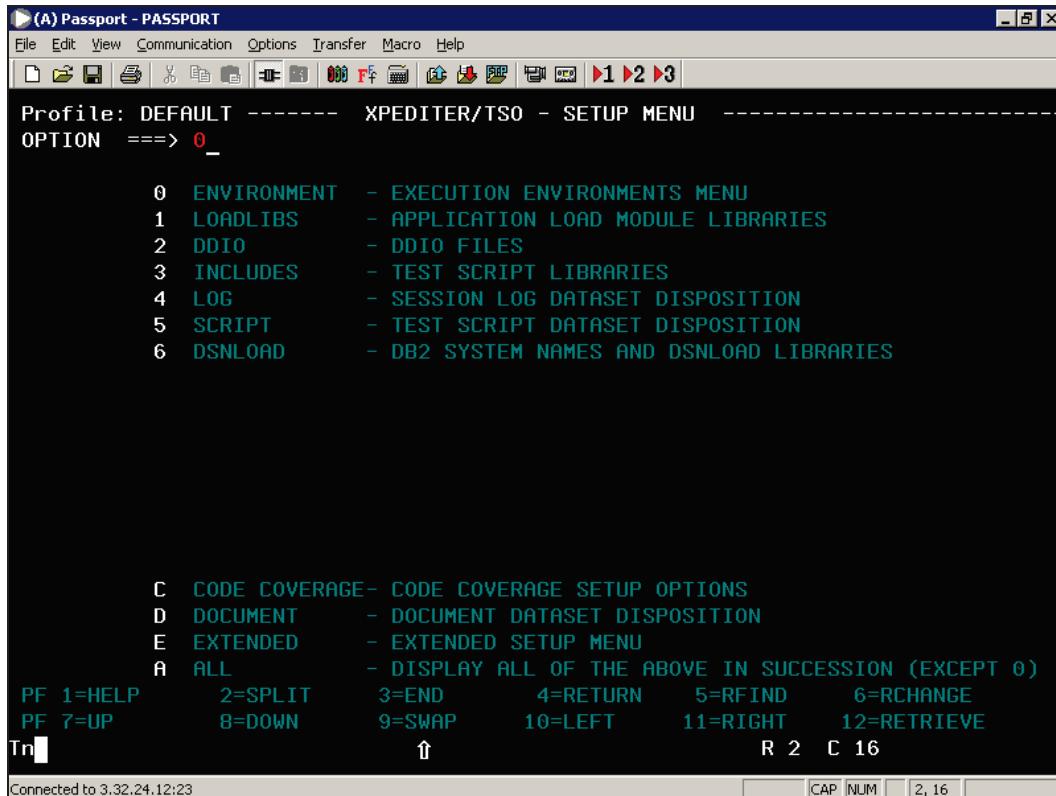


Figure 50: Output

Step 21: Enter the Load Libraries (**LOADLIBS**) that you will be using to execute the program. You may add more than one Loadlib. To enter the **LOADLIBS**, type 1 in the following **Environments Menu** and press **ENTER**. This will take you to the **Setup Menu**.

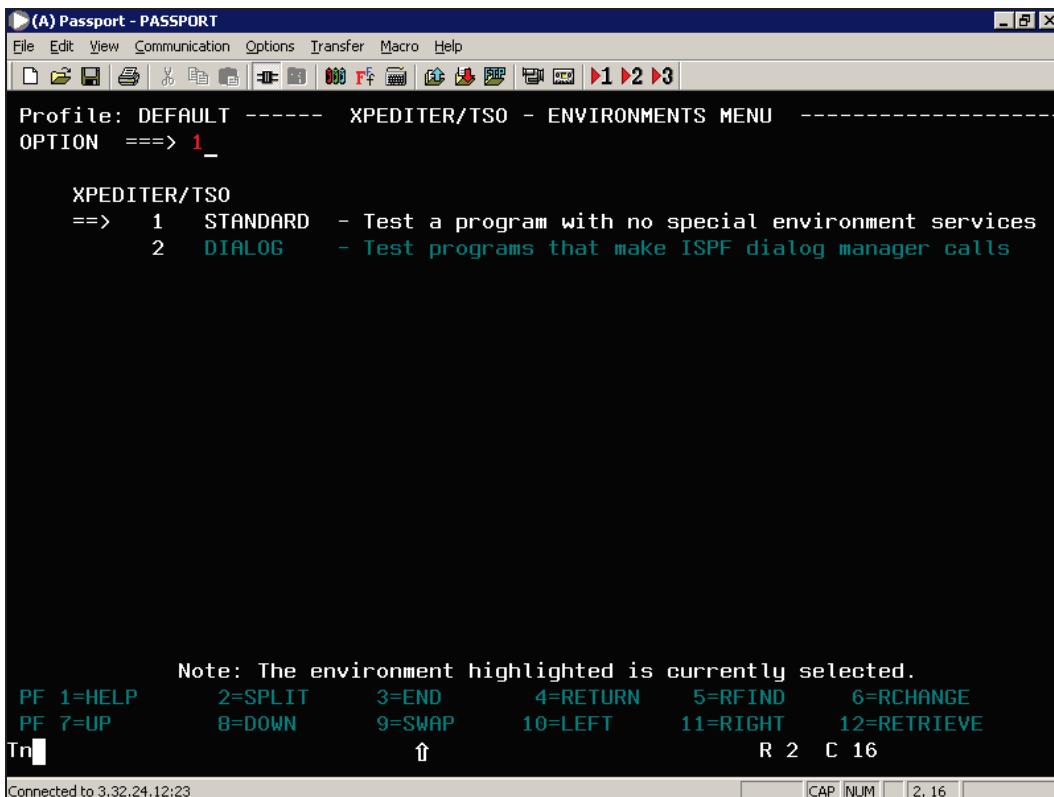


Figure 51: Output

Step 22: Select 1 to enter the **LOADLIBs**, where the load module of your COBOL program is created.

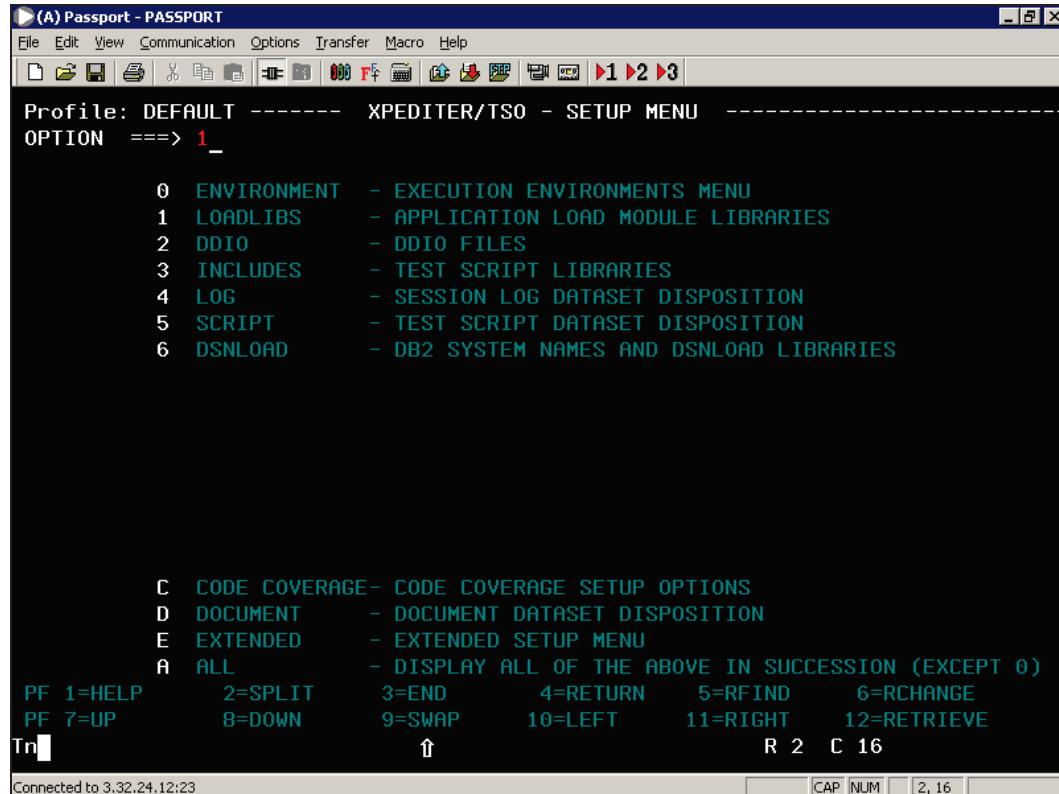
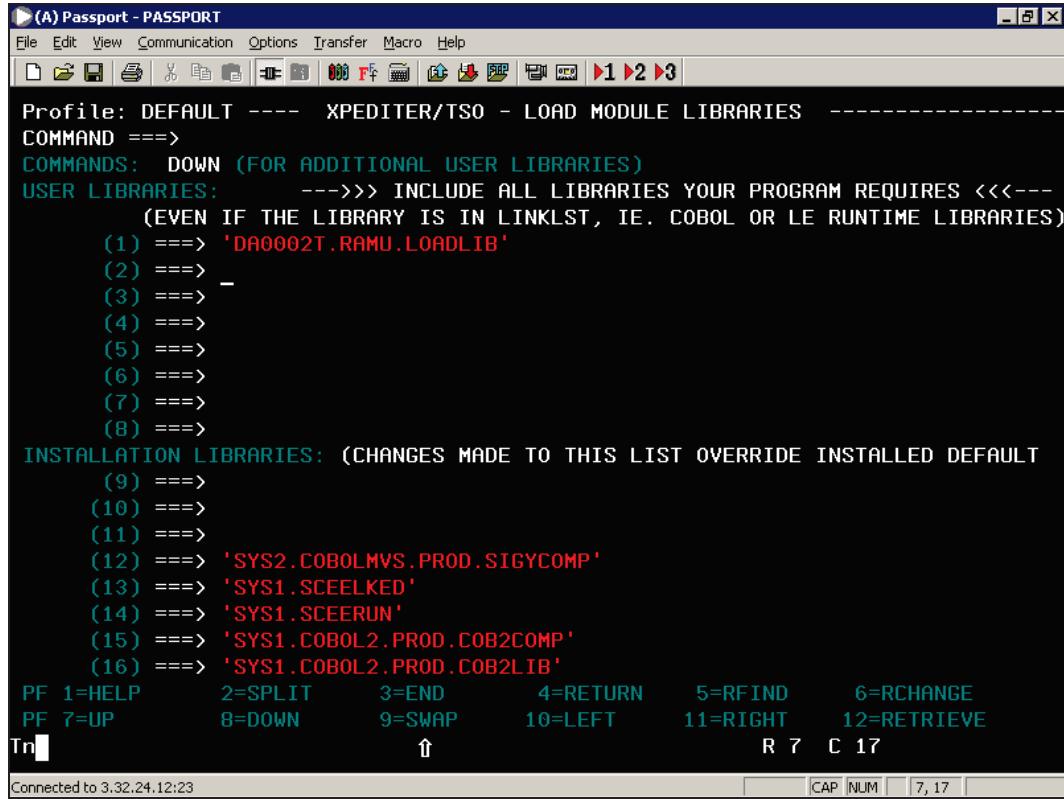


Figure 52: Output

Step 23: Enter the LOADLIB Dataset name of your COBOL program.



```

(A) Passport - PASSPORT
File Edit View Communication Options Transfer Macro Help
| 1 2 3
Profile: DEFAULT ---- XPEDITER/TSO - LOAD MODULE LIBRARIES -----
COMMAND ===>
COMMANDS: DOWN (FOR ADDITIONAL USER LIBRARIES)
USER LIBRARIES: --->>> INCLUDE ALL LIBRARIES YOUR PROGRAM REQUIRES <<<---
(EVEN IF THE LIBRARY IS IN LINKLST, IE. COBOL OR LE RUNTIME LIBRARIES)
(1) ===> 'DA0002T.RAMU.LOADLIB'
(2) ===> -
(3) ===>
(4) ===>
(5) ===>
(6) ===>
(7) ===>
(8) ===>
INSTALLATION LIBRARIES: (CHANGES MADE TO THIS LIST OVERRIDE INSTALLED DEFAULT
(9) ===>
(10) ===>
(11) ===>
(12) ===> 'SYS2.COBOLMVS.PROD.SIGYCOMP'
(13) ===> 'SYS1.SCEELKED'
(14) ===> 'SYS1.SCEERUN'
(15) ===> 'SYS1.COBOL2.PROD.COB2COMP'
(16) ===> 'SYS1.COBOL2.PROD.COB2LIB'
PF 1=HELP      2=SPLIT      3=END      4=RETURN      5=RFIND      6=RCHANGE
PF 7=UP        8=DOWN       9=SWAP      10=LEFT       11=RIGHT     12=RETRIEVE
Tr  R 7 C 17
Connected to 3.32.24.12:23
[CAP] [NUM] [7, 17]

```

Figure 53: Output

Step 24: From the **Setup Menu**, select **2** to go to **DDIO Files** option.

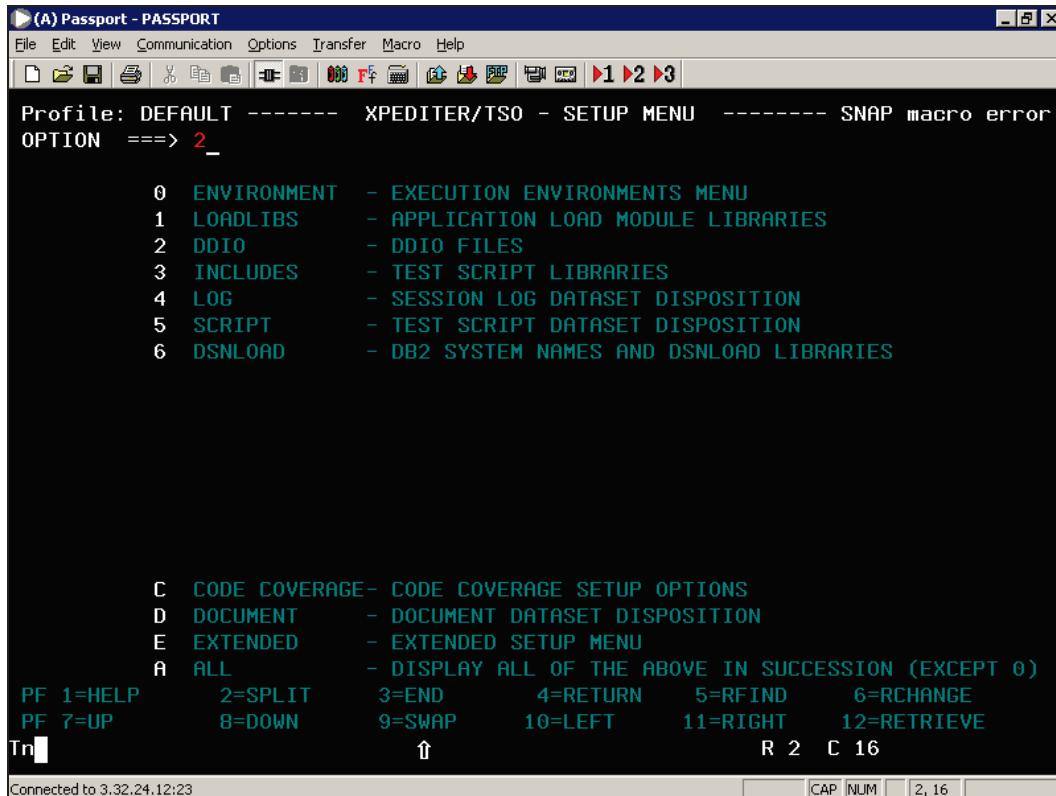
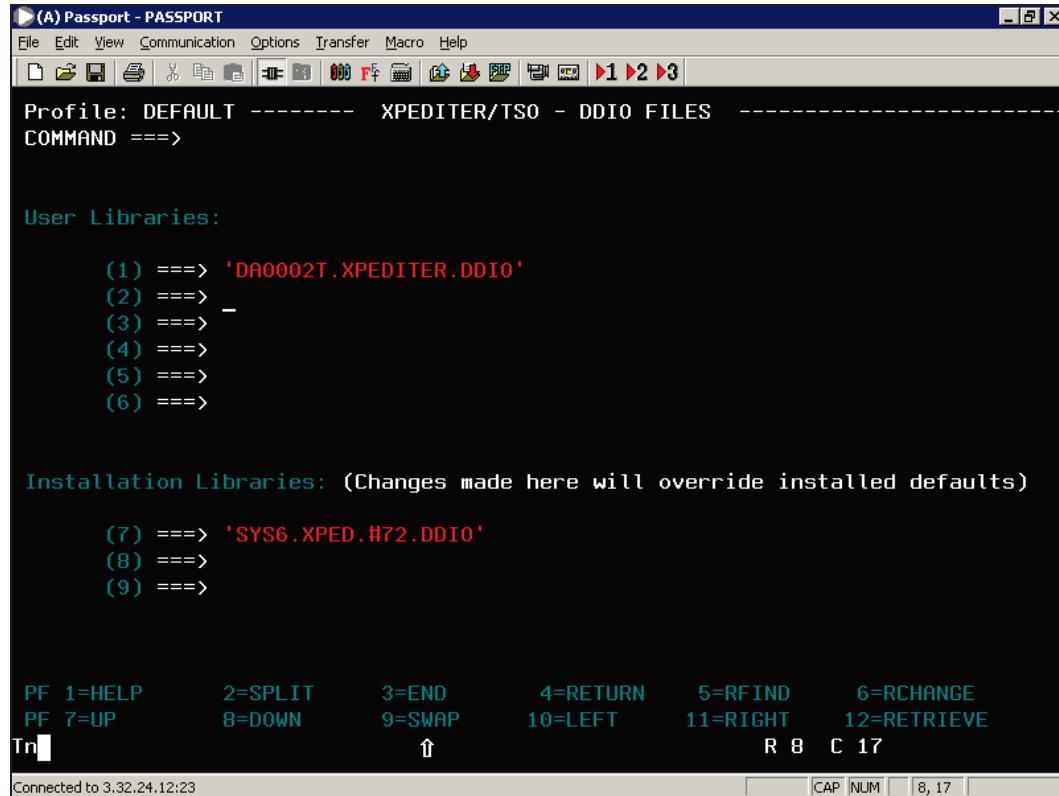


Figure 54: Output

Step 25: Enter the name of the **DDIO file** and press **Enter**. Press **F3** key to go back to the **TSO Execution Screen**.



The screenshot shows a TSO session window with the following content:

```

(A) Passport - PASSPORT
File Edit View Communication Options Transfer Macro Help
[File, Edit, View, Communication, Options, Transfer, Macro, Help, PF keys 1-12, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12]

Profile: DEFAULT ----- XPEDITER/TSO - DDIO FILES -----
COMMAND ==>

User Libraries:

(1) ==> 'DA0002T.XPEDITER.DDIO'
(2) ==> -
(3) ==>
(4) ==>
(5) ==>
(6) ==>

Installation Libraries: (Changes made here will override installed defaults)

(7) ==> 'SYS6.XPED.#72.DDIO'
(8) ==>
(9) ==>

PF 1=HELP      2=SPLIT      3=END      4=RETURN      5=RFIND      6=RCHANGE
PF 7=UP        8=DOWN       9=SWAP      10=LEFT       11=RIGHT     12=RETRIEVE
Tr [ ]          ↑           R 8 C 17

Connected to 3.32.24.12:23
[CAP NUM] 8,17

```

Figure 55: Output

Step 26: Select option **2** to debug programs interactively under TSO.

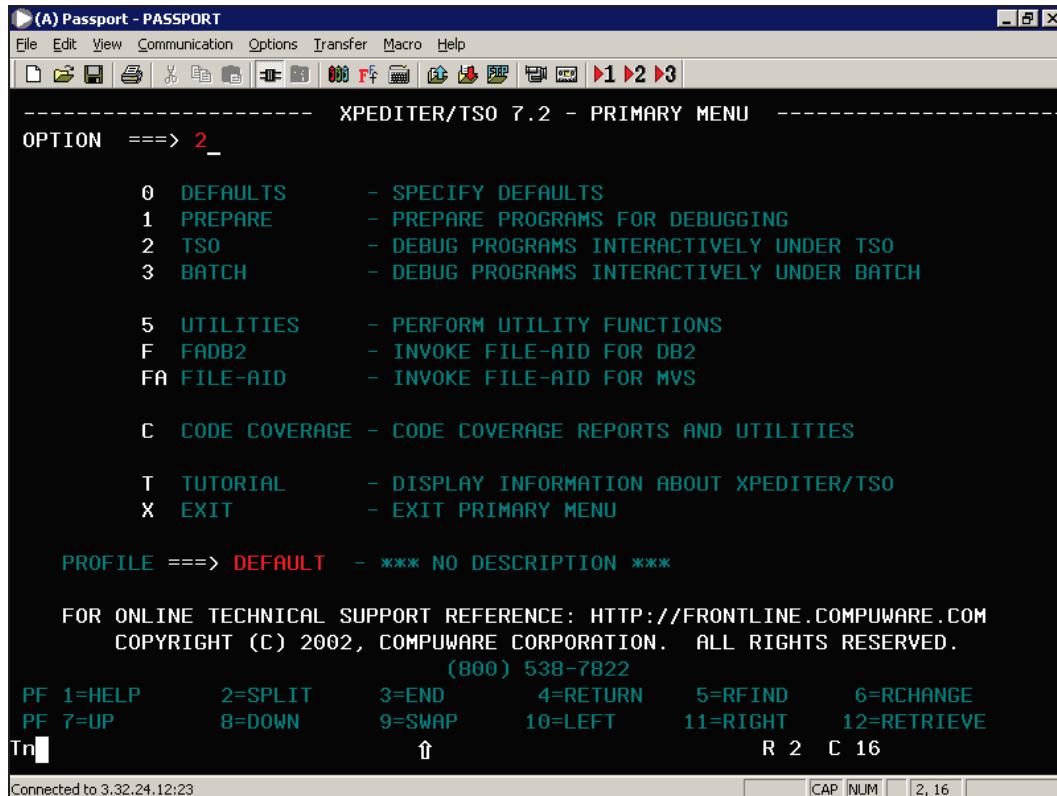
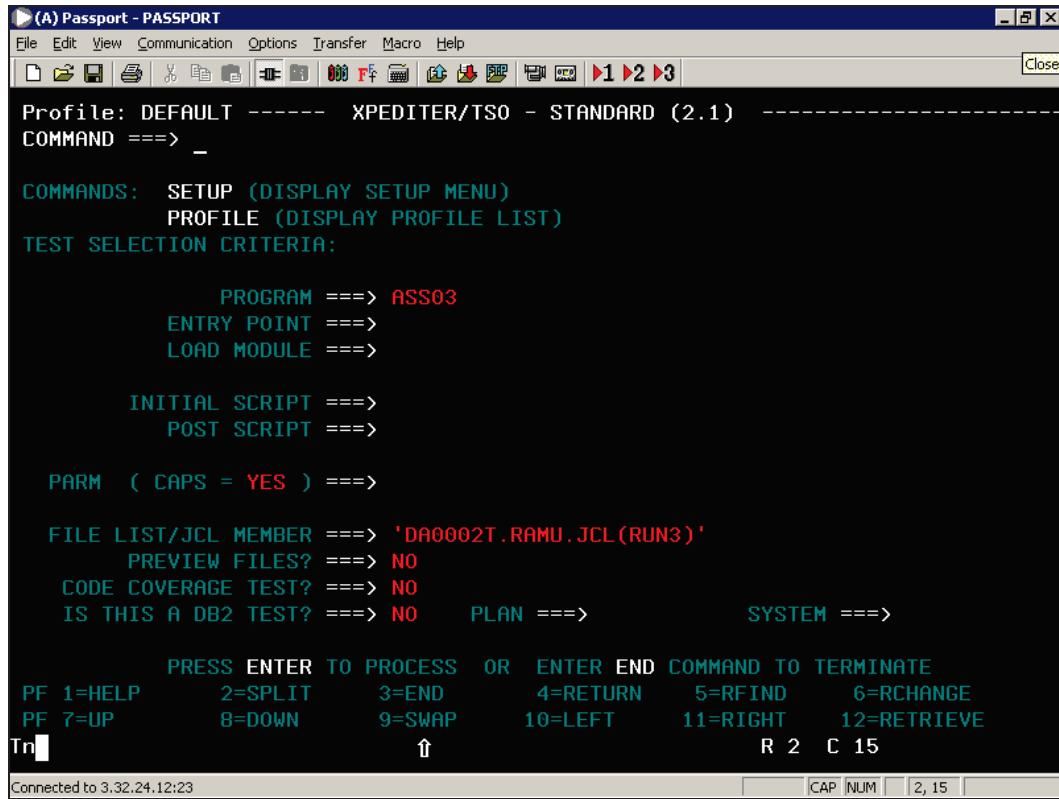


Figure 56: Output

Step 27: Press **Enter** to start debugging the Program.



```

(A) Passport - PASSPORT
File Edit View Communication Options Transfer Macro Help
[File Explorer] [Print] [Exit] [Close]
Profile: DEFAULT ----- XPEDITER/TSO - STANDARD (2.1) -----
COMMAND ===> _

COMMANDS: SETUP (DISPLAY SETUP MENU)
          PROFILE (DISPLAY PROFILE LIST)
TEST SELECTION CRITERIA:

      PROGRAM ===> ASS03
      ENTRY POINT ===>
      LOAD MODULE ===>

      INITIAL SCRIPT ===>
      POST SCRIPT ===>

PARM ( CAPS = YES ) ===>

FILE LIST/JCL MEMBER ===> 'DA0002T.RAMU.JCL(RUN3)'
      PREVIEW FILES? ===> NO
      CODE COVERAGE TEST? ===> NO
      IS THIS A DB2 TEST? ===> NO      PLAN ===>           SYSTEM ===>

PRESS ENTER TO PROCESS OR ENTER END COMMAND TO TERMINATE
PF 1=HELP    2=SPLIT    3=END    4=RETURN   5=RFIND    6=RCHANGE
PF 7=UP      8=DOWN     9=SWAP    10=LEFT    11=RIGHT   12=RETRIEVE
Tn [Up Arrow] [Down Arrow] R 2 C 15
Connected to 3.32.24.12:23
[CAP NUM] [2, 15]

```

Figure 57: Output

Step 28: The TSO debugging entry screen will be displayed as shown below. Press **ENTER** to continue.



Figure 58: Output

Step 29: Now you are in the debugging session. You can see a highlighted line (above line no. 000053) in your code, which indicates that the control of execution is started with the highlighted line.

(A) Passport - PASSPORT

File Edit View Communication Options Transfer Macro Help

COMMAND ==> _ SCROLL ==> CSR

--- XPEDITOR/TSO - SOURCE ---

BEFORE BREAKPOINT ENCOUNTERED

000048 01 PARM > ..
*** END ***

----- Before ASS031 <>

===== B PROCEDURE DIVISION USING PARM.

000053
000054 0000-MAIN.
000055 OPEN INPUT EMP-FILE.
000056
000057 PERFORM 1000-READ THRU 1000-EXIT UNTIL 88-YES.
000058 PERFORM 2000-TERM THRU 2000-EXIT
000059 A STOP RUN.
000060
000061 1000-READ.
000062
000063 READ EMP-FILE
000064 AT END
PF 1=HELP 2=PEEK CSR 3=END 4=EXIT 5=FIND 6=LOCATE *
PF 7=UP B=DOWN 9=GO 1 10=LEFT 11=RIGHT 12=GO

Tn [] ↑ R 2 C 15

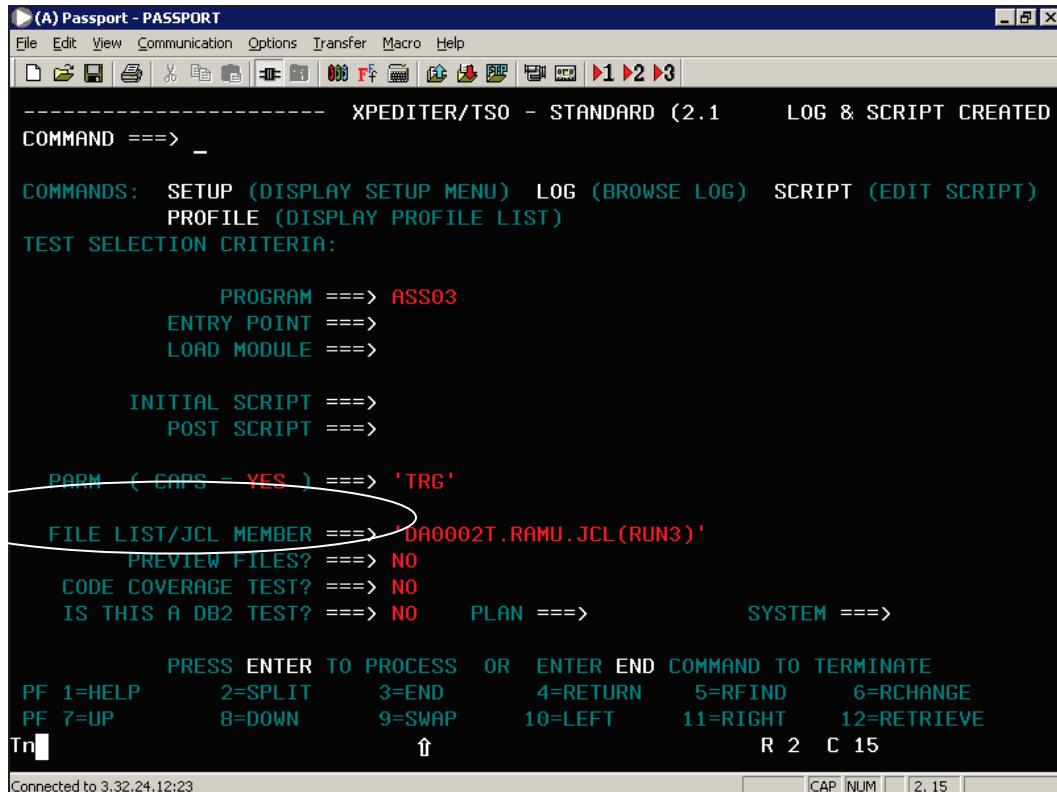
Connected to 3.32.24.12:23 CAP NUM 2,15

Figure 59: Output

Note: On screen, you can see a red-dotted arrow on the left of the highlighted line.

Step 30: Since the above code is using **PARM parameter value** as the input for this program, you must set the **PARM parameter value** before you move to the debug session.

To set the **PARM parameter**, type the **PARM parameter value**, under the **Test selection criteria** in the highlighted area as shown in the following figure.



```

(A) Passport - PASSPORT
File Edit View Communication Options Transfer Macro Help
----- XPEDITOR/TSO - STANDARD (2.1) LOG & SCRIPT CREATED
COMMAND ===> _

COMMANDS: SETUP (DISPLAY SETUP MENU) LOG (BROWSE LOG) SCRIPT (EDIT SCRIPT)
          PROFILE (DISPLAY PROFILE LIST)
TEST SELECTION CRITERIA:

      PROGRAM ===> ASS03
      ENTRY POINT ===>
      LOAD MODULE ===>

      INITIAL SCRIPT ===>
      POST SCRIPT ===>

      PARM ( CAPS = YES ) ===> 'TRG' (highlighted)
      FILE LIST/JCL MEMBER ===> 'DA0002T.RAMU.JCL(RUN3)'
          PREVIEW FILES? ===> NO
          CODE COVERAGE TEST? ===> NO
          IS THIS A DB2 TEST? ===> NO      PLAN ===>           SYSTEM ===>

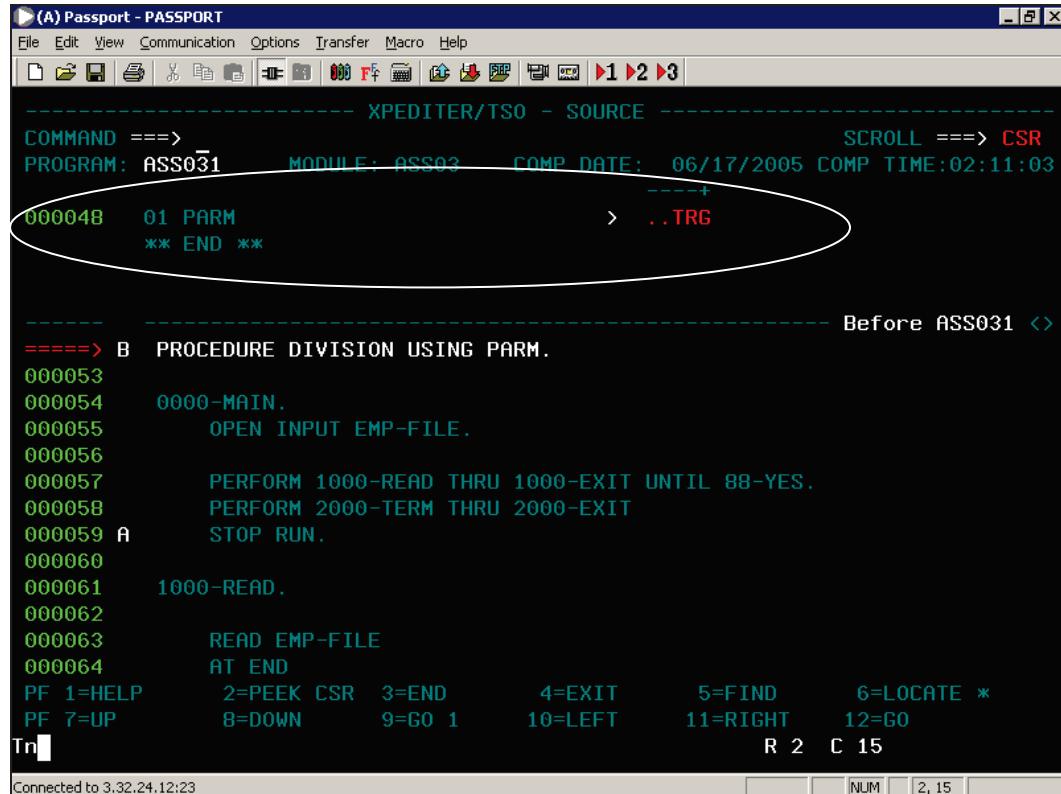
      PRESS ENTER TO PROCESS OR ENTER END COMMAND TO TERMINATE
PF 1=HELP    2=SPLIT    3=END    4=RETURN   5=RFIND    6=RCHANGE
PF 7=UP     8=DOWN     9=SWAP    10=LEFT    11=RIGHT   12=RETRIEVE
Tn[ ]                                ↑
                                         R 2 C 15
Connected to 3.32.24.12:23

```

Figure 60: Output

Note: 'TRG' is the input value specified for this program and it varies according to your code and your input.

Step 31: Press **ENTER** key till you get the following screen and check if the **PARM** parameter is set to **TRG**.



```

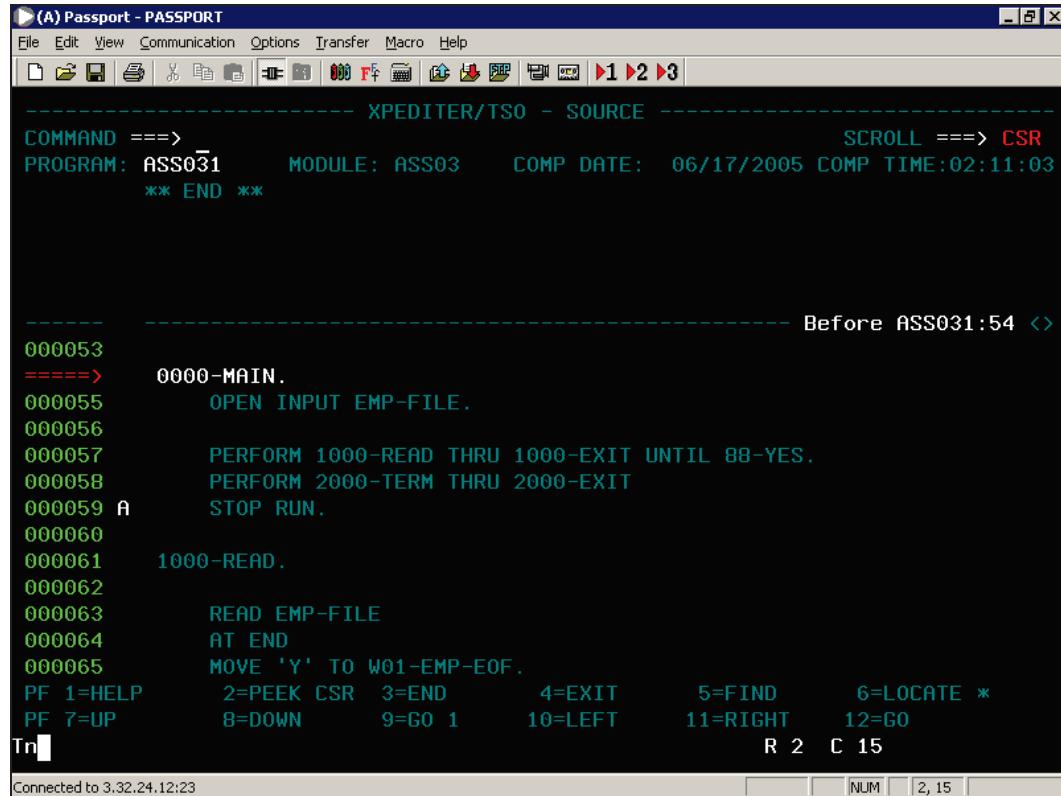
(A) Passport - PASSPORT
File Edit View Communication Options Transfer Macro Help
| | | | | | F | | | | | | 1 2 3 |
----- XPEDITER/TSO - SOURCE -----
COMMAND ===> SCROLL ===> CSR
PROGRAM: ASS031 MODULE: ASS03 COMP DATE: 06/17/2005 COMP TIME: 02:11:03
-----+
000048 01 PARM > ..TRG
** END **

----- Before ASS031 <>
=====> B PROCEDURE DIVISION USING PARM.
000053
000054 0000-MAIN.
000055      OPEN INPUT EMP-FILE.
000056
000057      PERFORM 1000-READ THRU 1000-EXIT UNTIL 88-YES.
000058      PERFORM 2000-TERM THRU 2000-EXIT
000059 A      STOP RUN.
000060
000061 1000-READ .
000062
000063      READ EMP-FILE
000064      AT END
PF 1=HELP   2=PEEK CSR 3=END      4=EXIT      5=FIND      6=LOCATE *
PF 7=UP     8=DOWN    9=GO 1    10=LEFT    11=RIGHT   12=GO
Tr [ ] R 2 C 15
Connected to 3.32.24.12:23
[ ] [ ] NUM [ ] 2, 15

```

Figure 61: Output

Step 32: Press **F9** key for line-by-line debugging of the program. Observe that the first line from the source code, **0000-MAIN.**, is highlighted.



```

(A) Passport - PASSPORT
File Edit View Communication Options Transfer Macro Help
File Edit View Communication Options Transfer Macro Help
----- XPDITER/TSO - SOURCE -----
COMMAND ===> SCROLL ===> CSR
PROGRAM: ASS031 MODULE: ASS03 COMP DATE: 06/17/2005 COMP TIME:02:11:03
*** END **

----- ----- Before ASS031:54 <>
000053
=====> 0000-MAIN.
000055      OPEN INPUT EMP-FILE.
000056
000057      PERFORM 1000-READ THRU 1000-EXIT UNTIL 88-YES.
000058      PERFORM 2000-TERM THRU 2000-EXIT
000059 A      STOP RUN.
000060
000061      1000-READ.
000062
000063      READ EMP-FILE
000064      AT END
000065      MOVE 'Y' TO W01-EMP-EOF.
PF 1=HELP    2=PEEK CSR  3=END      4=EXIT      5=FIND      6=LOCATE *
PF 7=UP      8=DOWN     9=GO 1    10=LEFT    11=RIGHT   12=GO
Tr [ ] R 2 C 15
Connected to 3.32.24.12:23
[ ] [ ] NUM [ ] 2, 15

```

Figure 62: Output

Step 33: Press **F9** key so that control passes to the next line. The line being executed is highlighted.

(A) Passport - PASSPORT

File Edit View Communication Options Transfer Macro Help

XPEDITER/TSO - SOURCE

COMMAND ==> SCROLL ==> CSR

PROGRAM: ASS031 MODULE: ASS03 COMP DATE: 06/17/2005 COMP TIME: 02:11:03

000025 01 EMP-REC > NO ADDR
*** END ***

----- Before ASS031:55 <>

000053
000054 0000-MAIN.
===== > OPEN INPUT EMP-FILE.
000056
000057 PERFORM 1000-READ THRU 1000-EXIT UNTIL 88-YES.
000058 PERFORM 2000-TERM THRU 2000-EXIT
000059 A STOP RUN.
000060
000061 1000-READ.
000062
000063 READ EMP-FILE
000064 AT END
000065 MOVE 'Y' TO W01-EMP-EOF.
PF 1=HELP 2=PEEK CSR 3=END 4=EXIT 5=FIND 6=LOCATE *
PF 7=UP 8=DOWN 9=GO 1 10=LEFT 11=RIGHT 12=GO
Tn R 2 C 15

Connected to 3.32.24.12:23

Figure 63: Output

Step 34: Press **F9** key. Whenever the control reaches a **PERFORM PARA** statement, the control jumps to the respective **PARA**.

(A) Passport - PASSPORT

File Edit View Communication Options Transfer Macro Help

--- XPEDITER/TSO - SOURCE ---

COMMAND ==> SCROLL ==> CSR

PROGRAM: ASS031 MODULE: ASS03 COMP DATE: 06/17/2005 COMP TIME: 02:11:03

000037 01 W01-EMP-EOF > .
*** END ***

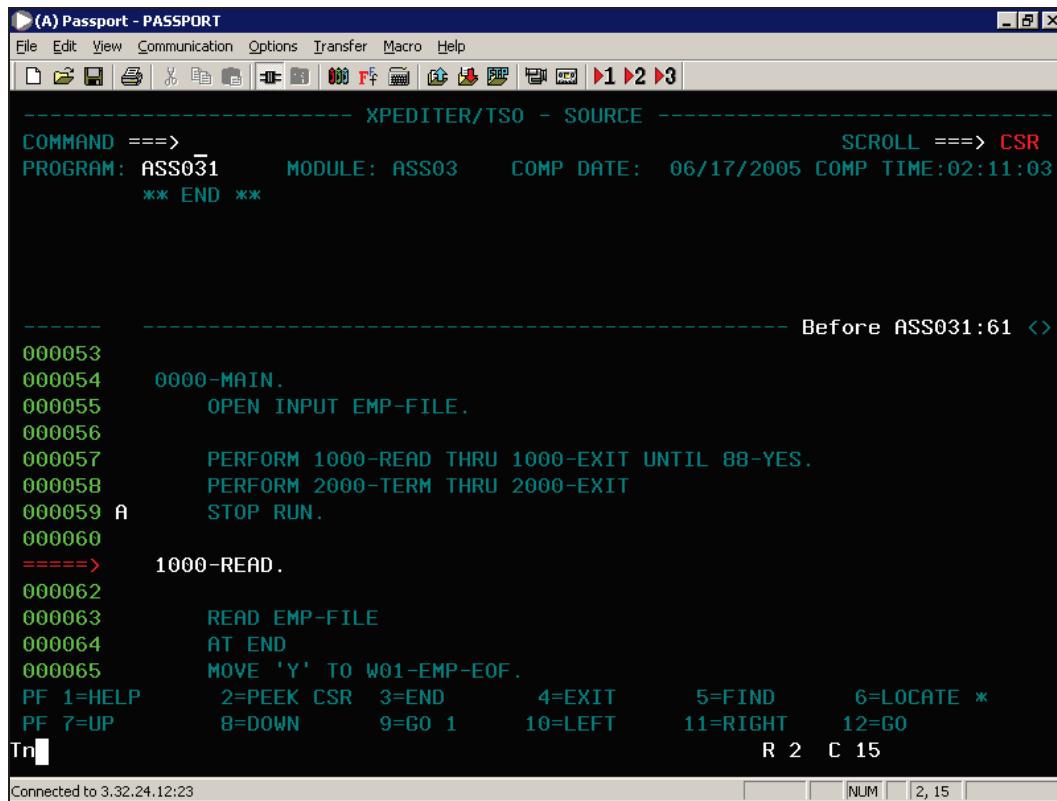
----- Before ASS031:57 -----

000053
000054 0000-MAIN.
000055 OPEN INPUT EMP-FILE.
000056
=====> PERFORM 1000-READ THRU 1000-EXIT UNTIL 88-YES.
000058 PERFORM 2000-TERM THRU 2000-EXIT
000059 A STOP RUN.
000060
000061 1000-READ.
000062
000063 READ EMP-FILE
000064 AT END
000065 MOVE 'Y' TO W01-EMP-EOF.
PF 1=HELP 2=PEEK CSR 3=END 4=EXIT 5=FIND 6=LOCATE *
PF 7=UP 8=DOWN 9=GO 1 10=LEFT 11=RIGHT 12=GO
Tn R 2 C 15

Connected to 3.32.24.12:23

Figure 64: Output

Step 35: Press **F9** key. Note that the control has transferred to **1000-READ** para.



```

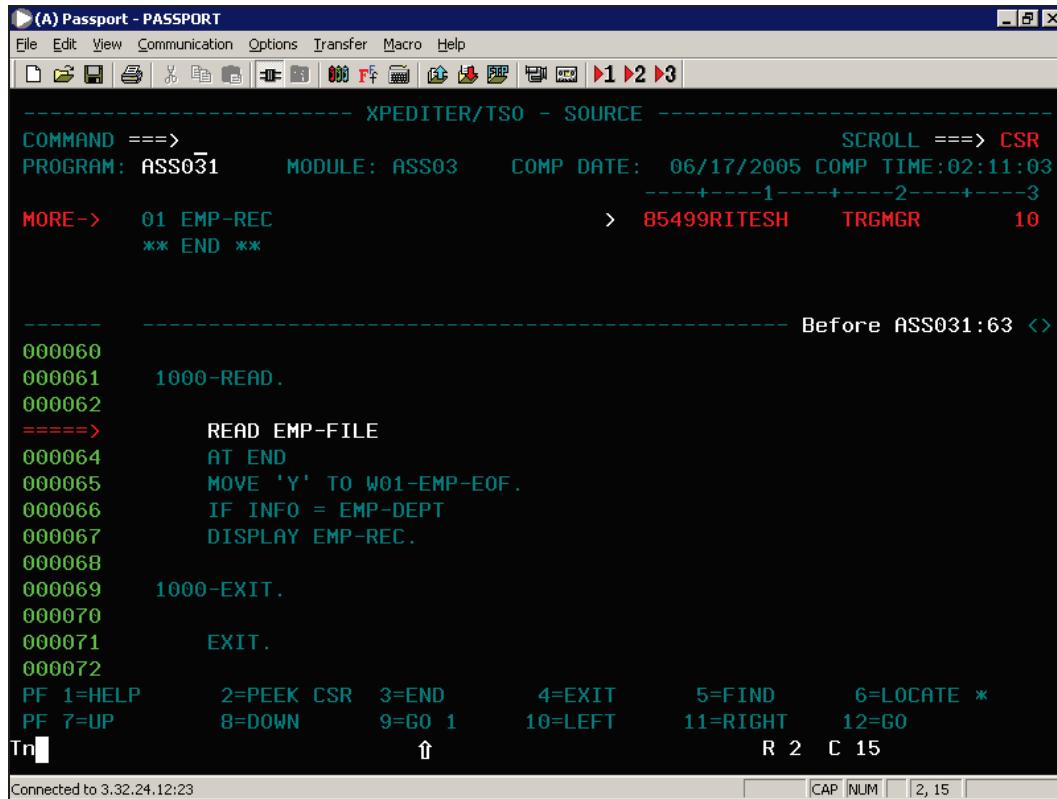
----- XPEDITER/TSO - SOURCE -----
COMMAND ===> SCROLL ==> CSR
PROGRAM: ASS031      MODULE: ASS03      COMP DATE: 06/17/2005 COMP TIME:02:11:03
** END **

----- Before ASS031:61 <>
000053
000054      0000-MAIN.
000055      OPEN INPUT EMP-FILE.
000056
000057      PERFORM 1000-READ THRU 1000-EXIT UNTIL 88-YES.
000058      PERFORM 2000-TERM THRU 2000-EXIT
000059 A      STOP RUN.
000060
=====> 1000-READ.
000062
000063      READ EMP-FILE
000064      AT END
000065      MOVE 'Y' TO W01-EMP-EOF.
PF 1=HELP    2=PEEK CSR   3=END      4=EXIT      5=FIND      6=LOCATE *
PF 7=UP      8=DOWN     9=GO 1     10=LEFT     11=RIGHT    12=GO
Tr[ ] R 2 C 15
Connected to 3.32.24.12:23
[ ] [ ] NUM [ ] 2, 15 [ ]

```

Figure 65: Output

Step 36: Continue pressing **F9** key for line-by-line execution.



```

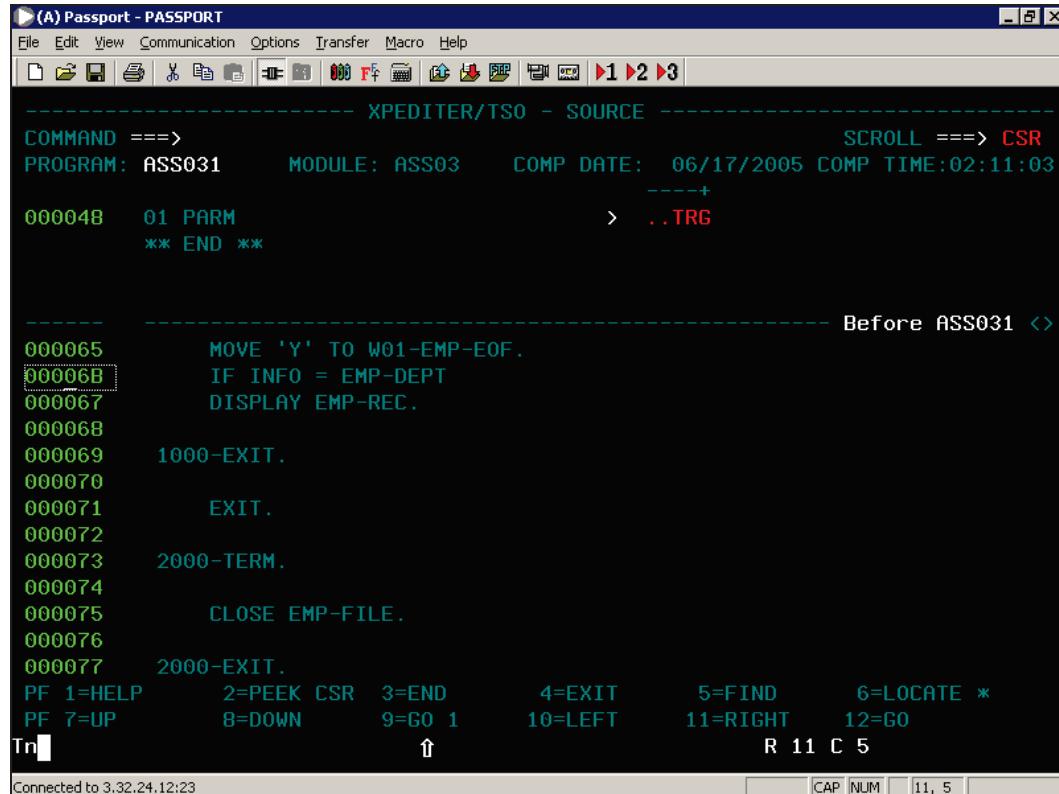
(A) Passport - PASSPORT
File Edit View Communication Options Transfer Macro Help
[File Explorer] [Print] [Exit] [F9] [Cancel] [OK] [1] [2] [3]
----- XPEDITER/TSO - SOURCE -----
COMMAND ==> SCROLL ==> CSR
PROGRAM: ASS031 MODULE: ASS03 COMP DATE: 06/17/2005 COMP TIME: 02:11:03
-----+---1---+---2---+---3
MORE-> 01 EMP-REC > 85499RITESH TRGMGR 10
** END **

----- Before ASS031:63 <>
000060
000061      1000-READ.
000062
=====>      READ EMP-FILE
000064      AT END
000065      MOVE 'Y' TO W01-EMP-EOF.
000066      IF INFO = EMP-DEPT
000067      DISPLAY EMP-REC.
000068
000069      1000-EXIT.
000070
000071      EXIT.
000072
PF 1=HELP    2=PEEK CSR  3=END      4=EXIT      5=FIND      6=LOCATE *
PF 7=UP      8=DOWN     9=GO 1    10=LEFT     11=RIGHT   12=GO
Tr [ ]          ↑          R 2 C 15
Connected to 3.32.24.12:23
[CAP] [NUM] [2, 15]

```

Figure 66: Output

Step 37: To set a breakpoint at a particular line, type **B** in the particular line number and press **ENTER**.



```

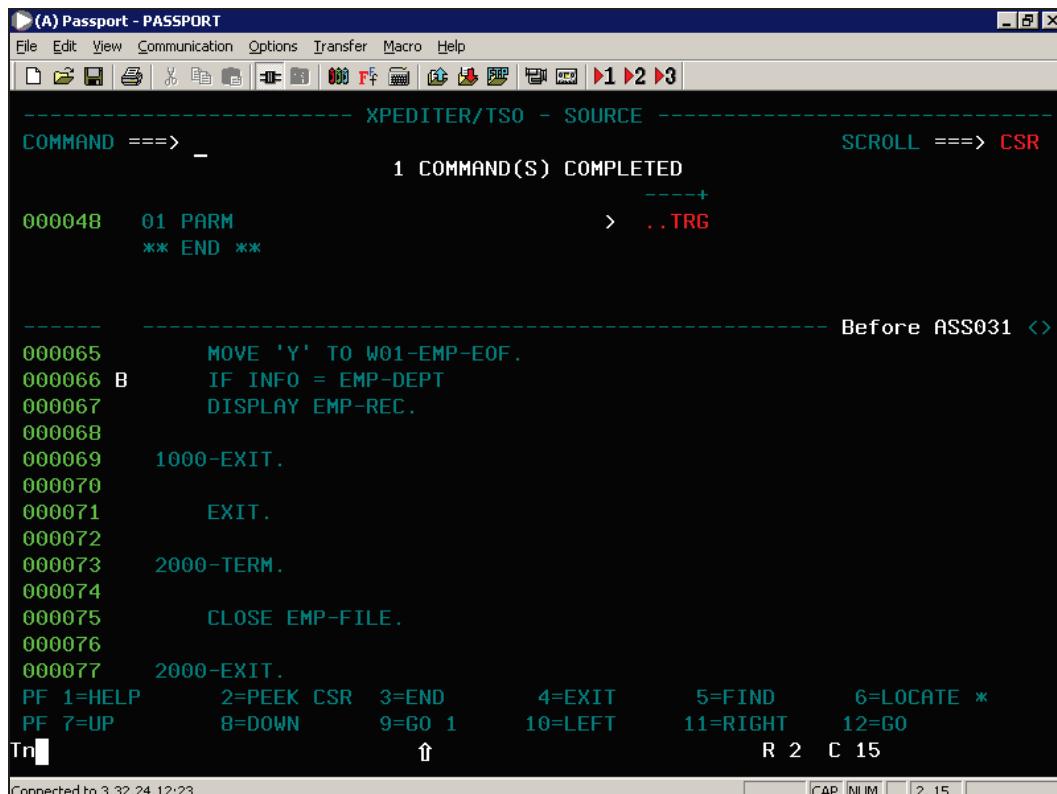
(A) Passport - PASSPORT
File Edit View Communication Options Transfer Macro Help
| | | | | | F | | | | | | 1 2 3 |
----- XPEDITER/TSO - SOURCE -----
COMMAND ===> SCROLL ===> CSR
PROGRAM: ASS031      MODULE: ASS03      COMP DATE: 06/17/2005 COMP TIME: 02:11:03
000048    01 PARM
          ** END **

----- Before ASS031 <>
000065      MOVE 'Y' TO W01-EMP-EOF.
00006B      IF INFO = EMP-DEPT
000067      DISPLAY EMP-REC.
000068
000069      1000-EXIT.
000070
000071      EXIT.
000072
000073      2000-TERM.
000074
000075      CLOSE EMP-FILE.
000076
000077      2000-EXIT.
PF 1=HELP   2=PEEK  CSR  3=END      4=EXIT      5=FIND      6=LOCATE *
PF 7=UP     8=DOWN   9=GO 1    10=LEFT     11=RIGHT    12=GO
Tr  R 11 C 5
Connected to 3.32.24.12:23

```

Figure 67: Output

Step 38: Once a breakpoint is placed on a line, the letter **B** appears in between the line number and the content on that line and you will get the status as “**1 COMMAND(S) COMPLETED**”.



```

----- XPEDITER/TSO - SOURCE -----
COMMAND ===> -
1 COMMAND(S) COMPLETED
-----+
000048 01 PARM
** END **

----- Before ASS031 <>
000065      MOVE 'Y' TO W01-EMP-EOF.
000066 B      IF INFO = EMP-DEPT
000067      DISPLAY EMP-REC.
000068
000069      1000-EXIT.
000070
000071      EXIT.
000072
000073      2000-TERM.
000074
000075      CLOSE EMP-FILE.
000076
000077      2000-EXIT.
PF 1=HELP    2=PEEK CSR  3=END      4=EXIT      5=FIND      6=LOCATE *
PF 7=UP      8=DOWN     9=GO 1    10=LEFT     11=RIGHT    12=GO
Tn[ ]          ↑
R 2 C 15

Connected to 3.32.24.12:23
[CAP] [NUM] [2, 15]

```

Figure 68: Output

Step 39: Use F12 to execute debugging between two breakpoints.

(A) Passport - PASSPORT

File Edit View Communication Options Transfer Macro Help

----- XPEDITER/TSO - SOURCE -----

COMMAND ==> - SCROLL ==> CSR

BEFORE BREAKPOINT ENCOUNTERED

000050 05 INFO > TRG
000028 05 EMP-DEPT > TRG
** END **

----- Before ASS031:66 <>

000065 MOVE 'Y' TO W01-EMP-EOF.
=====> B IF INFO = EMP-DEPT
000067 DISPLAY EMP-REC.
000068
000069 1000-EXIT.
000070
000071 EXIT.
000072
000073 2000-TERM.
000074
000075 CLOSE EMP-FILE.
000076
000077 2000-EXIT.

PF 1=HELP 2=PEEK CSR 3=END 4=EXIT 5=FIND 6=LOCATE *
PF 7=UP B=DOWN 9=GO 1 10=LEFT 11=RIGHT 12=GO

Tn [] ↑ R 2 C 15

Connected to 3.32.24.12:23 CAP NUM 2, 15

Figure 69: Output

Note: The **F12** key will not debug line-by-line. If you want to execute line-by-line, then press **F9** key at any point of time.

(A) Passport - PASSPORT

File Edit View Communication Options Transfer Macro Help

1 2 3

----- XPEDITOR/TSO - SOURCE -----

COMMAND ===> - SCROLL ===> CSR

BEFORE BREAKPOINT ENCOUNTERED

000050 05 INFO > INC

000028 05 EMP-DEPT > RND

** END **

----- Before ASS031:66 <>

000060

000061 1000-READ.

000062

000063 READ EMP-FILE

000064 AT END

000065 MOVE 'Y' TO W01-EMP-EOF.

=====> B IF INFO = EMP-DEPT

000067 DISPLAY EMP-REC.

000068

000069 1000-EXIT.

000070

000071 EXIT.

000072

PF 1=HELP 2=PEEK CSR 3=END 4=EXIT 5=FIND 6=LOCATE *

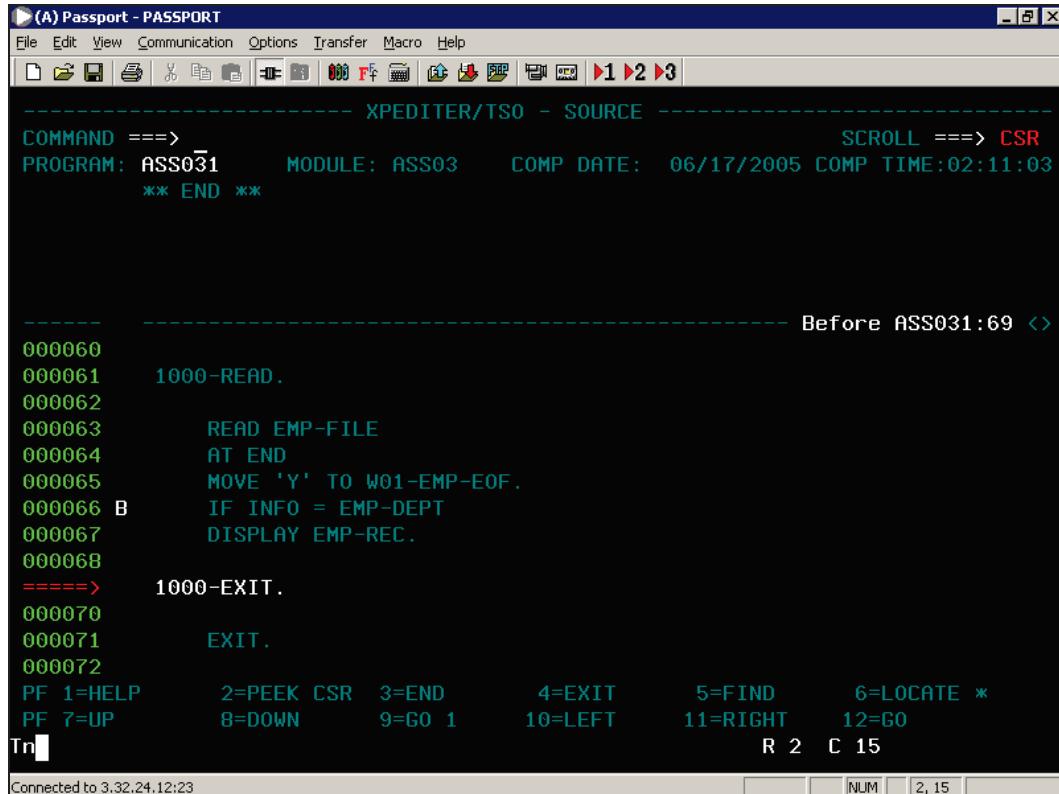
PF 7=UP 8=DOWN 9=GO 1 10=LEFT 11=RIGHT 12=GO

Tn R 2 C 15

Connected to 3.32.24.12:23

Figure 70: Output

Step 40: Since the condition is not matching, it skipped the Display statement (**Display Emp-rec**) and jumped to **1000-Exit**.



```

(A) Passport - PASSPORT
File Edit View Communication Options Transfer Macro Help
[File] [Edit] [View] [Communication] [Options] [Transfer] [Macro] [Help]
[File] [Edit] [View] [Communication] [Options] [Transfer] [Macro] [Help]
----- XPEDITER/TSO - SOURCE -----
COMMAND ===> SCROLL ==> CSR
PROGRAM: ASS031 MODULE: ASS03 COMP DATE: 06/17/2005 COMP TIME: 02:11:03
** END **

----- ----- Before ASS031:69 <>
000060
000061      1000-READ .
000062
000063      READ EMP-FILE
000064      AT END
000065      MOVE 'Y' TO W01-EMP-EOF .
000066 B      IF INFO = EMP-DEPT
000067      DISPLAY EMP-REC .
000068
=====> 1000-EXIT .
000070
000071      EXIT .
000072
PF 1=HELP   2=PEEK  CSR  3=END      4=EXIT      5=FIND      6=LOCATE *
PF 7=UP     8=DOWN   9=GO 1    10=LEFT    11=RIGHT   12=GO
Tr [ ] R 2 C 15
Connected to 3.32.24.12:23
[ ] [ ] [NUM] [ ] 2, 15

```

Figure 71: Output

Step 41: Once the test is completed, you will get the following screen.

(A) Passport - PASSPORT

File Edit View Communication Options Transfer Macro Help

----- XPDITER/TSO - SOURCE -----

COMMAND ==> _ SCROLL ==> CSR

TEST COMPLETED

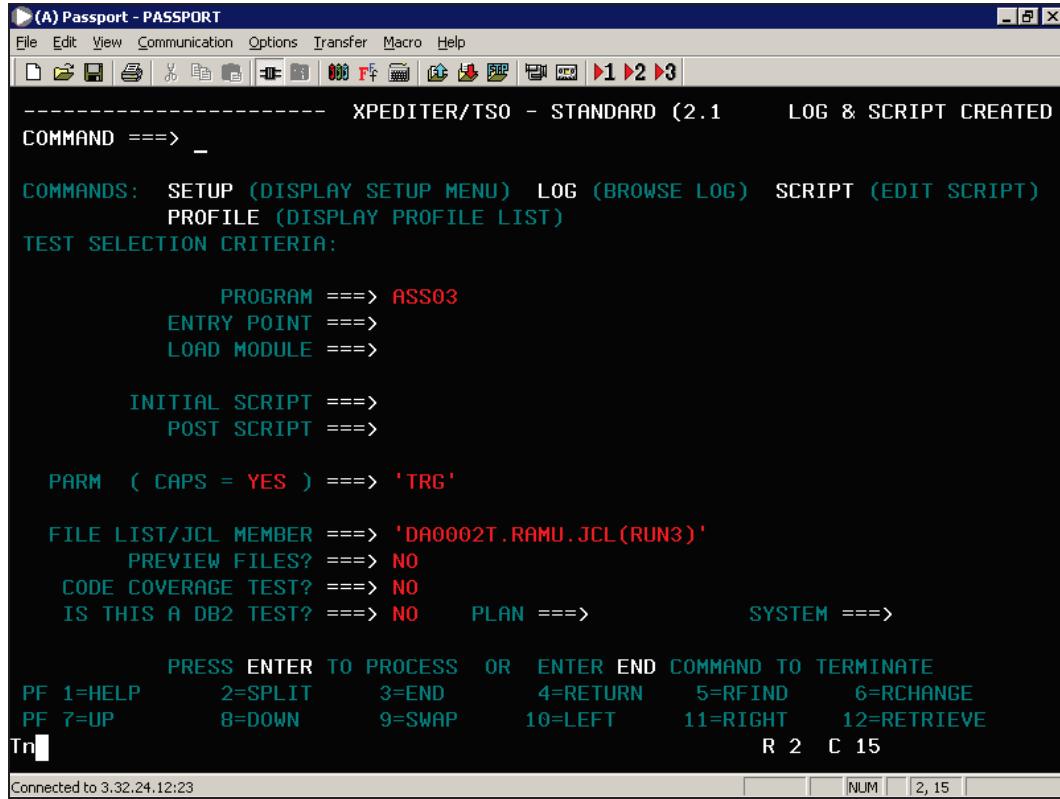
** END **

----- After ASS031 <>

```
000058      PERFORM 2000-TERM THRU 2000-EXIT
=====> A    STOP RUN.
000060
000061      1000-READ.
000062
000063      READ EMP-FILE
000064      AT END
000065      MOVE 'Y' TO W01-EMP-EOF.
000066 B    IF INFO = EMP-DEPT
000067      DISPLAY EMP-REC.
000068
000069      1000-EXIT.
000070
PF 1=HELP      2=PEEK CSR  3=END        4=EXIT        5=FIND        6=LOCATE *
PF 7=UP        8=DOWN       9=GO 1      10=LEFT      11=RIGHT     12=GO
Tn
```

Figure 72: Output

Step 42: Press **F9** to STOP the debugging and you will be on the following screen:



```

(A) Passport - PASSPORT
File Edit View Communication Options Transfer Macro Help
----- XPEDITER/TSO - STANDARD (2.1) LOG & SCRIPT CREATED
COMMAND ===> _

COMMANDS: SETUP (DISPLAY SETUP MENU) LOG (BROWSE LOG) SCRIPT (EDIT SCRIPT)
          PROFILE (DISPLAY PROFILE LIST)
TEST SELECTION CRITERIA:

      PROGRAM ===> ASS03
      ENTRY POINT ===>
      LOAD MODULE ===>

      INITIAL SCRIPT ===>
      POST SCRIPT ===>

PARM ( CAPS = YES ) ===> 'TRG'

FILE LIST/JCL MEMBER ===> 'DA0002T.RAMU.JCL(RUN3)'
      PREVIEW FILES? ===> NO
      CODE COVERAGE TEST? ===> NO
      IS THIS A DB2 TEST? ===> NO      PLAN ===>           SYSTEM ===>

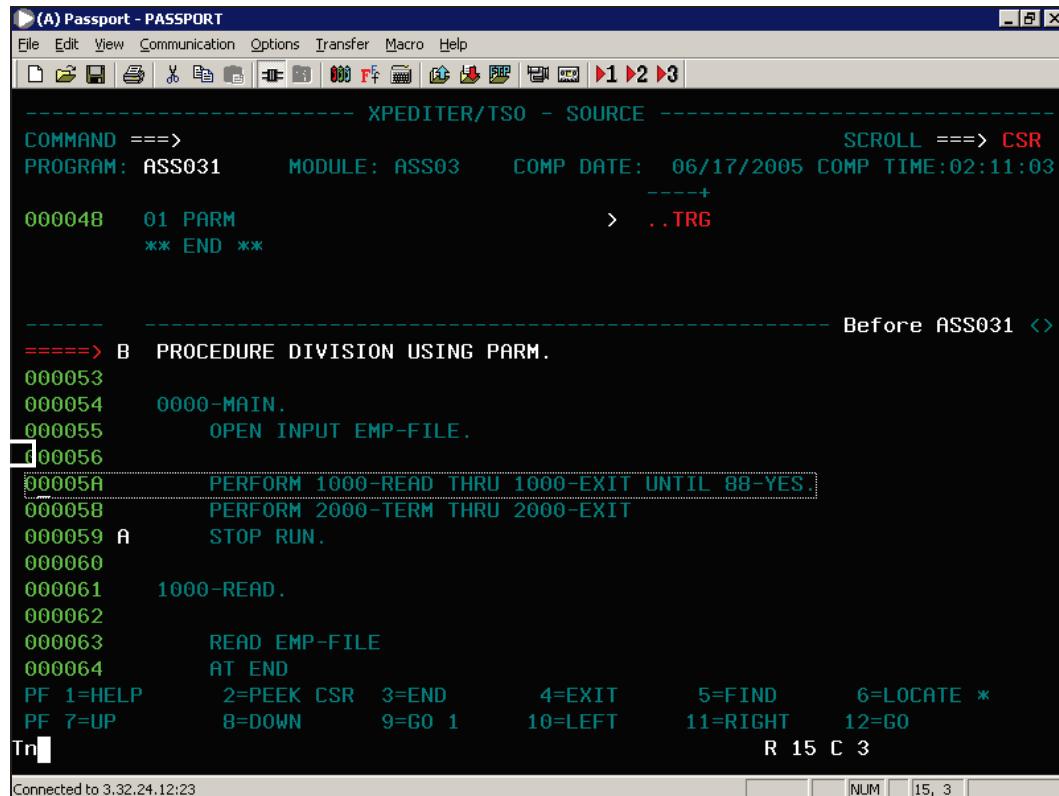
      PRESS ENTER TO PROCESS OR ENTER END COMMAND TO TERMINATE
PF 1=HELP    2=SPLIT    3=END    4=RETURN   5=RFIND    6=RCHANGE
PF 7=UP     8=DOWN     9=SWAP    10=LEFT    11=RIGHT   12=RETRIEVE
Tr[ ]                                     R 2 C 15

Connected to 3.32.24.12:23

```

Figure 73: Output

Step 43: To set **After Break Point** in a particular line, type **A** in the particular line number and press **ENTER**.



```

(A) Passport - PASSPORT
File Edit View Communication Options Transfer Macro Help
File Edit View Communication Options Transfer Macro Help
----- XPEDITER/TSO - SOURCE -----
COMMAND ===> SCROLL ===> CSR
PROGRAM: ASS031 MODULE: ASS03 COMP DATE: 06/17/2005 COMP TIME: 02:11:03
-----+
000048 01 PARM > ..TRG
** END **

-----+----- Before ASS031 <>
=====> B PROCEDURE DIVISION USING PARM.
000053
000054 0000-MAIN.
000055      OPEN INPUT EMP-FILE.
000056
000057 00005A      PERFORM 1000-READ THRU 1000-EXIT UNTIL 88-YES.
000058      PERFORM 2000-TERM THRU 2000-EXIT
000059 A      STOP RUN.
000060
000061 1000-READ .
000062
000063      READ EMP-FILE
000064      AT END
PF 1=HELP    2=PEEK CSR 3=END      4=EXIT      5=FIND      6=LOCATE *
PF 7=UP      8=DOWN     9=GO 1    10=LEFT     11=RIGHT   12=GO
Tr [ ] R 15 C 3
Connected to 3.32.24.12:23
[ ] [ ] NUM [ ] 15, 3

```

Figure 74: Output

Step 44: Once a breakpoint is placed on a line, the letter **A** appears in between the line number and the content on that line and you will get the status as “**1 COMMAND(S) COMPLETED**”.

(A) Passport - PASSPORT

File Edit View Communication Options Transfer Macro Help

COMMAND ==> - SCROLL ==> CSR

----- XPEDITER/TSO - SOURCE -----

1 COMMAND(S) COMPLETED

000048 01 PARM > ..TRG
*** END ***

----- Before ASS031 <>

=====> B PROCEDURE DIVISION USING PARM.

000053
000054 0000-MAIN.
000055 OPEN INPUT EMP-FILE.
000056
000057 A PERFORM 1000-READ THRU 1000-EXIT UNTIL 88-YES.
000058 PERFORM 2000-TERM THRU 2000-EXIT
000059 A STOP RUN.
000060
000061 1000-READ.
000062
000063 READ EMP-FILE
000064 AT END

PF 1=HELP PF 7=UP 2=PEEK CSR 3=END 4=EXIT 5=FIND 6=LOCATE *
B=DOWN 9=GO 1 10=LEFT 11=RIGHT 12=GO

Tn [] R 2 C 15

Connected to 3.32.24.12:23 NUM 2,15

Figure 75: Output

Step 45: The command, **Go n**, where n is an integer, is used to take the control 'n' lines forward from the current line. This will skip the line-by-line execution of in between lines and you can resume the same from the n^{th} line.

For example: In the following figure, the control is kept on the line 54, and **Go 2** command is issued.

(A) Passport - PASSPORT

File Edit View Communication Options Transfer Macro Help

COMMAND ==> GO 2 SCROLL ==> CSR

PROGRAM: ASS031 MODULE: ASS03 COMP DATE: 06/17/2005 COMP TIME: 02:11:03
*** END ***

----- Before ASS031:54 -----

```
000052 B PROCEDURE DIVISION USING PARM.  
000053  
=====> 0000-MAIN.  
000055      OPEN INPUT EMP-FILE.  
000056  
000057 A      PERFORM 1000-READ THRU 1000-EXIT UNTIL 88-YES.  
000058      PERFORM 2000-TERM THRU 2000-EXIT  
000059 A      STOP RUN.  
000060  
000061      1000-READ.  
000062  
000063      READ EMP-FILE  
000064      AT END  
PF 1=HELP      2=PEEK CSR  3=END      4=EXIT      5=FIND      6=LOCATE *  
PF 7=UP        8=DOWN     9=GO 1    10=LEFT     11=RIGHT    12=GO  
Tr [ ]          ↑          R 2 C 19  
Connected to 3.32.24.12:23
```

Figure 76: Output

Step 46: After the execution of **Go 2** command, the control has now shifted to line 57. The message "**2 STATEMENTS EXECUTED**" is displayed on the screen.

(A) Passport - PASSPORT

File Edit View Communication Options Transfer Macro Help

[File Open Save Print Find Go Macro Run] | 1 2 3

XPEDITER/TSO - SOURCE

COMMAND ===> - SCROLL ===> CSR

2 STATEMENTS EXECUTED

000037 01 W01-EMP-EOF > .
*** END ***

----- Before ASS031:57 <>

000052 B PROCEDURE DIVISION USING PARM.
000053
000054 0000-MAIN.
000055 OPEN INPUT EMP-FILE.
000056
=====> A PERFORM 1000-READ THRU 1000-EXIT UNTIL 88-YES.
000058 PERFORM 2000-TERM THRU 2000-EXIT
000059 A STOP RUN.
000060
000061 1000-READ.
000062
000063 READ EMP-FILE
000064 AT END
PF 1=HELP 2=PEEK CSR 3=END 4=EXIT 5=FIND 6=LOCATE *
PF 7=UP B=DOWN 9=G0 1 10=LEFT 11=RIGHT 12=G0

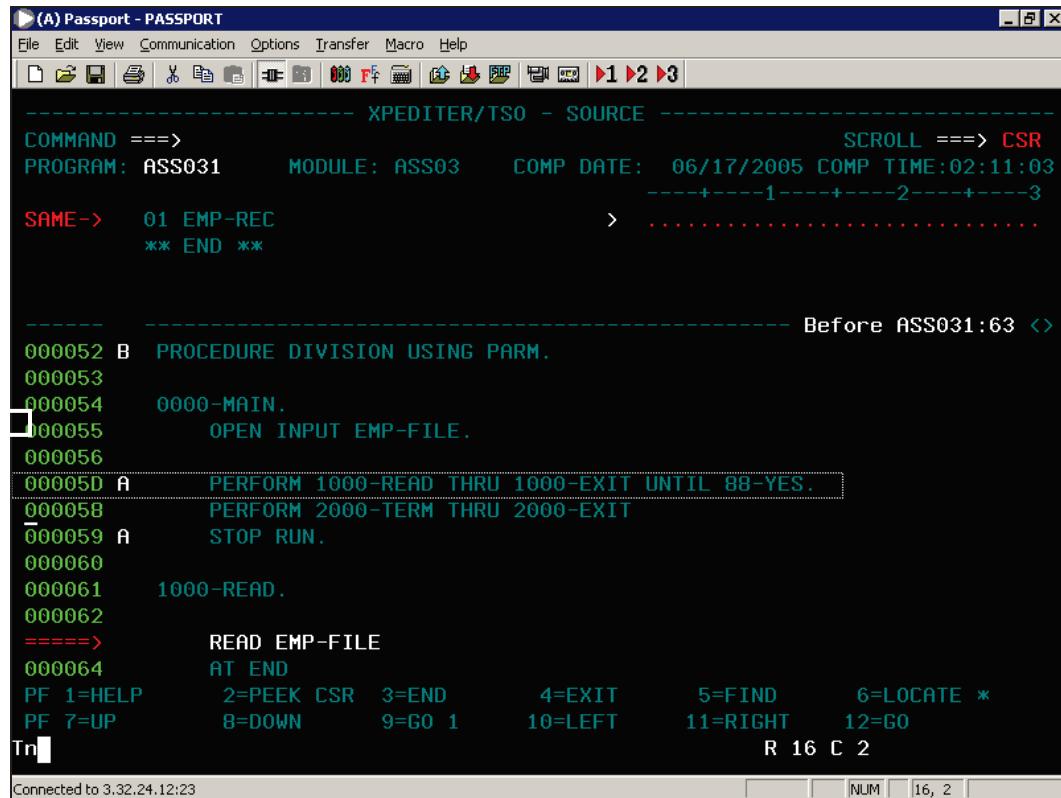
Tn [] ↑ R 2 C 15

Connected to 3.32.24.12:23 CAP NUM 2, 15

Figure 77: Output

Step 47: Use the delete command to clear the breakpoints. Take the cursor to the line from which you want to remove the breakpoint and type the letter 'D' at the beginning of that line to remove the breakpoint.

For example: The figure given below shows how the breakpoint from line 57 is removed.



```

(A) Passport - PASSPORT
File Edit View Communication Options Transfer Macro Help
[Icons] 1 2 3

----- XPEDITER/TSO - SOURCE -----
COMMAND ===> SCROLL ===> CSR
PROGRAM: ASS031 MODULE: ASS03 COMP DATE: 06/17/2005 COMP TIME: 02:11:03
-----+---1---+---2---+---3
SAME-> 01 EMP-REC > .....
** END **

----- Before ASS031:63 <>
000052 B PROCEDURE DIVISION USING PARM.
000053
000054     0000-MAIN.
000055         OPEN INPUT EMP-FILE.
000056
000057 D     PERFORM 1000-READ THRU 1000-EXIT UNTIL 88-YES.
000058         PERFORM 2000-TERM THRU 2000-EXIT
000059 A     STOP RUN.
000060
000061     1000-READ.
000062
=====>     READ EMP-FILE
000064     AT END
PF 1=HELP    2=PEEK CSR  3=END      4=EXIT      5=FIND      6=LOCATE *
PF 7=UP      8=DOWN       9=GO 1    10=LEFT     11=RIGHT    12=GO
Tn[ ] R 16 C 2

Connected to 3.32.24.12:23
[Icons] NUM 16, 2

```

Figure 78: Output

Step 48: The message “**DELETE COMMAND PROCESSED**” is flashed after removing the breakpoint. Now note the breakpoint is removed in the particular line.

(A) Passport - PASSPORT

File Edit View Communication Options Transfer Macro Help

COMMAND ===> _ SCROLL ===> CSR

XPEDITER/TSO - SOURCE

DELETE COMMAND PROCESSED

SAME-> 01 EMP-REC > 1-----2-----3
** END **

----- Before ASS031:63 <>

000052 B PROCEDURE DIVISION USING PARM.

000053

000054 0000-MAIN.

000055 OPEN INPUT EMP-FILE.

000056

000057 PERFORM 1000-READ THRU 1000-EXIT UNTIL 88-YES.

000058 PERFORM 2000-TERM THRU 2000-EXIT

000059 A STOP RUN.

000060

000061 1000-READ.

000062

=====> READ EMP-FILE

000064 AT END

PF 1=HELP 2=PEEK CSR 3=END 4=EXIT 5=FIND 6=LOCATE *

PF 7=UP 8=DOWN 9=GO 1 10=LEFT 11=RIGHT 12=GO

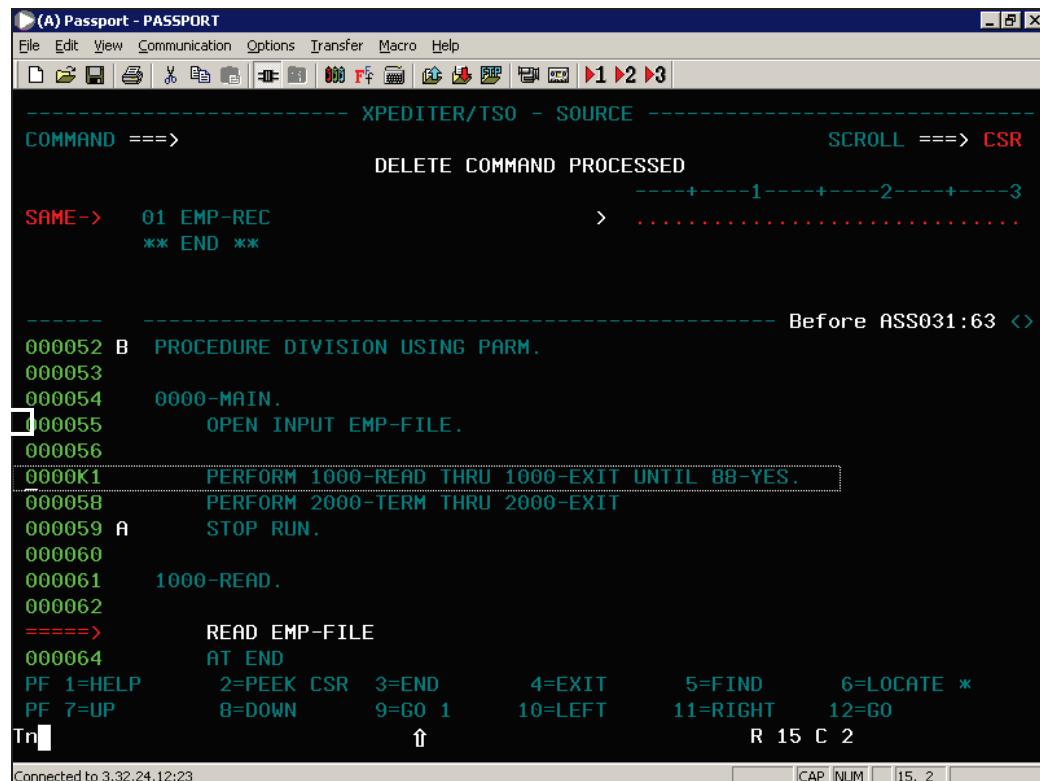
Tn T 2 C 15

Connected to 3.32.24.12:23

Figure 79: Output

Step 49: The **Keep** Command is used to keep a watch on values of variables. Enter the letter 'K' on a line in which the variable you want to watch is present.

For example: In figure given below, **K1** is entered on the line 57 indicating that a watch should be placed on the variable associated with **88-YES**.



```
(A) Passport - PASSPORT
File Edit View Communication Options Transfer Macro Help
| D F G H I J L R C 1 2 3 |
----- XPEDITER/TSO - SOURCE -----
COMMAND ===> SCROLL ===> CSR
      DELETE COMMAND PROCESSED
      -----+-----1-----+-----2-----+-----3
SAME-> 01 EMP-REC > .....
*** END **

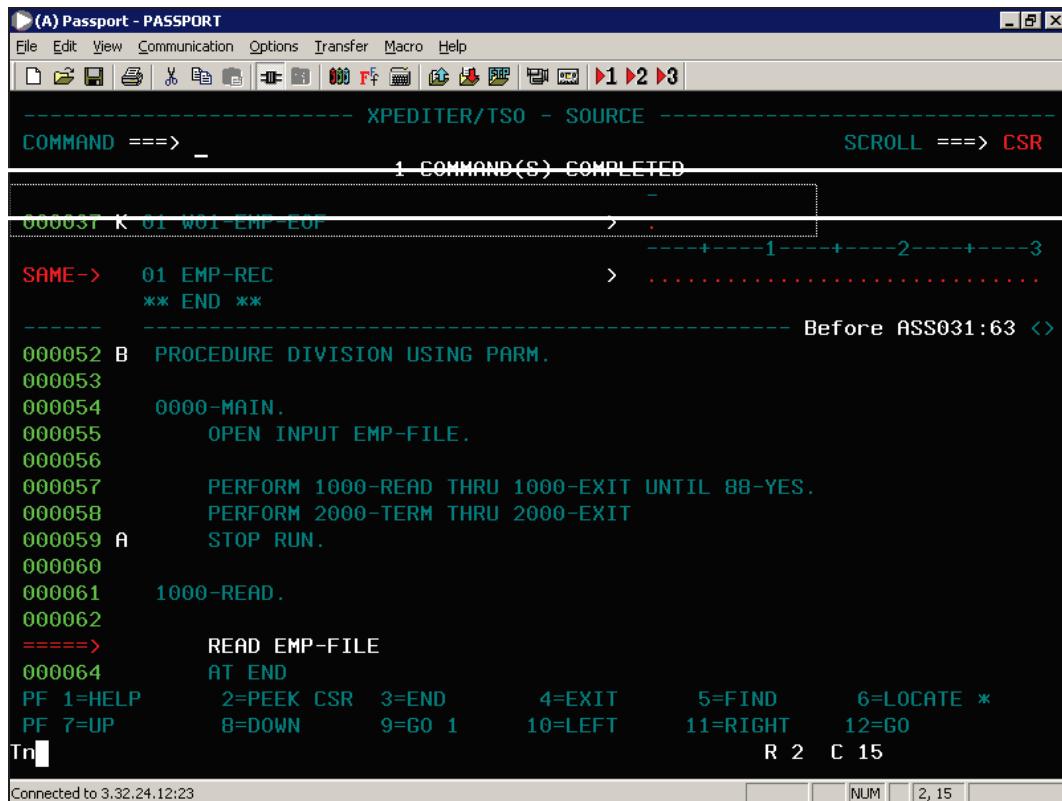
----- Before ASS031:63 <>
000052 B PROCEDURE DIVISION USING PARM.
000053
000054     0000-MAIN.
000055         OPEN INPUT EMP-FILE.
000056
000057
000058         PERFORM 1000-READ THRU 1000-EXIT UNTIL 88-YES.
000059         PERFORM 2000-TERM THRU 2000-EXIT
000060         STOP RUN.
000061
000062     1000-READ.
000063
000064         READ EMP-FILE
000065         AT END
PF 1=HELP    2=PEEK CSR  3=END      4=EXIT      5=FIND      6=LOCATE *
PF 7=UP      8=DOWN       9=GO 1    10=LEFT    11=RIGHT   12=GO
Tn[ ]          ↑           R 15 C 2

Connected to 3.32.24.12:23
[CAP] [NUM] [15, 2]
```

Figure 80: Output

Note: K1 indicates the first (1) variable in the line has to be kept on the watch window. General Syntax is **Kn**, where n indicates the position of the variable in a particular line.

Step 50: Observe here that the variable **W01-EMP-EOF** is placed in the **KEEP** window and you can monitor its value. The highlighted area is the **KEEP** window, where you can monitor the values of the variable at any point of time.



The screenshot shows the Xpediter/TSO - SOURCE window. The window title is '(A) Passport - PASSPORT'. The menu bar includes File, Edit, View, Communication, Options, Transfer, Macro, and Help. The toolbar contains various icons. The status bar at the bottom left says 'Connected to 3.32.24.12:23' and the bottom right shows 'R 2 C 15'.

```

----- XPEDITER/TSO - SOURCE -----
COMMAND ==> -
1 COMMAND(S) COMPLETED

000031 K 01 W01-EMP-EOF > .-----+---1---+---2---+---3
SAME-> 01 EMP-REC > .....----- Before ASS031:63 <>
000052 B PROCEDURE DIVISION USING PARM.
000053
000054     0000-MAIN.
000055         OPEN INPUT EMP-FILE.
000056
000057         PERFORM 1000-READ THRU 1000-EXIT UNTIL BB-YES.
000058         PERFORM 2000-TERM THRU 2000-EXIT
000059 A     STOP RUN.
000060
000061     1000-READ.
000062
=====>     READ EMP-FILE
000064     AT END
PF 1=HELP   2=PEEK CSR  3=END      4=EXIT      5=FIND      6=LOCATE *
PF 7=UP     8=DOWN       9=GO 1    10=LEFT     11=RIGHT    12=GO
Tn[ ]                                     R 2 C 15

```

The variable **W01-EMP-EOF** is highlighted in the first line of the code. The status bar at the bottom left says 'Connected to 3.32.24.12:23' and the bottom right shows 'R 2 C 15'.

Figure 81: Output

Step 51: The value of a variable can be changed during the execution by using the “**MOVE new-value TO variable-name**” command.

(A) Passport - PASSPORT

File Edit View Communication Options Transfer Macro Help

COMMAND ==> MOVE 'RND' TO INFO. SCROLL ==> CSR

PROGRAM: ASS031 MODULE: ASS03 COMP DATE: 06/17/2005 COMP TIME: 02:11:03

000050 05 INFO > TRG

000028 05 EMP-DEPT > TRG
** END **
----- Before ASS031:66 <>
000065 MOVE 'Y' TO W01-EMP-EOF.
=====> IF INFO = EMP-DEPT
000067 DISPLAY EMP-REC.
000068
000069 1000-EXIT.
000070
000071 EXIT.
000072
000073 2000-TERM.
000074
000075 CLOSE EMP-FILE.
000076
000077 2000-EXIT.
PF 1=HELP 2=PEEK CSR 3=END 4=EXIT 5=FIND 6=LOCATE *
PF 7=UP 8=DOWN 9=GO 1 10=LEFT 11=RIGHT 12=GO
Tn [] ↑ R 2 C 34

Connected to 3.32.24.12:23 CAP NUM 2, 34

Figure 82: Output

Step 52: Notice that the value of the variable **INFO** is now **RND** and the same is displayed in the **KEEP** window.

(A) Passport - PASSPORT

File Edit View Communication Options Transfer Macro Help

[File Open Save Print Find Go Back Forward Home] [1 2 3]

= XEDITTER/TSO - SOURCE =

COMMAND ==>	SCROLL ==> CSR		
PROGRAM: ASS031	MODULE: ASS03	COMP DATE: 06/17/2005	COMP TIME: 02:11:03

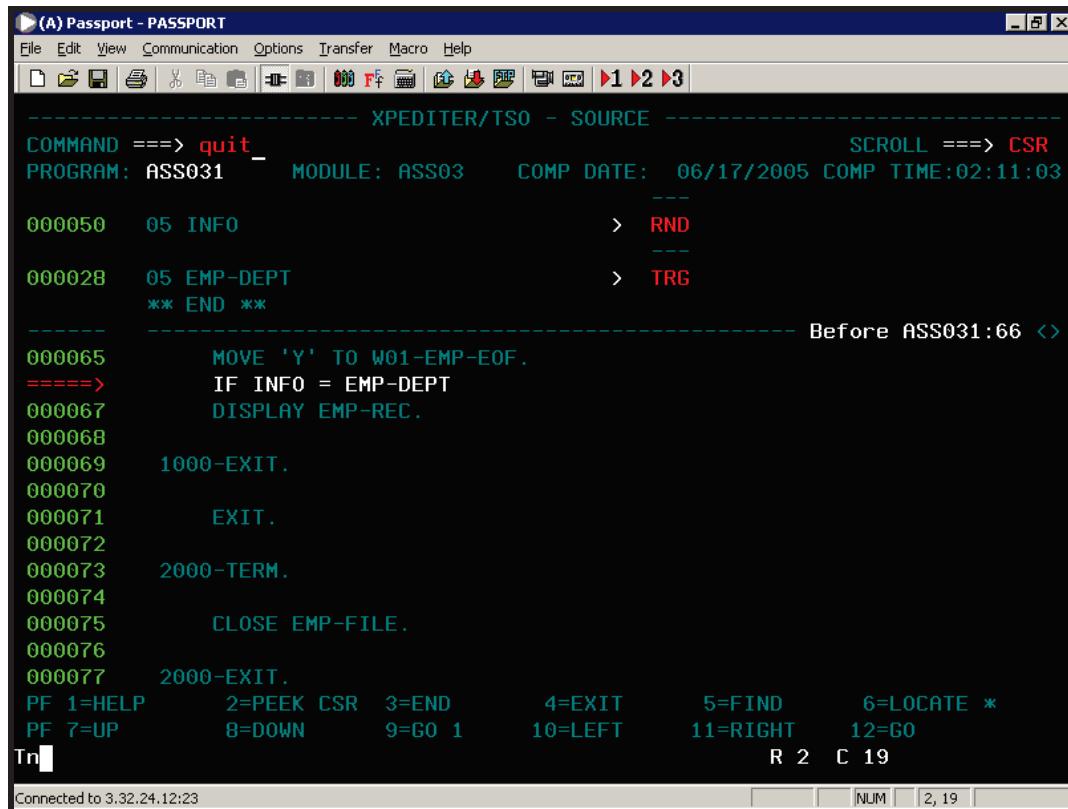
```
000050 05 INFO > RND
000028 05 EMP-DEPT > TRG
*** END ***
000065 MOVE 'Y' TO W01-EMP-EOF.
=====> IF INFO = EMP-DEPT
000067 DISPLAY EMP-REC.
000068
000069 1000-EXIT.
000070
000071 EXIT.
000072
000073 2000-TERM.
000074
000075 CLOSE EMP-FILE.
000076
000077 2000-EXIT.
PF 1=HELP 2=PEEK CSR 3=END 4=EXIT 5=FIND 6=LOCATE *
PF 7=UP 8=DOWN 9=GO 1 10=LEFT 11=RIGHT 12=GO
```

Tn [] R 2 C 15

Connected to 3.32.24.12:23

Figure 83: Output

Step 53: To quit the Xpediter, type QUIT.



```

----- XPEDITER/TSO - SOURCE -----
COMMAND ==> quit                                SCROLL ==> CSR
PROGRAM: ASS031      MODULE: ASS03     COMP DATE: 06/17/2005 COMP TIME:02:11:03
                                                     ---
000050    05 INFO                               > RND
                                                     ---
000028    05 EMP-DEPT                          > TRG
** END **

----- Before ASS031:66 <>
000065      MOVE 'Y' TO W01-EMP-EOF.
=====>      IF INFO = EMP-DEPT
000067          DISPLAY EMP-REC.
000068
000069      1000-EXIT.
000070
000071      EXIT.
000072
000073      2000-TERM.
000074
000075      CLOSE EMP-FILE.
000076
000077      2000-EXIT.
PF 1=HELP   2=PEEK CSR  3=END      4=EXIT      5=FIND      6=LOCATE *
PF 7=UP     8=DOWN     9=GO 1    10=LEFT    11=RIGHT   12=GO
R 2 C 19

Tn
Connected to 3.32.24.12:23

```

Figure 84: Output

Step 54: Quitting the **Xpediter** takes you to the **XPEDITER/TSO – STANDARD** menu screen. Press **F3** key to return to the **XPEDITER/TSO – PRIMARY** menu.

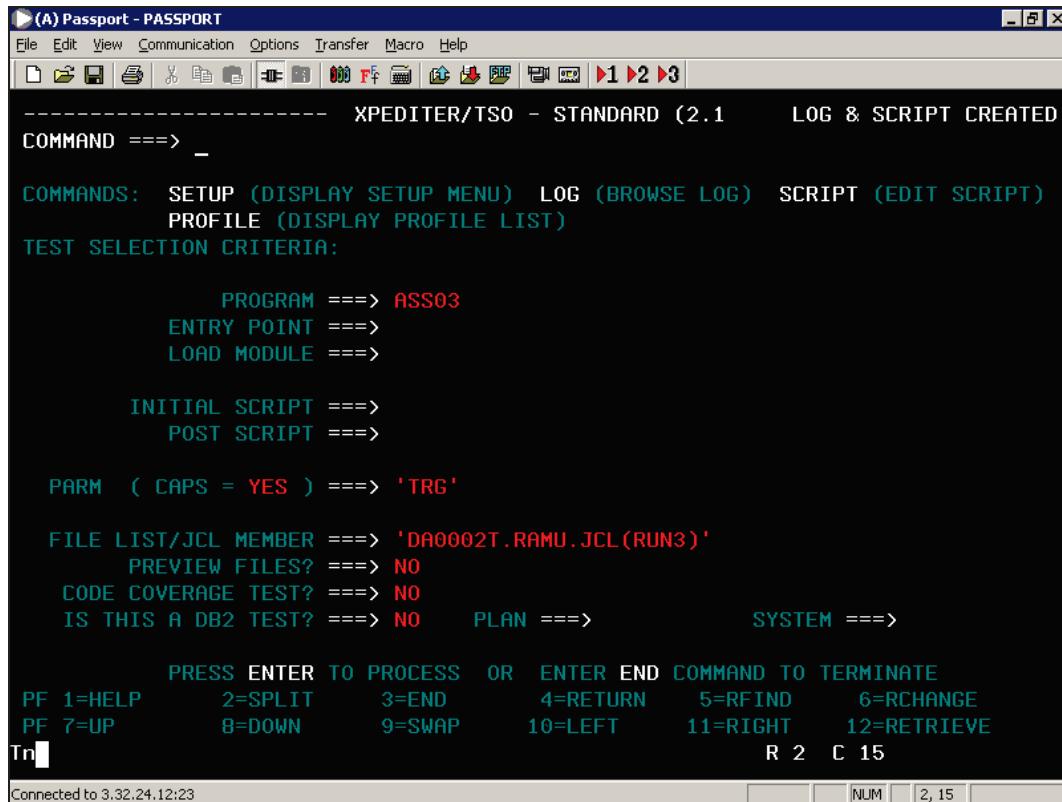


Figure 85: Output

Step 55: Press F3 key or X to return to the ISPF Primary option menu.

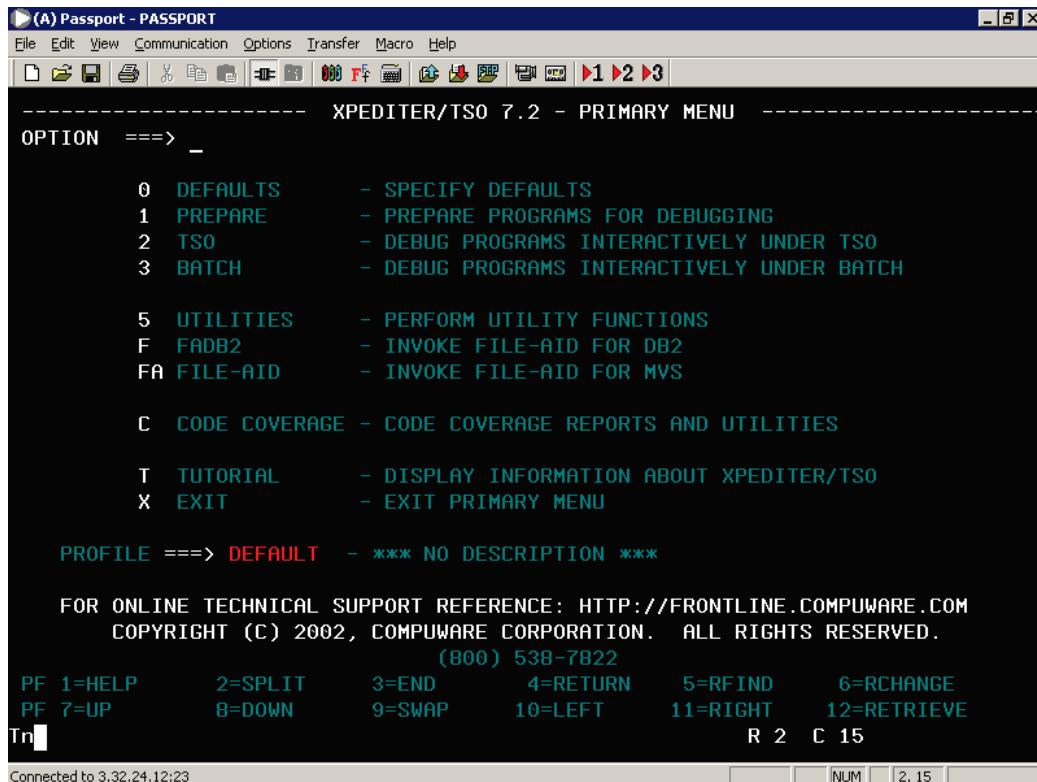


Figure 86: Output