Burroughs

Student Guide

A Series And B 5/6/7000 Work Flow Language And Utilities Mark 3.6 EP 4386

March 1986

"The names used in this publication are not of individuals living or otherwise. Any similarity of likeness of the names used in this publication with the names of any individual living or otherwise is purely coincidental and not intentional."

Burroughs believes that the information described in this publication is accurate and reliable, and much care has been taken in its preparation. However, no responsibility, financial or otherwise, can be accepted for any consequences arising out of the use of this material, including loss of profit, indirect, special, or consequential damages. There are no warranties which extend beyond the program specification.

The customer should exercise care to assure that use of information in this publication will be in full compliance with laws, rules, and regulations of the jurisdiction with respect to which it is used.

The information contained herein is subject to change. Revisions may be issued from time to time to advise of changes and/or additions.

Correspondence regarding this document should be forwarded directly to Burroughs Corporation, Room 205, Education Development, 2611 Corporate West Drive, Lisle, IL 60532-3697.

A SERIES AND B 5/6/7000 WORKFLOW AND UTILITIES

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

INTRODUCTION

COURSE OBJECTIVE

Design, create, implement, and maintain job schedules and manipulate files on a Burroughs large system using Work Flow Language.

COURSE GOALS

After completion of this course, you will have acquired the necessary knowledge and skills to:

- Create and execute a job stream.
- Utilize the job summary.
- Define job attributes which interface with job queues.
- Process tasks asynchronously.
- Provide control of printer backup facilities.
- Utilize terminal related system utility programs.

AUDIENCE

This course is directed toward programmers, analysts, and senior operators with some programming experience who work through terminals or remote computing systems.

PREREQUISITES

You should have the equivalent understanding of, or should have completed, Introduction to Large Systems and Introduction to A Series CANDE. A working knowledge of CANDE is necessary and programming experience is helpful. You will be responsible for your own data entry requirements for the WFL source.

COURSE LENGTH

The course length is approximately 40 hours.

COURSE MODE

The course will be instructor-led, usually following the sequence as listed in the Table of Contents. This course will involve extensive practice of the skills used in creating and implementing Work Flow jobs.

Classroom sessions will be held to give background information and lab sessions will provide hands on experience. It is expected that the learner will utilize reference materials to supplement the classroom lectures.

GENERAL LAB INFORMATION

1. Sign on to CANDE.
   Use a USERCODE/PASSWORD pair provided by your instructor.

   USERCODE _____

   PASSWORD _____

   If a DESTINATION NAME is required, it will be provided.

   DESTNAME _____

2. Your Work Flow source should be of type JOB.

3. All files created, either through CANDE or your Work Flow jobs, should have a first name of your initials and a family name provided by your instructor unless otherwise stated.

   FAMILY _____

4. Start your Work Flow jobs through CANDE using the START command.

5. Sign off CANDE and MARC when finished with your lab assignment each day.

SECTION 1

TASK INITIATION AND CONTROL

TASK INITIATION AND CONTROL

SECTION OBJECTIVE

Construct task initiation and control statements.

PURPOSE

This section provides an introduction to the Work Flow Language (WFL) and the Work Flow process. It will also provide information on the initiation of and control of tasks while executing under WFL.

UNIT OBJECTIVES

- Identify the elements of the Work Flow process and related terms.
- Identify and describe the five sections of a Work Flow job.
- Construct simple task initiation statements.
- Construct basic task control and communication statements.

TASK INITIATION AND CONTROL

Unit 1

Work Flow Language and Process

OBJECTIVE

Identify the elements of the Work Flow process and related terms.

PURPOSE

In order to write effective WFL source programs, it is necessary to have some understanding of the Work Flow Management system, its terminology, its features, and its performance.

RESOURCES

a. Student Guide	Section 1 Unit 1
b. A Series WFL Reference Manual	Section 1

KEY WORDS

Task
Job Task
Dependent Task
Independent Task
Synchronous Task
Asynchronous Task
Job
Mix Numbers
Job Stream
Job Queue
WFL Compiler
Jobfile
WORK FLOW MANAGEMENT SYSTEM
CONTROLLER
JOBDESC
MCPHOST
AUTOBACKUP
Job Summary

SECTION 1: TASK INITIATION AND CONTROL

UNIT 1: Work Flow Language and Process

LEARNING SEQUENCE

WORK FLOW TERMINOLOGY

A **task** is the execution of a program. Tasks can be divided into three types: job task, dependent task, and independent task.

- A **job task** is the execution of a program written in Work Flow Language.
- A **dependent task** is any task initiated by a job task or called by another task.
- An **independent task** is any task that is initiated from a task and is no longer under the control of the originating task.
- Tasks may be run either synchronously or asynchronously.
  - **Synchronous tasks** are those that execute serially. This means that the next task cannot start until the one executing finishes.
  - **Asynchronous tasks** are those that run concurrently in a multiprogramming environment.

TASK EXAMPLES:

An execution of a job	% Job Task
A program execution	% Dependent Task
A program compilation	
A file copy from tape or disk	
A job starting another job	% Independent Task

A **job** is a collection of one or more related tasks, including the job task and all the dependent tasks initiated by tasks in the job. The system keeps track of these tasks by assigning them numbers. These numbers are called **mix numbers**.

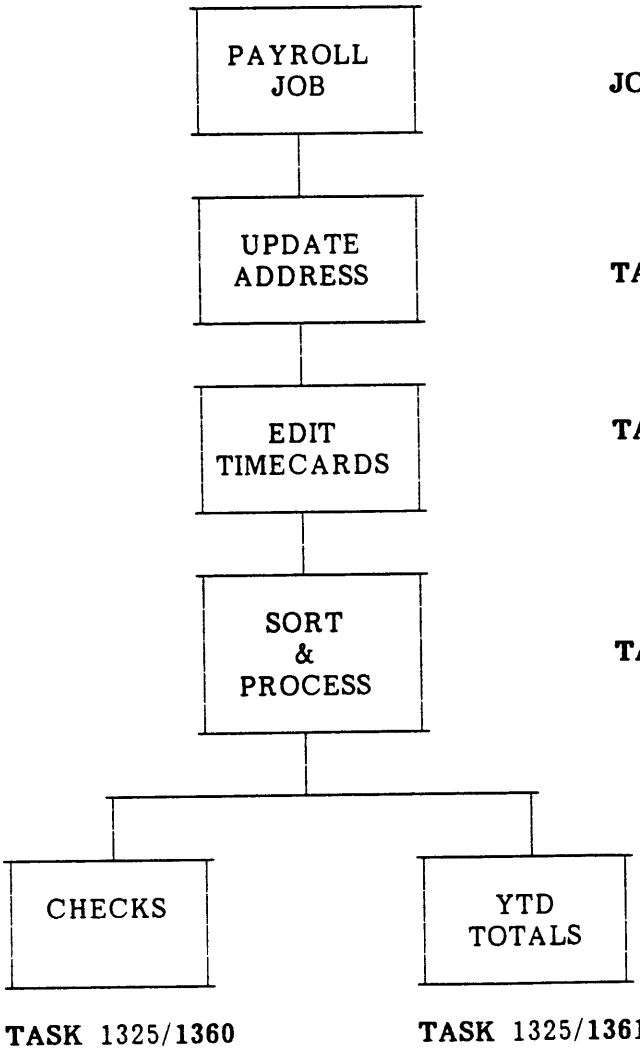
SECTION 1: TASK INITIATION AND CONTROL

UNIT 1: Work Flow Language and Process

LEARNING SEQUENCE

JOB EXAMPLE:

```
PAYROLL JOB
  UPDATE ADDRESS PROGRAM
  EDIT TIMECARD PROGRAM
  SORT & PROCESS PROGRAM
  PRINT PROGRAMS
    CHECKS
    YTD TOTALS
```



SECTION 1: TASK INITIATION AND CONTROL

UNIT 1: Work Flow Language and Process

LEARNING SEQUENCE

A **job stream** is the order of tasks to be executed within the job, sometimes referred to as just the job.

A **job queue** is a place to store jobs that are to be executed. The system provides the means for controlling job execution based on the attributes assigned to that job and attributes of the queue. Such items as priority, available system resources, and limits imposed by the operational staff are used in these decisions.

WORK FLOW FEATURES

The Work Flow Language (WFL) is a high level language in which jobs are described. It is designed to control, execute, and monitor the flow of tasks within a job stream.

WFL is the means by which a job is described and presented to the system for execution. The language allows the user to programmatically control the execution of a set of interrelated tasks. A job may decide, at run-time, whether to run a program, which programs to run, and in what order to run them. The printed and punched output for all tasks within the job is retained and output as a group at the end of the job. As part of that output, the log information generated for that job is output at the same time. Comprehensive billing information is accumulated to be used if desired.

WFL source statements are English-like and, with a few exceptions, are free-format. The **WFL compiler** syntactically checks and compiles all control statements into machine code and outputs a disk file referred to as a **jobfile**. This jobfile is then linked to a job queue for scheduling.

SECTION 1: TASK INITIATION AND CONTROL

UNIT 1: Work Flow Language and Process

LEARNING SEQUENCE

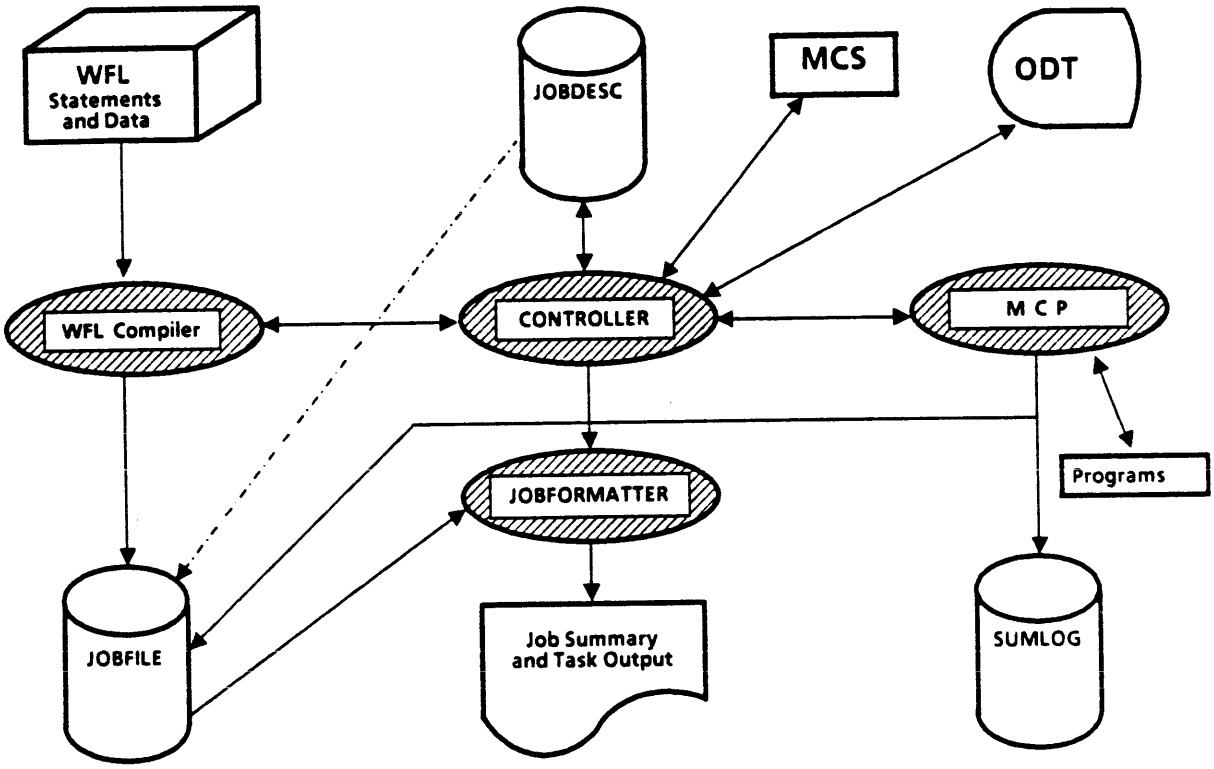
WORK FLOW MANAGEMENT SYSTEM

When a Work Flow job is started, the selected source statements are given to the WFL compiler. After a successful compilation, the system will attempt to link the job file to an appropriate queue where the job will wait for selection to be executed. If an appropriate queue is not available, the job will be discontinued. Upon selection, the system must check if enough memory is available. If not, the job is scheduled; otherwise, the job is loaded into memory and begins execution. When all of the tasks within the job have been completed, the printed or punched output from the tasks within the job is released to the printer backup facility. The output includes the Job Summary, the WFL source, the WFL Log, and the dependent tasks printed or punched output.

SECTION 1: TASK INITIATION AND CONTROL

UNIT 1: Work Flow Language and Process

LEARNING SEQUENCE



Shaded areas represent parts of the MCP.

Work Flow Management System Organization

SECTION 1: TASK INITIATION AND CONTROL

UNIT 1: Work Flow Language and Process

LEARNING SEQUENCE

WORK FLOW MANAGEMENT SYSTEM consists of several parts of the MCP represented by the shaded areas on the preceding page.

- The **WFL compiler** generates the job by:
  - checking the syntax of the WFL source;
  - translating the job control statements to machine code;
  - storing the machine code, WFL source, any data decks used, and space for logging and restart information in a special type of file called a jobfile.

- The **CONTROLLER** controls the system work load, queue level scheduling, and operator communications by:
  - maintaining the Jobfile Description File (**JOBDESC**) that contains
    - directories of jobfiles
    - queue for controller scheduling
    - links to the jobfiles
    - Job Queue Heads, number of queues, and pointers to the individual queues
  - choosing the queue to which the job will be linked;
  - inserting the job at the appropriate location in the queue;
  - selecting the job to execute next;
  - removing the job from the job queue;
  - passing the addresses of the diskfile header and Program Parameter Block (HDR/PPB) to JOBSTARTER.

- The **MCPHOST** includes
  - building part of the **Process Information Block (PIB);**
  - calling additional routines to finish building the PIB, allocate memory, build the process stack, and place it in the ready queue to be executed;
  - indirectly placing a message in the Controller's queue that will cause the Controller to initiate AUTOBACKUP.

- The **AUTOBACKUP** facility % On 3.6 the Printing Subsystem
  - formats the **job summary** for printing;
  - formats the WFL source statements for printing;
  - prints or punches job output;
  - removes the jobfile.

SECTION 1: TASK INITIATION AND CONTROL

UNIT 1: Work Flow Language and Process

PRACTICE EXERCISE

A. Please complete the following statements by filling in the blank.

1. A _____ consists of a collection of related _____.

2. The _____ creates the WFL code file and stores it in a _____.

3. As a job runs, the MCP continually stores log information for that job in the _____ file maintained by the _____.

4. (True or False) A WFL code file will remain on disk after a WFL job is completed. _____

5. (True or False) Tasks within a job may be run one at a time or at the same time. _____

PRACTICE EXERCISE continued

B. Identify the sections/blocks of the Work Flow Management system by selecting the letter that represents the program unit or procedure corresponding to that function.

Function	Unit/Procedure
_____ 1. This unit syntactically checks the job and translates job control statements into machine language code.	A. MCPHOST
_____ 2. This file contains object code, WFL source, data decks, logging and restart information.	B. AUTOBACKUP
_____ 3. This program unit is fired-up by the MCP at halt/load time and runs as an independent runner.	C. JOBDESC
_____ 4. This file is maintained by the controller and has entries that consist of file headers and links used to organize the jobs by class and priority.	D. JOBFILE
_____ 5. This section chooses the appropriate class by matching requirements of the job with the specifications of the various queues.	E. CONTROLLER
_____ 6. This section chooses a job to be started by paying attention to the queue priorities, mix limits, and also assigns attributes not explicitly set by the job.	F. WFL COMPILER
_____ 7. This section is notified by the Controller to transfer the job entry to the proper task/stack structure to be run.	
_____ 8. This section is called after receiving notice of EOJ from the MCP and formats the job summary and WFL source statements for printing.	

BLANK PAGE

TASK INITIATION AND CONTROL

Unit 2

Job Structure

OBJECTIVE

Identify and describe the five sections of a Work Flow job.

PURPOSE

To write effective WFL jobs, it is necessary to understand how the language is structured.

RESOURCES

a. Student Guide	Section 1 Unit 2
b. A Series WFL Reference Manual	Section 3

KEY WORDS

BEGIN JOB
Job Attributes
Declarations
Statements
END JOB

SECTION 1: TASK INITIATION AND CONTROL

UNIT 2: Job Structure

LEARNING SEQUENCE

Work Flow Job:

The Work Flow (WFL) job controls the execution of the dependent tasks initiated from it. Statements may be written in a free format structure within each of the sections. All statements must end with a semi-colon (;). In order to describe the WFL job structure, the job is divided into five sections; three of these are mandatory, (BEGIN JOB, STATEMENTS, END JOB) and must appear in the correct order. The job attributes and declarations sections are optional and allow additional control capabilities.

SECTION	WFL PROGRAM STATEMENTS
BEGIN JOB SECTION	BEGIN JOB MYJOB ;
JOB ATTRIBUTES (optional)	QUEUE = 20 ;
DECLARATIONS (optional)	TASK T1 ;
STATEMENTS	RUN PROGA ;
END JOB SECTION	END JOB

SECTION 1: TASK INITIATION AND CONTROL

UNIT 2: Job Structure

LEARNING SEQUENCE

BEGIN JOB

This is the first section and must begin with the reserved words **BEGIN JOB** and may have a job title, job parameters list and a job disposition. The job title is a file title and is defined in the WFL Manual. The job parameters are named identifiers and may be of the type Boolean, Integer, Real, or String. The job disposition is usually used to compile the job and check for syntax errors.
(See Section 3 of the WFL Reference Manual)

JOB ATTRIBUTES

Task attributes are used to control the environment and behavior of a task before, during, and after execution of the task. These attributes, when used at the job level, are known as **Job Attributes** and are used to assign attributes to the job such as USERCODE, PRIORITY, QUEUE, and MAXLINES.
(See Section 3 of the WFL Reference Manual)

DECLARATIONS

Declarations are Work Flow constructs that define variables by type and express their intended use. Each variable used in a job must be declared via a declaration statement before it can be used. There are seven variable types: File, Task, Real, String, Boolean, Integer, and Subroutine.
(See Section 4 of the WFL Reference Manual)

STATEMENTS

The **statements** section is the working section of the job and consists of WFL statements. A statement is a combination of basic elements, constructs, and commands that can be used to initiate or control a process. A statement may also be used to assign values to previously declared variables. Each statement ends with a semicolon (;).
(See Section 6 of the WFL Reference Manual)

END JOB

The reserved words **END JOB** designate the end of the job. It is the last section and is required.

SECTION 1: TASK INITIATION AND CONTROL

UNIT 2: Job Structure

LEARNING SEQUENCE

BEGIN JOB	JOB NAME
JOB ATTRIBUTES	QUEUE or CLASS USERCODE DESTNAME FAMILY TASK ATTRIBUTES STARTTIME
DECLARATIONS	REAL INTEGER STRING BOOLEAN TASKS SUBROUTINES GLOBAL FILES
STATEMENTS	TASK INITIATION TASK CONTROL FLOW CONTROL COMPOUND SUBROUTINE CONTROL FILE HANDLING FILE MANAGEMENT TASK SECURITY COMMUNICATION CATALOGING VALUE ASSIGNMENT
END JOB	

SECTION 1: TASK INITIATION AND CONTROL

UNIT 2: Job Structure

PRACTICE EXERCISE

A. List the five sections of a Work Flow job including a brief description of each.

1.

2.

3.

4.

5.

B. Which, if any, are the optional sections of a Work Flow job?

BLANK PAGE

TASK INITIATION AND CONTROL

Unit 3

Task Initiation

OBJECTIVE

Construct simple task initiation statements.

PURPOSE

Once the purpose of WFL is known and the basic structure of the language understood, it is necessary to begin constructing some simple statements to accomplish the purpose of a WFL job.

RESOURCES

a. Student Guide	Section 1 Unit 3
b. A Series WFL Reference Manual	Section 5 Section 6

KEY WORDS

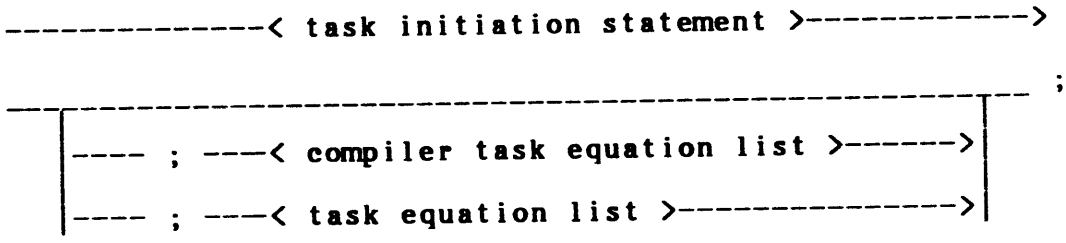
COMPILE
RUN
COPY
ADD
START
PROCESS
LOG
PB
SCR
PTD

SECTION 1: TASK INITIATION AND CONTROL

UNIT 3: Task Initiation

LEARNING SEQUENCE

A task initiation statement is used to start user programs and system functions as separate dependent tasks. Some simple task initiation statements are COMPILE, RUN, START, and COPY. These task initiation statements run synchronously; that is, one task must complete before the next task is started.



The **COMPILE** statement is used to initiate a compiler, which uses a source program to generate an object program.

- COMPILE PROGA COBOL74 LIBRARY ;
- COMPILE PROGB/OBJ WITH FORTRAN SYNTAX ;
- COMPILE PROGC ON USER1 ALGOL LIBRARY GO ;

The **RUN** statement is used to initiate an object program.

- RUN PROGA ;
- RUN PROGB ON USER1 ;

The **COPY** and **ADD** statements allow the capability to do file maintenance through WFL.

- COPY A AS B ;
- COPY A/=, B/= TO BKUP ; % (KIND = TAPE)
- ADD COBOL/= FROM DEVELOPMENT ; % (KIND = TAPE)
- ADD Z/= FROM T TO R (KIND=PACK), TO DISK ;
  % (KIND = DISK)

SECTION 1: TASK INITIATION AND CONTROL

UNIT 3: Task Initiation

LEARNING SEQUENCE

The **START** statement is used to initiate another job that is independent of the requesting task. The START statement initiates the WFL compiler as a synchronous dependent task. After the compiler finishes, the resultant WFL program is initiated as a job task. The original job waits for the WFL compiler, but not for the job task to complete, before continuing to the next statement.

The started job is independent of the job that started it and runs according to its own attributes. If the started job does not have a usercode job attribute, the usercode of the job that performed the start is used.

- START INVENTORY ;
- START PAYROLL ON PAYPACK ;
- START AJOB; STARTTIME = 12:02 ;

The **PROCESS** statement initiates a task asynchronously and can be used with any other task initiation statement. The job will not terminate, however, until all asynchronous tasks have terminated.

- PROCESS RUN PROGA ;
- PROCESS COPY A AS (WFL)A ;

Other task initiation statements are **LOG**, **PB**, **SCR** and PTD.

- LOG OPERATOR AX ;
- PB "PRINTOUT/ONE" COPIES 3 SAVE ;
- SCR ;
- PTD ;

SECTION 1: TASK INITIATION AND CONTROL

UNIT 3: Task Initiation

LEARNING SEQUENCE

SAMPLE JOBS:

- BEGIN JOB MYJOB ;

  RUN PROGX ;

  END JOB

- BEGIN JOB MYJOB ;

  PROCESS RUN PAYROLL ON PRODUCTION ;

  END JOB % Control gets to the end of job before Payroll is finished.

- BEGIN JOB MYJOB ;

  DESTNAME = SITE ;

  PROCESS RUN USER1/PROGA ;

  RUN USER1/PROGB ;

  END JOB

- BEGIN JOB MYJOB ;

  QUEUE = 30 ; % CLASS = 30 ;

  DESTNAME = CECRJE1 ;

  RUN (USER)CEC/INITIALS ON EDUCATION ;

  END JOB

SECTION 1: TASK INITIATION AND CONTROL

UNIT 3: Task Initiation

PRACTICE EXERCISE

Write the WFL statements that will have the system do the following. Assume that the files are on the Halt/Load unit DISK unless otherwise stated.

1. Initiate the object program CHECK/WRITER which is located on disk named USERB.

2. Initiate the WFL program BACKUP which is located on the Halt/Load unit.

3. Initiate the program CHECKBOOK/UPDATE simultaneously with the CHECK/WRITER program initiated in step 1.

4. Initiate the programs CHECKBOOK/TOTALS and CHECKBOOK/PRINTOUTS synchronously.

5. Initiate the programs A, B, and C with programs A and B asynchronous to each other and program C synchronous to B.

BLANK PAGE

TASK INITIATION AND CONTROL

Unit 4

Basic Task Control and Communications

OBJECTIVE

Construct basic task control and communication statements.

PURPOSE

After learning the skills to structure basic WFL and task initiation, the next step is to control those tasks which perform a required sequence of events.

RESOURCES

a. Student Guide	Section 1 Unit 4
b. A Series WFL Reference Manual	Section 6 Section 7

KEY WORDS

Task state
Task variable
Task identifier
IF
THEN
ELSE
DISPLAY
String expression
ABORT
COMPOUND
BEGIN - END
WAIT

SECTION 1: TASK INITIATION AND CONTROL

UNIT 4: Basic Task Control and Communications

LEARNING SEQUENCE

Up to this point, the basic structure of a WFL job has been presented along with simple task initiation statements. Consideration must be given to having some knowledge of what those tasks are doing. The system provides a means for checking on certain types of status such as whether the task is active, scheduled, completed, or completed ok. This type of information is referred to as the **task state**. To be able to check on the status or state of a task, there is a special type of declaration statement particularly for use with tasks. The use of this type of declaration statement provides a **task variable** which may then be assigned to a particular task. When it is assigned, the task variable becomes a **task identifier**. A task identifier may then be referenced, as to the task state, to make a decision concerning the flow of the job.

- TASK T1, T2 ; % Declaration of a task variable
- RUN PROGA [T1] ; % Assignment of a task variable to a task

To make a decision within a WFL job, the simplest statement to use is an **IF** statement. The IF statement evaluates a boolean expression to determine if the value is true or false. If the expression is true, the statement following the **THEN** will be executed. If the expression is false, then either the statement after the **ELSE** will be executed, or if there is no ELSE phrase, the next sequential statement in the job will be executed. The Boolean expression used below is checking on a task state while the statements that may be used are any of those in section 7 of the reference manual.

- IF T1 ISNT COMPLETEDOK THEN < statement > ;
- IF T1 IS COMPLETEDOK

  THEN < statement > ;
- IF T1 IS ACTIVE

  THEN < statement >

  ELSE < statement > ;

LEARNING SEQUENCE

Once the ability of checking on a task is available, the next step is to be able to communicate that status with the operator when necessary. The **DISPLAY** statement is used to communicate with the operator by putting a message on the ODT, in the log file, and in the job summary. The message must be in the form of a **string expression**. A string expression normally is treated as a group of characters.

- DISPLAY < string expression > ;
- DISPLAY "THIS TASK DID NOT COMPLETE" ;
- DISPLAY STR1 ; % Using a string variable

Another statement that might be useful for displaying information to the operator and also for discontinuing the job is the **ABORT** statement. The ABORT statement allows the WFL programmer to "abort" the job currently executing. It optionally allows the capability to display a string expression. The abort statement also allows one to abort an individual task through the use of its task identifier. An optional string expression may be displayed prior to aborting the task.

- ABORT ; % JOB
- ABORT < string expression > ;
- ABORT "THIS TASK FAILED, ABORTING THE JOB" ;
- ABORT STR1 ;
- ABORT [TSK1] ; % TASK
- ABORT [T1] "TASK T1 IS ABORTED" ;

One limitation of the IF statement is that the THEN and the ELSE phrases only allow one statement to be executed. To make up for that limitation, another statement has been provided to allow several statements to be treated as one statement. This statement is the **COMPOUND** statement. If the COMPOUND statement is used after the THEN or ELSE phrases, any number of statements may be executed. The COMPOUND statement starts with **BEGIN** and stops with an **END**.

SECTION 1: TASK INITIATION AND CONTROL

UNIT 4: Basic Task Control and Communications

LEARNING SEQUENCE

- IF T1 ISNT COMPLETEDOK THEN < statement > ;

- IF T1 IS COMPLETEDOK

```
  THEN

    BEGIN                          % USE OF A COMPOUND STATEMENT
      < statement > ;
      < statement > ;
      < statement > ;
      < statement > ;
      < statement > ;
    END ;
```

- IF T1 IS ACTIVE

```
  THEN

    BEGIN                          % USE OF A COMPOUND STATEMENT
      < statement > ;
      < statement > ;
      < statement > ;
      < statement > ;
    END

  ELSE

    BEGIN                          % USE OF A COMPOUND STATEMENT
      < statement > ;
      < statement > ;
      < statement > ;
    END ;
```

LEARNING SEQUENCE

There is still one problem that can exist in basic flow control in that there is not any method to temporarily suspend the current flow until an asynchronous task is finished. The statement provided for this purpose is the **WAIT** statement. This statement allows the programmer to wait on a task completion, time element, file existence, particular status of a task, or operator input. Examples of these follow:

```
BEGIN JOB MYJOB ;
  TASK T1 ;
  PROCESS RUN MYPROG1  [ T1 ] ;
  RUN MYPROG2 ;
  WAIT ( T1 ) ;                  % MYPROG2 COMPLETES & WAITS T1
  RUN MYPROG3 ;                  % MYPROG3 WAITS FOR MYPROG1
END JOB
```

```
BEGIN JOB MYJOB ;
  RUN MYPROG1 ;
  WAIT (30) ;                    % WAIT FOR 30 SECONDS
  RUN MYPROG2 ;                  % MYPROG2 EXECUTES AFTER WAIT
END JOB
```

```
BEGIN JOB MYJOB ;
  PROCESS RUN MYPROG1 ;
  RUN MYPROG2 ;
  WAIT (FILE MASTERFILE IS RESIDENT) ;
                                 % MYPROG2 COMPLETES & WAITS
                                 % ON MASTERFILE BEING RESIDENT
  RUN MYPROG3 ;                  % MYPROG3 EXECUTES WHEN
                                 % MASTERFILE IS RESIDENT
END JOB
```

SECTION 1: TASK INITIATION AND CONTROL

UNIT 4: Basic Task Control and Communications

LEARNING SEQUENCE

```
BEGIN JOB MYJOB ;
  PROCESS RUN MYPROG1 ;
  RUN MYPROG2 ;
  WAIT ;                          % WAIT FOR < mix number > HI
  RUN MYPROG3 ;                   % FROM THE OPERATOR
END JOB
```

```
BEGIN JOB MYJOB ;
  PROCESS RUN MYPROG1 ;
  RUN MYPROG2 ;
  WAIT (OK) ;                     % WAIT FOR < mix number > OK
  RUN MYPROG3 ;                   % FROM THE OPERATOR
END JOB
```

```
BEGIN JOB MYJOB ;
  PROCESS RUN MYPROG1 ;
  RUN MYPROG2 ;
  WAIT ("MOUNT TAPE", OK) ;       % DISPLAYS MOUNT TAPE &
  RUN MYPROG3 ;                   % WAITS FOR < mix number > OK
END JOB.                          % FROM THE OPERATOR
```

Using the basic job structure below, the task control and communication with the operator usually becomes the largest portion of a WFL job. This can be seen in the example on the next page.

```
BEGIN JOB MYJOB ;
  PROCESS RUN PROGX ;             % RUN TASK X ASYNCHRONOUSLY
  RUN PROGY ;                     % RUN TASK Y
  RUN PROGZ ;                     % RUN TASK Z AFTER Y
END JOB
```

LEARNING SEQUENCE

```
00000100 BEGIN JOB MYJOB ;
00000200   QUEUE = 30 ;
00000300   PRIORITY = 70 ;
00000400   DESTNAME = RJE1 ;
00000500   TASK T1, T2, T3, T4 ;
00000600   PROCESS RUN PROGX [ T1 ] ;
00000700   RUN PROGY [ T2 ] ;
00000800   WAIT ( T1 ) ;
00000900   IF T1 ISNT COMPLETEDOK
00001000     THEN
00001100       BEGIN
00001200         DISPLAY "PROGX FAILED, RETRY IN PROGRESS" ;
00001300         RUN PROGX/BKUP [ T3 ] ;
00001400         IF T3 ISNT COMPLETEDOK
00001500           THEN ABORT "PROGX/BKUP FAILED" ;
00001600       END;
00001700   IF T2 ISNT COMPLETEDOK
00001800       THEN ABORT "PROGY FAILED " ;
00001900 RUN PROGZ [ T4 ] ;
00002000 IF T4 ISNT COMPLETEDOK
00002100   THEN DISPLAY "PROGZ FAILED, SO DID THE JOB" ;
00002200 END JOB
```

PRACTICE EXERCISE

Write the WFL statements that will have the system do the following:

1. Write a WFL statement to execute a task synchronously using a task identifier TSK1. The object code file for the task is called OBJECT/PAYROLL/ONE.

2. Write a WFL statement to check on the status of task T1 to see if it completed ok. If T1 did complete ok, then execute the task PROG/UPDATE2; otherwise, display the error message "T1 FAILED".

3. Write a WFL statement to abort the job if task T2 fails.

4. Write the WFL statements to execute two tasks, asynchronously. The names of the two tasks are PROG1 and PROG2 respectively. Each task should have a task identifier and there should be a statement to suspend further execution of the job until the asynchronous task is finished.

TASK INITIATION AND CONTROL

Unit 5

Job Initiation

OBJECTIVE

Construct WFL job initiating statements.

PURPOSE

Once it is known how to write WFL jobs, it becomes necessary to be able to start them to verify that they do what is intended.

RESOURCES

a.	Student Guide	Section 2 Unit 5
b.	A Series WFL Reference Manual	Section 2 Section 6
c.	A Series ODT Reference Manual	Section 2
d.	The B 6700 WFL Primer	Chapt 14

KEY WORDS

START
STARTJOB

SECTION 2: TASK INITIATION AND CONTROL

UNIT 5: Job Initiation

LEARNING SEQUENCE

With enough information to write an effective WFL job, the next logical step is to initiate the job. There are several places from where that may be accomplished. CANDE and the ODT are probably the most common; however, jobs may also be started from a card reader, a zip from a program, or from other jobs.

- Using a CANDE terminal:

  - **START** < WFL file name > % Used to start a WFL file resident on disk

  - START % Used to start the current CANDE workfile

  or

  - ST < WFL file name > % Use of the abbreviated form of the start command

  - ST

- Using the ODT console

  - START < WFL file name > % Used to start a WFL file resident on disk

  - **STARTJOB** < WFL file name >

A point to note about starting WFL jobs is that the Controller may recognize some statements passed to it as being part of WFL even though they are not an ordinary ODT command. The Controller will insert a BEGIN JOB in front of the statements and an END JOB behind them which effectively creates a WFL job. The Controller then passes this job to the WFL compiler. Task initiation statements are good examples of this.

SECTION 2: TASK INITIATION AND CONTROL

UNIT 5: Job Initiation

PRACTICE EXERCISES

1. Write the command to initiate a CANDE workfile whose type is JOB.

2. Write the command to initiate a job to compile a program. The name of the job file is called COMP/FILE.

3. Explain the use of the START command when your workfile is the job file and your workfile has not been saved.

SECTION 1: TASK INITIATION AND CONTROL

LAB : 1-1

Write and run a Work Flow job which includes the steps described below. You may find it helpful to write, perform, and check each step separately before adding the next step to the job.

STEP 1: The job name is to be in the following format.

<your initials>/LAB1

STEP 2: The job is to execute at the default priority for the class usercode.

STEP 3: Program Print/Practice is to execute first.

STEP 4: If program Print/Practice is completed successfully then the program Cobol/Sample is to be executed asynchronously with Algol/Sample. Job flow control is to be maintained through Algol/Sample.

STEP 5: If either sample program fails, terminate the job immediately, specifying which program failed.

STEP 6: After the previous step is accomplished, start the job from CANDE at your work station. Get onto the MARC window and determine what is in the mix for this job(AA from MARC action line). Get back onto the CANDE window and enter the required information. Terminating the Algol/Sample program should terminate the Cobol/Sample normally and proceed to a normal EOJ.

SAVE YOUR JOB FOR FUTURE PROJECTS.

SHOW THE JOB SUMMARY INCLUDING STEPS 1 THRU 6 TO THE INSTRUCTOR.

SECTION 2

FILE MAINTENANCE AND COMPILATIONS

FILE MAINTENANCE AND COMPILATIONS

SECTION OBJECTIVE

Construct file maintenance and compilation statements.

PURPOSE

This section provides the information necessary to perform basic library maintenance and compilations using task attributes and file attributes to set the environment for each executing task.

UNIT OBJECTIVES

- Construct task equation statements using task attributes.
- Construct file equation statements using file attributes.
- Construct file maintenance statements using appropriate WFL naming conventions.
- Construct COMPILE statements using file equations.
- Construct WFL job initiating statements.

FILE MAINTENANCE AND COMPILATIONS

Unit 1

Task Equations

OBJECTIVE

Construct task equation statements using task attributes.

PURPOSE

In the operation of a large system, it is sometimes necessary to change the environment in which a task is executing. Under WFL, the capability exists to modify the environment of a task on an individual basis.

RESOURCES

a.	Student Guide	Section 2 Unit 1
b.	A Series WFL Reference Manual	Section 5 Appendix A
c.	A Series Print System Reference Manual	Section 3.2

KEY WORDS

Task attribute
DESTNAME
FAMILY
JOBNUMBER
STACKNO
NAME
MAXCARDS
MAXIOTIME
MAXLINES
MAXPROCTIME
MAXWAIT
PRINTDEFAULTS
PRIORITY
RESTART
RESTARTED
STATION
STATUS
SW1 THRU SW8
TASKVALUE
USERCODE
Task equation

SECTION 2: FILE MAINTENANCE AND COMPILATIONS

UNIT 1: Task Equations

LEARNING SEQUENCE

A **Task Attribute** defines a characteristic of a task. Changing the state of a single attribute changes the characteristics of a task and may cause changes in subsequent tasks if a task identifier is used.

- TASK ATTRIBUTES

- **DESTNAME**	Destination name for printer or punch backup. This attribute may be set to either a station name or SITE. It is used frequently for sending printed reports to remote sites under RJE.
- **FAMILY**	Indicates which family specifications are to be applied to a task. It basically controls the access to families. TAPE is not allowed as a familyname.
- **JOBNUMBER**	Represents the mix number of the job.
- **STACKNO**	Represents the mix number of the task.
- **NAME**	Represents the name of the task.
- **MAXCARDS** **MAXIOTIME** **MAXLINES** **MAXPROCTIME** **MAXWAIT**	Represents limits imposed on the task or job by either system defaults or programmer supplied values.
- **PRINTDEFAULTS**	Used to set file attribute default values for printed output.
- **PRIORITY**	Used by the system for scheduling functions.
- **RESTART**	Specifies the number of times the task may be restarted following an error termination.
- **RESTARTED**	True if the task has been restarted or after a rerun statement.

LEARNING SEQUENCE

- TASK ATTRIBUTES (continued)

  - **STATION** — Contains the logical station number.
  - **STATUS** — Represents the current task state. It is very similar to the task state discussed earlier. It may not be used in a task equation.
  - **SW1 THRU SW8** — Used in COBOL74.
  - **TASKVALUE** — Provides a means for passing one real number to a task. It is not dealt with as a parameter. Known in WFL as VALUE.
  - **USERCODE** — Specifies the usercode under which a task is to execute.

A **Task Equation** is used to specify changes to the current task attributes within the program being initiated. The attribute name must be used in a task equation.

- RUN PROGA/OBJ ;

  PRIORITY = 70 ;

  MAXLINES = 1000 ;

  DESTNAME = CECRJE1 ;

- PROCESS RUN PROGB/OBJ ;

  FAMILY DISK = PRODUCTION OTHERWISE DISK ;

  USERCODE = MYUSERCODE/MYPASSWORD ;

  VALUE = 3 ;

SECTION 2: FILE MAINTENANCE AND COMPILATIONS

UNIT 1: Task Equations

PRACTICE EXERCISE

Write the WFL statements that will have the system do the following:

1. Write the WFL statements to execute PROGA using a task identifier TSK1 synchronously with another task. PROGA is to run with a priority of 80.

2. Write the WFL statements to execute PROGB asynchronously with PROGC, and allow a maximum output of 3000 printed lines for PROGC.

3. Write the WFL statements to execute PROGD synchronously with another task and find its files only under the family USER2. Those files are located in the usercode PRODUCTION and a have a password of PROD1.

4. Write the WFL statements to execute PROGD synchronously with another task. PROGD needs a value of 3 to be passed to it for correct operation.

FILE MAINTENANCE AND COMPILATIONS

Unit 2

File Equations

OBJECTIVE

Construct file equation statements using file attributes.

PURPOSE

When a programmer writes a program, the filenames that are used in the program seldom match those used on a system. The capability exists to modify the physical file that a program is to access, but also other attributes or specifications about the file.

RESOURCES

a.	Student Guide	Section 2 Unit 2
b.	A Series WFL Reference Manual	Section 5
c.	I/O Subsystem Reference Manual	Section 4

KEY WORDS

File attribute
INTNAME
TITLE
FILENAME
FAMILYNAME
FAMILYINDEX
KIND
FILETYPE
FILEUSE
UNITS
MAXRECSIZE
MINRECSIZE
BLOCKSIZE
AREASIZE
AREAS
FLEXIBLE
CRUNCHED
MYUSE

SECTION 2: FILE MAINTENANCE AND COMPILATIONS

UNIT 2: File Equations

LEARNING SEQUENCE

File names and file titles are used to identify physical files. A file name identifies a physical file without consideration of the disk where it resides. No pack name is required if the file resides on the Halt/Load family or if security invokes a default pack. A File Title identifies a physical file and the disk on which it resides. Directory names and directory titles are used to identify a group of physical files all having the same first name or belonging to the same user.

- XYZ
- Y/Z ON USERA
- A/=
- (A)B ON DISK
- ABC/DEF
- (LMN)= ON PACKB

SECTION 2: FILE MAINTENANCE AND COMPILATIONS

UNIT 2: File Equations

LEARNING SEQUENCE

A **File Attribute** defines a characteristic of a file. Changing the state of a single attribute changes the characteristics of the file and may cause the states of other attributes to be changed.

- FILE ATTRIBUTES

- **INTNAME**	Internal filename chosen by the programmer of a task.
- **TITLE**	Complete external filename. Associates a logical file with a physical file or match port files. Default = INTNAME
- **FILENAME**	External filename. Used to indentify the physical file. Default = INTNAME
- **FAMILYNAME**	Name or label of a disk family.
- **FAMILYINDEX**	Relative number indicating a physical unit in the family.
- **KIND**	Describes the peripheral unit associated with the logical file.
- **FILETYPE**	Specifies the format of the records and the structure used to store them.
- **FILEUSE**	Describes how the file may be used.

LEARNING SEQUENCE

- FILE ATTRIBUTES (continued)

- **UNITS**	Indicates whether certain attribute values are in words or bytes.
- **MAXRECSIZE**	Specifies the maximum record size that may be used in the logical file.
- **MINRECSIZE**	Specifies the minimum record size that may be used in the logical file.
- **BLOCKSIZE**	Usually the amount of physical transfer that takes place in a physical I/O. Size is usually a multiple of MAXRECSIZE.
- **AREASIZE**	Specifies the number of logical records in a disk area called a row.
- **AREAS**	Specifies the total number of rows that may be allocated for a file.
- **FLEXIBLE**	Indicates whether a file may be allocated more areas.
- **CRUNCHED**	Indicates whether a file was closed with a crunch. If it was, the area within the row beyond the last used block is returned to the system.
- **MYUSE**	Describes the user's intended use of the file, such as: INPUT, OUTPUT, or IO.

SECTION 2: FILE MAINTENANCE AND COMPILATIONS

UNIT 2: File Equations

LEARNING SEQUENCE

A **file equation** is used to specify changes to the various file attributes declared within the program being initiated. The internal name of the file must be used with a file equation.

- RUN PROGB/OBJ ;
  - FILE INFILE(BLOCKSIZE = 30, KIND = TAPE) ;
  - FILE LINE(KIND = PRINTER) ;

- RUN PROGC/MYVERSION ;
  - FILE INP(KIND = DISK, TITLE = DATAFILE/MYVERSION) ;
  - FILE OUT(KIND = TAPE) ;

- COMPILE PROGC/MYVERSION COBOL74 LIBRARY ;
  - COMPILER FILE CARD(KIND = DISK, TITLE = PROGC/SRC) ;

- COMPILE PROGB/OBJ ALGOL LIBRARY ;
  - ALGOL FILE CARD(KIND = DISK, TITLE = PROGB/SRC) ;
  - FILE CARD(KIND = TAPE) ;

PRACTICE EXERCISE

Write the WFL statements that will have the system do the following:

1. Write a task initiating statement for the program PAYROLL that file equates the internal file INP to the external file named UPDATE/DATA which is located on the family PRODUCTION.

2. Write a file equation that equates the internal file OUTPUTT to the external file whose record size is 180 bytes and whose block size is 3600 bytes. The Units attribute should be set to words, the device to be used is tape, and the filename is NEWMASTER.

3. Write a file equation for the ALGOL compiler whose input card image file is located on the family USERPACK and whose name is SOURCEIN.

4. Write a file equation for the task PRODUCTIONII whose input is normally on tape but for this run it will be found on disk. The family name is PRODUCTION, the filename is DATAIN, record size is 180 bytes, block size is 300 words, and it will be used for input only. The internal file name being used for PRODUCTIONII is the same as the external name.

SECTION 2: FILE MAINTENANCE AND COMPILATIONS

UNIT 2: File Equations

PRACTICE EXERCISE

A. Write a WFL job using the instructions below.

1. Name the job < your initials > .
2. Put the job in queue 20.
3. The user-coded files are on the family SYSTEMS, and the utilities and compilers are on the family DISK.
4. The job should execute at priority 70.
5. Run the programs PROG/2A and PROG/2B synchronously.
6. End the job.

B. Add the following to the job written above.

1. If PROG/2A does not end normally, abort the job. (Do not allow PROG/2B to run)
2. Display a message to the operator indicating whether PROG/2B ran successfully or not.

C. Add the following to the job written above.

1. PROG/2A should run at priority 75.
2. Pass a value of 3 to PROG/2B.
3. PROG/2B expects a tape file with the internal name INPUTT. For this run, it should use the disk file called DATA/IN on SYSTEMS instead of the tape file.

FILE MAINTENANCE AND COMPILATIONS

Unit 3

Library Maintenance

OBJECTIVE

Construct file maintenance statements using appropriate WFL naming conventions.

PURPOSE

To be able to accomplish anything with a computer system, it is necessary to be able to have the appropriate files present. The user has available routines which will allow one to copy, add, and remove files, change filenames and security on selected files.

RESOURCES

a. Student Guide	Section 2 Unit 3
b. A Series WFL Reference Manual	Section 6

KEY WORDS

COPY
AUTORM
ADD
CHANGE
REMOVE
SECURITY
SECURITYTYPE
SECURITYUSE

SECTION 2: FILE MAINTENANCE AND COMPILATIONS

UNIT 3: Library Maintenance

LEARNING SEQUENCE

File maintenance statements may be used to change the name, attributes, security, or residence of disk files. WFL uses the MCP library maintenance routines to perform this function.

The **COPY** statement is used to copy files between disk and tape media. If duplicate files exist, the results are dependent on how system options are set. If **AUTORM** is set, the existing version will be removed and the new version loaded in its place. If it is reset, a duplicate file condition will arise and display an appropriate message to the operator. If the **ADD** statement is used instead of COPY, the files will be copied to the destination where they are not already resident.

- COPY A FROM X(KIND=TAPE) TO Z(KIND=DISK) ;
- COPY A AS B FROM USERA ;
- COPY A/=, B/= TO BKUP(KIND=DISK) ;
- COPY A/= FROM XYZ(KIND=TAPE) TO USER(DISK), TO BKUP(DISK) ;
- COPY AND COMPARE = FROM DISK TO XYZ(KIND=TAPE) ;
- ADD COBOL/= FROM SYSTEM(KIND=TAPE) ;

SECTION 2: FILE MAINTENANCE AND COMPILATIONS

UNIT 3: Library Maintenance

LEARNING SEQUENCE

The **CHANGE** statement changes the names of files on a disk family. In WFL, if a family substitution statement is in effect, it will change the file on the substitute family only as of the 3.6 release.

CAUTION Use of the "from" clause may save you grief !

- CHANGE W/X TO Y/Z ;
- CHANGE A ON USERC TO B ;
- CHANGE R/= TO S/= ;
- CHANGE L/M TO N/O FROM USERA ;
- CHANGE A/= TO B/=, C/C TO D/D FROM USER1 ;

LEARNING SEQUENCE

The **REMOVE** statement removes files from a disk family. In WFL, if a family substitution statement is in effect, it will remove the file from the substitute family only as of the 3.6 release.

CAUTION Use of the "from" clause may save you grief !

- REMOVE A/B ;
- REMOVE C FROM USERC ;
- REMOVE D/= FROM USERD ;
- REMOVE V/W, X FROM PACKA, Y/= FROM PACKB, Z ;

The **SECURITY** statement is used to change the **SECURITYTYPE** and **SECURITYUSE** of files on disk.

- SECURITY A/B PRIVATE IO ;
- SECURITY C/D ON USER1 PUBLIC IN ;
- SECURITY E/F ON MYPACK GUARDED XYZ ;

SECTION 2: FILE MAINTENANCE AND COMPILATIONS

UNIT 3: Library Maintenance

PRACTICE EXERCISE

Using the files A through F below, generate the LIBRARY MAINTENANCE statements to accomplish the following in a WFL job.

Assume the following:

1. Unless otherwise noted, all these files should be manipulated under your usercode.

2. Unless otherwise noted, all these files are on the family DISK.

3. A = SYSTEM/ALGOL
   B = PROGONE/CLASS
   C = PROGTWO/CLASS
   D = PROGTWO/CLASSMATES
   E = SYSTEM/COBOL
   F = SYSTEM/FORTRAN

1. Copy file C from DISK to a tape labeled CDETAPE.

2. Copy file C from the tape CDETAPE to a tape labeled DATAPE.

3. Copy files A, B, C from DISK to a tape labeled MYTAPE.

4. Copy file B to exist under your usercode. Then using the CHANGE WFL command, change the name to < your initials >/PROGONE/CLASS.

5. Change the security of the file < your initials >/PROGONE/CLASS to PRIVATE, I/O.

6. Remove the files created in steps 4 and 5.

PRACTICE EXERCISE (continued)

Write the WFL statements that will have the system do the following. Assume that the Halt/Load Family is DISK.

1. Copy all the files with a first name of PAYROLL from a tape named PAYBKUP to the Halt/Load family.

2. Copy the files PROG1/SRC and PROG1/OBJ from the system disk to a user disk named BKUPPACK. Have each file compared immediately after it is copied.

3. Copy only those files that are not currently resident on a user disk named SYSBKUP to that disk from a tape named SYSTEM. The group of files is named DOCUMENT.

4. Change the first node of a group of filenames from ABC to XYZ. These files are located on a pack named USER1.

5. Remove the file RS/TU and XYZ from the pack named USER2.

6. Change the SECURITYTYPE of the file A on the system pack to PUBLIC and its SECURITYUSE to INPUT only.

FILE MAINTENANCE AND COMPILATIONS

Unit 4

Compiles

OBJECTIVE

Construct COMPILE statements using file equations.

PURPOSE

Since compilers are used at most sites, it is beneficial to know how to compile through a WFL job. In this way, the operations staff has more control over system resources.

RESOURCES

a. Student Guide	Section 2 Unit 4
b. A Series WFL Reference Manual	Section 6

KEY WORDS

$ (Dollar Sign Option)
LINE
LIST
ERRORFILE
ERRLIST
ERRORLIST
TAPE
SOURCE
MERGE
NEWTAPE
NEW
NEWSOURCE
SET
RESET
POP
COMPILEDOK
DATA CARD
? (invalid punch)

SECTION 2: FILE MAINTENANCE AND COMPILATIONS

UNIT 4: Compiles

LEARNING SEQUENCE

As covered in Unit 1, the WFL compiler takes source input and generates a specialized object code file. The other compilers on the system work in a similar manner in that they also take source input and generate object code output. These other compilers offer more options than the WFL compiler.

The basic compile statement was covered in Section 1 Unit 3 and the use of file equations in Unit 2 of this section. To be able to make use of a file equation it is necessary to know the internal files that a task has. There are several options available to the compiler through the use **$ (dollar sign) options** in those compilers. The normal or default diagram of a compilation is shown below.

SOURCE **COMPILER** **OBJECT**

Every compile in WFL must have two statements:

- Compiler initiate
- Compiler file equation or data card file in the WFL deck

PRIMARY SOURCE

COMPILER **OBJECT**

SECONDARY SOURCE

Compilers may have two input files which requires the use of an additional compiler file equation and a dollar sign option.

PRIMARY SOURCE (CHANGES)	SECONDARY SOURCE (ORIGINAL)	THE COMPILER USES (END RESULT)
010**$SET MERGE**		010**$SET MERGE**
100 BEGIN JOB ONE;	100 BEGIN JOB;	100 **BEGIN JOB ONE;**
250 PRIORITY = 80;	200 QUEUE = 50;	200 QUEUE = 50;
		250 **PRIORITY = 80;**
	300 DISPLAY "ONE";	300 DISPLAY "ONE";
	400 END JOB	400 END JOB

SECTION 2: FILE MAINTENANCE AND COMPILATIONS

UNIT 4: Compiles

LEARNING SEQUENCE

The compile statement format is

COMPILE < object code file name > < compiler > < disposition > ;

- The object code file name should be different from the source file name.
- The compilers are:

ALGOL			
COBOL	BASIC	DCALGOL	NDL
COBOL74	XALGOL	ESPOL	NDLII
FORTRAN	PL/1	BINDER	NEWP
FORTRAN77	PASCAL	RPG	SORT

- The disposition is one of the following:

GO LIBRARY SYNTAX LIBRARY GO < blank >

INTERNAL FILENAME	$ OPTION	FUNCTION
CODE	--------	OBJECT CODE FILE (DISK)
CARD	--------	PRIMARY SOURCE FILE (CARD)
LINE	**LIST**	SOURCE LISTING (PRINTER)
ERRORFILE	**ERRLIST** **ERRORLIST** (COBOL74)	ERROR LISTING (PRINTER)
TAPE **SOURCE** (COBOL74)	**MERGE**	SECONDARY SOURCE (DISK)
NEWTAPE **NEWSOURCE** (COBOL74)	**NEW**	NEW SOURCE FILE (DISK)

SECTION 2: FILE MAINTENANCE AND COMPILATIONS

UNIT 4: Compiles

LEARNING SEQUENCE

The compiler options mentioned are made available through the use of a SET, RESET, or POP function. These functions with the options usually occur as the first records in the primary input file, although other compiler options may occur at any place in the file. The $ must occur in certain columns.

- The functions operate as follows:

SET	Turns the option on.
RESET	Turns the option off.
POP	Returns the option to the previous setting.

- The $ must be placed in certain columns as follows to have that $ record shown or not shown in the listing:

ALGOL	$ in column 1 (not shown) $ in column 2 (shown)
COBOL74	$ in column 7 (not shown) $ in column 8 (shown)

- A sample compile statement is as follows:

```
COMPILE OBJECT/ALGOL/ONE ALGOL LIBRARY;
  ALGOL FILE CARD(KIND = DISK, TITLE = SOURCE/ALGOL/PATCH1);
  ALGOL FILE TAPE(KIND = DISK, TITLE = SOURCE/ALGOL/ONE);
  ALGOL FILE NEWTAPE(TITLE = NEWSOURCE/ALGOL/ONE);
```

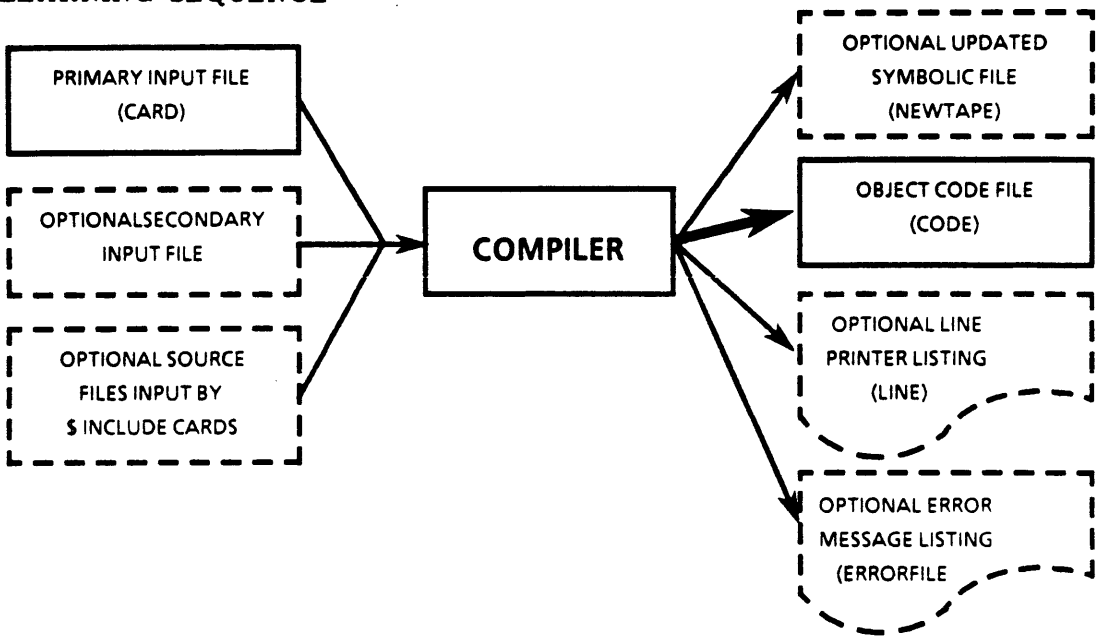
- A primary input file may contain as its first record when it is a patchfile:

```
00000100$ SET MERGE NEW
```

SECTION 2: FILE MAINTENANCE AND COMPILATIONS

UNIT 4: Compiles

LEARNING SEQUENCE



COMPILER DATA FLOW

To check on the status of a compile, one of the task states available is **COMPILEDOK** which is set if there were no errors during compilation. A task identifier may be used to check on this status. There are two task identifier assignments possible in a compile statement. The first will be assigned to the object code being generated and the second will be assigned to the compiler itself.

If this is used in the previous example, T1 is assigned to the status of the program being compiled and T2 is assigned to the compile status itself.

```
COMPILE OBJECT/ALGOL/ONE [ T1 ] ALGOL [ T2 ] LIBRARY GO;
  ALGOL FILE CARD(KIND = DISK, TITLE = SOURCE/ALGOL/PATCH1);
  ALGOL FILE TAPE(KIND = DISK, TITLE = SOURCE/ALGOL/ONE);
  ALGOL FILE NEWTAPE(TITLE = NEWSOURCE/ALGOL/ONE);
```

SECTION 2: FILE MAINTENANCE AND COMPILATIONS

UNIT 4: Compiles

LEARNING SEQUENCE

By adding the data deck to the previous example, the file equation will be replaced by the data deck. When using the data deck, all data following the ALGOL **DATA CARD** is assumed to be data. The only way that the WFL compiler will stop regarding input as data is to find a **? (invalid punch)** in the first column of a record.

```
COMPILE OBJECT/ALGOL/ONE [ T1 ] ALGOL [ T2 ] LIBRARY GO ;
  ALGOL FILE TAPE(KIND = DISK, TITLE = SOURCE/ALGOL/ONE) ;
  ALGOL FILE NEWTAPE(TITLE = NEWSOURCE/ALGOL/ONE) ;
  ALGOL DATA CARD
$ SET MERGE NEW                                          00000010
END.                                                     00001500
? RUN MYPROG ;
  RUN OBJECT/ALGOL/ONE ;


COMPILE OBJECT/COBOL/ONE [ T1 ] COBOL [ T2 ] LIBRARY GO ;
  COBOL FILE TAPE(KIND = DISK, TITLE = SOURCE/COBOL/ONE) ;
  COBOL FILE NEWTAPE(TITLE = NEWSOURCE/COBOL/ONE) ;
  COBOL DATA CARD
000010$  SET MERGE NEW
013200      77  A                PIC 9V99 VALUE 99.

?
 RUN MYPROG ;
 RUN OBJECT/COBOL/ONE ;
```

SECTION 2: FILE MAINTENANCE AND COMPILATIONS

UNIT 4: Compiles

PRACTICE EXERCISE

Write the WFL statements that will have the system do the following. In all cases, the code generated should be saved on disk.

1. Compile a program naming it PROGA/OBJ. The RPG source is named PROGA/SRC on the same disk.

2. Compile a COBOL program whose object is OBJECT/PRODUCTIVITY/ONE and whose source is SRC/PRODUCTIVITY/ONE. A patchfile is used to supply corrections to the source file and its name is SRC/PRODUCTIVITY/PATCHONE.

3. Compile the ALGOL program OBJ/ACCTS/PAY with the source SRC/ACCTS/PAY and the patchfile PATCHFILE/ACCTS/PAY. Create a new source file called NEWSRC/ACCTS/PAY, put any errors in a disk file called NEWSRC/ACCTS/PAY/ERRS, and send the listing to a normal printer backup file.

4. Compile the COBOL source SRC/INVENTORY naming the task OBJECT/INVENTORY. Save the object file on disk after compilation and immediately execute it. The task should execute at a priority of 65 and have a maximum processor time of 300 seconds which are to be compiled into the task.

SECTION 2: FILE MAINTENANCE AND COMPILATIONS

LAB : 2-1

Write and run a Work Flow job which includes the steps described below. You may find it helpful to write, perform, and check each step separately before adding the next step to the job.

STEP 1: Copy the following files from the family _____ to the family _____: WFL/PROG1SRC, WFL/PROG3SRC, WFL/PROG5SRC, and WFL/PROG8SRC.

STEP 2: Compile PROG8 using PROG8SRC with the COBOL74 compiler making sure to keep a copy on disk.

STEP 3: Run PROG8. PROG8 has an output disk file internally named DISK-FILE. This newly created file should have a directory name of your initials, a name of MASTER and reside on the user disk. Assure that more areas can be allocated than may be specified within the program.

STEP 4: Make a backup copy of MASTER named MASTERBKUP under your directory.

STEP 5: Change the name of the file MASTERBKUP TO MSTRBKUP.

STEP 6: Compile PROG3 using PROG3SRC with the COBOL74 compiler making sure to keep a copy on disk.

STEP 7: Run PROG3. The input file was created in STEP 3. The output file, internally named PUNCH-FILE, is a card file. For this run only, create a disk file with a directory name of your initials and a name of DISKPUNCH.

STEP 8: Compile PROG5 using PROG5SRC with the COBOL compiler making sure to keep a copy on disk.

STEP 9: Run PROG5. The input file, internally named TIME-CARD, is a card file. Use the disk file WFL/TIMECARD instead.

STEP 10: Compile PROG1 using PROG1SRC and COBOL74 making sure to keep a copy on disk.

STEP 11: Run PROG1. The internal name of the print file is PRINTFILE.

STEP 12: Remove all files you created in this WFL job from the disk.

SECTION 2: FILE MAINTENANCE AND COMPILATIONS

LAB : 2-2

Write and run a Work Flow job which includes the steps described below. You may find it helpful to write, perform, and check each step separately before adding the next step to the job.

STEP 1: The source files CLASS/PROGANEW and CLASS/PROGBNEW exist on disk. Copy these files simultaneously (not one after the other) into your own library, under the names UTIL/< yourname >/PROGANEW and UTIL/< yourname >/PROGBNEW.

STEP 2: Compile the COBOL source UTIL/< yourname> /PROGANEW. Name the object file UTIL/< yourname >/PROGA, and store it in your library for execution later.

STEP 3: If PROGANEW fails to compile (step 2 above), then do the following:

A. Place the COBOL source CLASS/PROGAOLD into your library under the name UTIL/< yourname >/PROGAOLD.
B. Compile this source, and store the object as UTIL/< yourname >/PROGA.

STEP 4: If PROGAOLD has syntax errors (step 3 above), stop the entire job and give the reason for termination in the job summary.

STEP 5: Patch and compile the ALGOL program described below.

A. Name the object UTIL/< yourname >/PROGB.
B. The patchfile is CLASS/PROGBNEW/PATCH ON DISK.
C. The source is UTIL/< yourname >/PROGBNEW ON DISK.
D. Create a new source disk file called UTIL/< yourname >/NEWBB.

STEP 6: If PROGBNEW fails to compile (step 5 above), then copy the object file CLASS/PROGB into your library under the name UTIL/< yourname >/PROGB.

STEP 7: If UTIL/< yourname >/PROGB does not reside on disk at this point (steps 5 and 6 above), stop the entire job and give the reason for termination in the job summary.

SAVE YOUR JOB FOR USE IN FUTURE LAB PROJECTS.

SHOW THE JOB SUMMARY INCLUDING STEPS 1 THRU 7 TO THE INSTRUCTOR.

SECTION 2: FILE MAINTENANCE AND COMPILATIONS

LAB : 2-3

Additional labs may be supplied by the instructor.

SECTION 3

EXPRESSIONS, ITERATIONS, and FUNCTIONS

EXPRESSIONS, ITERATIONS, and FUNCTIONS

SECTION OBJECTIVE

Construct a WFL job using assignment statements, iterative statements, and WFL functions.

PURPOSE

This section adds the concept of using variables in task control. Consideration will also be given to routines provided for the WFL programmer in the form of functions.

UNIT OBJECTIVES

- Construct assignment statements using appropriate expressions.
- Construct iterative and case statements to control the flow of a WFL job.
- Construct statements using string functions.
- Construct statements using numeric functions.
- Construct job attribute statements.
- Construct job initiating statements that pass parameters.

EXPRESSIONS, ITERATIONS, and FUNCTIONS

Unit 1

Declarations, Expressions, and Assignments

OBJECTIVE

Construct assignment statements using appropriate expressions.

PURPOSE

One of the most useful tools in any language, is the ability to calculate values. This unit deals with the declaration of variables, the evaluation of expressions, and the assignment of values.

RESOURCES

a. Student Materials	Section 3 Unit 1
b. A Series WFL Reference Manual	Section 6 Section 7

KEY WORDS

REAL
INTEGER
BOOLEAN
STRING
expressions
primary
real expressions
real variables
integer expressions
integer variables
boolean expression
boolean variables
string expressions
string variables
 # string primary
assignment statement
MYSELF
MYJOB

SECTION 3: EXPRESSIONS, ITERATIONS, and FUNCTIONS

UNIT 1: Declarations, Expressions, and Assignments

LEARNING SEQUENCE

In Section 1, the declaration section was mentioned as one of the five major sections of a job. In making a declaration, the compiler is directed to generate the code necessary to reserve a portion of memory for the variable being declared. The amount of memory is determined by the type of declaration being made. As covered earlier, task variables are one type of declaration. In WFL, all variables must be explicitly declared. The new types to be covered in this unit are REAL, INTEGER, BOOLEAN, and STRING.

- **REAL**
  - NUMERIC
  - 12 DIGITS AND DECIMAL POINT MAXIMUM
  - WHOLE AND FRACTIONAL NUMBERS
- **INTEGER**
  - NUMERIC
  - 12 DIGITS AND NO DECIMAL POINT
  - WHOLE NUMBERS ONLY
- **BOOLEAN**
  - LOGICAL
  - TRUE or FALSE
- **STRING**
  - GROUP OF CHARACTERS
  - 256 CHARACTERS MAXIMUM

SECTION 3: EXPRESSIONS, ITERATIONS, and FUNCTIONS

UNIT 1: Declarations, Expressions, and Assignments

LEARNING SEQUENCE

With areas of memory reserved by declaration statements, the purpose is to store a value in memory. It may be necessary to manipulate a combination of many values to obtain a single value for storage. This evaluation is performed on an expression. **Expressions** may be one value (**primary**) or a combination of values manipulated through the use of operators. As with declarations, there are different types of expressions and operators.

ARITHMETIC OPERATORS:

- REAL % WHOLE or FRACTIONAL NUMBERS
  - \+ % ADDITION
  - \- % SUBTRACTION
  - * % MULTIPLICATION
  - / % REAL DIVISION
  - DIV % INTEGER DIVISION
  - MOD % MODULUS DIVISION
- INTEGER % WHOLE NUMBERS ONLY
  - \+
  - \-
  - *
  - DIV
  - MOD

ARITHMETIC OPERATOR PRECEDENCE:

- PREFIX + or - % Unary operator - one operand
- *, /, DIV, or MOD % Equal precedence
- INFIX + or - % In an expression between operands

SECTION 3: EXPRESSIONS, ITERATIONS, and FUNCTIONS

UNIT 1: Declarations, Expressions, and Assignments

LEARNING SEQUENCE

ARITHMETIC EXPRESSIONS:

- **Real expressions** yield numerical values by combining real primaries with arithmetic operators. The conventional operators, +, -, *, / for addition, subtraction, multiplication, and real division are used. The DIV operator produces a quotient with a truncated fractional part. The MOD operator returns the remainder of a divide operation.

- **Real variables** may be used to store the value of real expressions.

  - EXAMPLES:

```
* REAL R1, R2, R3 ;          % REAL DECLARATIONS
* REAL R4 ;

* 95 + 72                    % REAL EXPRESSIONS
* 57 - R2
* R3 * 4 / 3
* R1 DIV 3
```

- **Integer expressions** yield numerical values by combining integer primaries with arithmetic operators. The conventional operators, +, -, * for addition, subtraction, and multiplication are used. The DIV operator produces a quotient with a truncated fractional part. The MOD operator returns the remainder of a divide operation.

- **Integer variables** may be used to store the value of integer expressions.

  - EXAMPLES:

```
* INTEGER I1, I2, I3 ;       % INTEGER DECLARATIONS
* INTEGER R4 ;

* 95 + 72                    % INTEGER EXPRESSIONS
* 57 - I2
* 3 + 17 - 4 * 3
* I1 DIV 3
```

SECTION 3: EXPRESSIONS, ITERATIONS, and FUNCTIONS

UNIT 1: Declarations, Expressions, and Assignments

LEARNING SEQUENCE

BOOLEAN OPERATORS: % BOOLEAN OPERATOR PRECEDENCE

- NOT % NEGATE PRESENT VALUE
- AND % TRUE IF BOTH ARE TRUE
- OR % TRUE IF EITHER IS TRUE
- IMP % FALSE IF 1ST TRUE 2ND FALSE
- EQV % TRUE IF BOTH ARE THE SAME

BOOLEAN RELATIONAL OPERATORS:

- ARITHMETIC COMPARISON
  - LSS
  - GEQ
  - GTR
  - LEQ
  - EQL
  - NEQ
  - <
  - >
  - =
- STRING COMPARISON
  - =
  - EQL
  - NEQ
- TASK STATE COMPARISON
  - IS or ISNT
    - INUSE
    - COMPLETED
    - SCHEDULED
    - ACTIVE
    - STOPPED
    - ABORTED
    - COMPLETEDOK
    - COMPILEDOK

SECTION 3: EXPRESSIONS, ITERATIONS, and FUNCTIONS

UNIT 1: Declarations, Expressions, and Assignments

LEARNING SEQUENCE

BOOLEAN RELATIONAL OPERATORS: (continued)

- TASK MNEMONIC COMPARISON
  - IS
  - ISNT
- FILE RESIDENCY COMPARISON
  - IS or ISNT
    - RESIDENT

BOOLEAN EXPRESSIONS:

- A **boolean expression** is an expression that always evaluates to one of only two values: TRUE or FALSE. They provide the user with the power to make decisions during the run of a job. Boolean expressions usually contain two primaries and one operator, but can be more complex.
- **Boolean variables** may be used to store the value of boolean expressions. They may also be used in place of the boolean expressions as used later in this unit.
  - EXAMPLES:

```
* BOOLEAN B1, B2;                    % BOOLEAN DECLARATIONS
* BOOLEAN BOOL;

* I < 10                             % BOOLEAN EXPRESSIONS
* STR1 EQL "ONE"
* JOBSUMMARY IS SUPPRESSED
* T1 IS COMPLETEDOK
* FILE ABC IS RESIDENT AND T2 IS COMPILEDOK
```

SECTION 3: EXPRESSIONS, ITERATIONS, and FUNCTIONS

UNIT 1: Declarations, Expressions, and Assignments

LEARNING SEQUENCE

STRING OPERATORS:

```
% CONCATENATION OF STRINGS

• &      % STRING STRING
• *      % * STRING
• /      % STRING / STRING
• ON     % STRING ON STRING
• /=     % STRING /=
```

STRING EXPRESSIONS:

- **String expressions** yield a group of characters. Two strings may be concatenated together by the use of the "&" operator.
- **String variables** may be used to store the value of string expressions.

  - EXAMPLES:

```
* STRING S1, S2 ;                % STRING DECLARATIONS
* STRING STR ;

* "HELLO THERE EVERYONE"         % STRING EXPRESSIONS
* STR1 & STR7
* S1 / S2 ON "SYSTEMS"
* RUN # S1
* START # S1 / S2
```

A **# <string primary>** syntax may be used in dynamically built constructs such as filenames, family names, and usercodes using the string concatenation operators provided. Use of this format tells the system to use the contents of the variable rather than the variable name itself.

SECTION 3: EXPRESSIONS, ITERATIONS, and FUNCTIONS

UNIT 1: Declarations, Expressions, and Assignments

LEARNING SEQUENCE

An **assignment statement** assigns values to declared variables. The evaluation of an expression reduces that expression to a single "value" which will be assigned to a location in memory set aside by the declaration of a variable. In simple terms, it means that a value is stored at a memory address for later access. Variables may be assigned an initial value in the declaration statement. Since there are many different kinds of declarations, there may also be many types of assignments. For real, integer, boolean, and string type variables, the assignment operator is := (colon equals).

- REAL

```
- REAL R2 := 3.63, R4 := 18.477 ;        % INITIAL VALUE
- R1 := 3 + 7.62 ;
- R3 := R1 + 3.72 ;
- R6 := R1 + R6 ;
```

- INTEGER

```
- INTEGER I2 := 37 ;                     % INITIAL VALUE
- I1 := 5 + 2 ;
- I1 := I1 + 1 ;
- I1 := I1 + I2 ;
```

- BOOLEAN

```
- BOOLEAN B4 := FALSE ;                  % INITIAL VALUE
- B1 := TRUE ;
- B2 := I1 = 3 ;
- B3 := NOT B1 AND TRUE ;
```

- STRING

```
- STRING S1 := "EXECUTING " ;            % INITIAL VALUE
- STR1 := "THIS PROGRAM IS" ;
- STR2 := "NOT " ;
- STR3 := STR1 & S1 & STR2 ;             % THE ORDER OF
                                           CONCATENATION
- STR3 := STR1 & STR2 & S1 ;               DETERMINES THE
                                           RESULT
```

SECTION 3: EXPRESSIONS, ITERATIONS, and FUNCTIONS

UNIT 1: Declarations, Expressions, and Assignments

LEARNING SEQUENCE

Other variables may have different forms for their assignment statements. In the previous section, task variables were assigned to a task to become a task identifier. Through the task equation, attributes were assigned to a particular task. Attributes may also be assigned to the task variable, which when assigned to a task, assigns those attribute values to that task. There are three ways to assign task attributes.

- TASK EQUATION
  - RUN PROGA ;
    PRIORITY = 65 ;
  - RUN MYPROG ;
    MAXLINES = 1000 ;
- TASK VARIABLE
  - T1 (PRIORITY = 75) ;
  - T4 (PRIORITY = 65, MAXLINES = 100) ;
    - T1 (MAXPROCTIME=20); % USE IN A JOB

      RUN PROGA [T1] ;
    - INITIALIZE (T1) ; % T1 (STATUS = NEVERUSED)
      % RESETS STATUS OF T1

      RUN PROGB [T1] ;
- TASK VARIABLE AT DECLARATION
  - TASK T1, T2 (PRIORITY = 90), T3 ;
  - TASK T1, T2, T3(MAXLINES = 500, MAXPROCTIME = 30) ;

MYSELF and **MYJOB** are predeclared task variables. They are used in exactly the same way as any task variable except that they may not be assigned to a task. MYJOB and MYSELF are task variables which provide access to the values of the job's task attributes.

SECTION 3: EXPRESSIONS, ITERATIONS, and FUNCTIONS

UNIT 1: Declarations, Expressions, and Assignments

PRACTICE EXERCISE

Evaluate the following expressions in the order of their execution within the job.

```
BEGIN JOB EXPRESSIONS;
  STRING S,T,U;
  INTEGER I,J,R,V,W,X;
  BOOLEAN B,C;
  S := "X" ; ..... % S CONTAINS _____
  T := "AB" & "C" ; ..... % T CONTAINS _____
  S := S & "#T" ;..... % S CONTAINS _____
  U := T & S ; ..... % U CONTAINS _____
  S := "1" ; ..... % S CONTAINS _____
  S := S & S ;..... % S CONTAINS _____
  S := S & S ;..... % S CONTAINS _____
  T := S & U ; ..... % T CONTAINS _____
  I := 15 ; ..... % I CONTAINS _____
  J := 105 ; ..... % J CONTAINS _____
  R := 2 ;..... % R CONTAINS _____
  V := I * J ; ..... % V CONTAINS _____
  W := V DIV R ;..... % W CONTAINS _____
  W := V MOD R ; ..... % W CONTAINS _____
  B := S = T ;..... % B CONTAINS _____
  C := R < I ; ..... % C CONTAINS _____
END JOB
```

EXPRESSIONS, ITERATIONS, and FUNCTIONS

Unit 2

Flow Control Statements

OBJECTIVE

Construct iterative and case statements to control the flow of a WFL job.

PURPOSE

With the ability to calculate values, one way to shorten the amount of coding necessary is to create a looping structure or decide the direction of flow based on a value.

RESOURCES

a. Student Materials	Section 3 Unit 2
b. A Series WFL Reference Manual	Section 2 Section 6

KEY WORDS

GO TO
label id
WHILE DO
DO UNTIL
CASE

SECTION 3: EXPRESSIONS, ITERATIONS, and FUNCTIONS

UNIT 2: Flow Control Statements

LEARNING SEQUENCE

The GO TO statement is perhaps the easiest of the flow control statements to use; however, care must be taken to be sure what will actually take place when the job executes. The **GO TO** statement requires the use of a **label id** to determine where control is to be passed and will unconditionally transfer control to that label. **(BE SURE TO READ THE CAUTION IN THE REFERENCE MANUAL.)**

-
```
BEGIN JOB MYJOB ;                    % INFINITE LOOP
  INTEGER I ;
 DOIT:                               % LABEL ID
  RUN MYPROG ;
  GO TO DOIT ;                       % TRANSFERS CONTROL TO DOIT
END JOB ;
```

-
```
BEGIN JOB MYJOB ;                    % INFINITE LOOP?
  INTEGER I, J ;
  I := 1 ;
  J := 1 ;
 DOIT :                              % LABEL
  RUN MYPROG ;
  J := J + 1 ;
  I := I + J + 1 ;
  IF I = 5                           % EXECUTES MYPROG _____ TIMES
    THEN GO TO FINISH ;
  GO TO DOIT ;
 FINISH:                             % LABEL
END JOB ;
```

-
```
BEGIN JOB MYJOB ;
  INTEGER I ;
 DOITAGAIN :                         % LABEL
  RUN MYPROG ;
  I := I + 1 ;
  IF I LEQ 5                         % EXECUTES MYPROG 5 TIMES
    THEN GO TO DOITAGAIN ;
END JOB ;
```

SECTION 3: EXPRESSIONS, ITERATIONS, and FUNCTIONS

UNIT 2: Flow Control Statements

LEARNING SEQUENCE

The **WHILE** statement is used to execute a WFL statement when the condition is true. The statement to be executed may be a COMPOUND statement, which treats many statements as a single statement. If the condition is to be modified in the loop, care should be exercised to prevent an infinite loop. **(BE SURE TO READ THE CAUTION IN THE REFERENCE MANUAL.)**

- ```
  BEGIN JOB ;
    INTEGER I ;
    WHILE TRUE                    % INFINITE LOOP
      DO
        RUN MYPROG ;
  END JOB ;
  ```

- ```
  BEGIN JOB MYJOB ;
    INTEGER I, J ;
    I := 1 ;
    J := 1 ;
    WHILE I NEQ 5                 % INFINITE LOOP?
      DO
        BEGIN
          RUN MYPROG ;
          J := J + 1 ;
          I := I + J + 1 ;
        END ;
  END JOB ;
  ```

- ```
  BEGIN JOB MYJOB ;
    INTEGER I ;
    WHILE I LSS 5                 % EXECUTES MYPROG 5 TIMES
      DO
        BEGIN
          RUN MYPROG ;
          I := I + 1 ;
        END ;
  END JOB ;
  ```
  ```

SECTION 3: EXPRESSIONS, ITERATIONS, and FUNCTIONS

UNIT 2: Flow Control Statements

LEARNING SEQUENCE

The **DO** statement performs the same function as the WHILE statement. However, the WFL statement is executed before the boolean expression is evaluated and will be repeated until the expression is true. Note, that with this order of events, the WFL statement is always executed at least once, regardless of the value of the boolean expression. **(BE SURE TO READ THE CAUTION IN THE REFERENCE MANUAL.)**

-
```
BEGIN JOB ;
  INTEGER I ;
  DO                                  % INFINITE LOOP
      RUN MYPROG ;
    UNTIL FALSE ;
END JOB ;
```

-
```
BEGIN JOB MYJOB ;
  INTEGER I, J ;
  I := 1 ;
  J := 1 ;
  DO                                  % INFINITE LOOP?
      BEGIN
        RUN MYPROG ;
        J := J + 1 ;
        I := I + J + 1 ;
      END ;
    UNTIL I = 5 ;
END JOB ;
```

-
```
BEGIN JOB MYJOB ;
  INTEGER I ;
  DO                                  % EXECUTES MYPROG 5 TIMES
      BEGIN
        RUN MYPROG ;
        I := I + 1 ;
      END ;
    UNTIL I GEQ 5 ;
END JOB ;
```

SECTION 3: EXPRESSIONS, ITERATIONS, and FUNCTIONS

UNIT 2: Flow Control Statements

LEARNING SEQUENCE

The **CASE** statement allows dynamic selection of one out of several alternative statements depending on the value of the case expression. The case expression must be either an integer expression or string expression and be of the same type as the case constants provided.

```
BEGIN JOB EXAMPLE ;
  TASK T ;
  INTEGER I ;

  DO
     BEGIN
       WAIT("ENTER OK WHEN SRC/PROGA IS CORRECT",OK) ;
       COMPILE OBJ/PROGA WITH COBOL74 [ T ] LIBRARY ;
       COMPILER FILE CARD (KIND = DISK, TITLE=SRC/PROGA) ;
     END ;
   UNTIL T IS COMPILEDOK ;

  I := 0;

  WHILE I < 3
   DO
    BEGIN
      CASE I OF
        BEGIN
          (0)  :  DISPLAY " 1ST RUN OF PROGA" ;
          (1)  :  DISPLAY " 2ND RUN OF PROGA" ;
          (2)  :  DISPLAY " 3RD RUN OF PROGA" ;
          ELSE :  DISPLAY " INVALID VALUE OF I" ;
        END ;
      RUN OBJ/PROGA ;
      I := I + 1 ;
    END ;

END JOB ;
```

SECTION 3: EXPRESSIONS, ITERATIONS, and FUNCTIONS

UNIT 2: Flow Control Statements

PRACTICE EXERCISE

Study the WFL jobs below, and then answer the questions following each job.

```
BEGIN JOB A ;
 REAL COUNT ;
 COUNT := 7 ;

  WHILE COUNT GTR 0 DO
    BEGIN
      RUN X ;
        TASKVALUE = COUNT ;
      COUNT := COUNT - 2 ;

    END ;

END JOB ;
```

A1. HOW MANY TIMES WILL THE LOOP BE EXECUTED? _____

A2. WHAT WILL BE THE VALUE OF COUNT AFTER ALL ITERATIONS HAVE BEEN EXECUTED? _____

```
BEGIN JOB B ;
 REAL CTR ;
 TASK TX ;

 DO
   BEGIN
     RUN X [ TX ] ;
     CTR := CTR + 1 ;
   END;
 UNTIL TX IS COMPLETEDOK OR CTR GTR 3 ;

END JOB ;
```

B1. WHAT IS THE MINIMUM NUMBER OF TIMES THAT THE LOOP WILL BE EXECUTED? _____

B2. WHAT IS THE MAXIMUM NUMBER OF TIMES THAT THE LOOP WILL BE EXECUTED? _____

EXPRESSIONS, ITERATIONS, and FUNCTIONS

Unit 3

String Functions

OBJECTIVE

Construct statements using string functions.

PURPOSE

Many times it is useful to break apart messages to determine the exact flow desired, get the data to pass to particular task, or to find out system information such as the time or date.

RESOURCES

a. Student Materials	Section	3
	Unit	3
b. A Series WFL Reference Manual	Section	7

KEY WORDS

TAKE
DROP
HEAD
TAIL
ACCEPT
STRING
TIMEDATE
SYSTEM

SECTION 3: EXPRESSIONS, ITERATIONS, and FUNCTIONS

UNIT 3: String Functions

LEARNING SEQUENCE

A string function is a prewritten procedure provided for the programmer which accepts one or more arguments and returns a string value. The string returned may be used anywhere a string expression may be used. The six string functions are: TAKE, DROP, HEAD, TAIL, ACCEPT, and STRING.

The TAKE and DROP functions are used to extract a portion of a given string. The result is a string whose value is some number of characters from the beginning or end of another string. The **TAKE** function returns a new string whose value is a copy of the first indicated number of characters taken from the target string. The **DROP** function returns a new string whose value is a copy of the characters remaining in the target string after the first indicated number of characters have been discarded.

```
• S := "ABCDEFGHIJ" ;
• T := TAKE(S,5) ;                    % RESULTS IN T = "ABCDE"
• T := DROP("ABCDEFG",2) ;            % RESULTS IN T = "CDEFG"

• S := "OPQRST" ;
• T := DROP(S,4) & TAKE(S,2) ;        % RESULTS IN T = "STOP"

• R := "ABCDEFGHIJK" ;
• U := DROP(TAKE(R,9) ,7) ;           % RESULTS IN U = "HI"
```

SECTION 3: EXPRESSIONS, ITERATIONS, and FUNCTIONS

UNIT 3: String Functions

LEARNING SEQUENCE

The HEAD and TAIL functions extract a substring from a given string based upon a character set. The **HEAD** function returns a new string consisting of a copy of all the leading characters in the string that belong to a given character set. The **TAIL** function returns a new string consisting of a copy of all the characters in the string that remains after the removal of all the leading characters that belong to the character set.

```
• S1 := "ABCDEF" ;
• T := HEAD(S1,"A12B3") ;          % RESULTS IN T = "AB"
• U := TAIL(S1,"A12B3") ;          % RESULTS IN U= "CDEF"
```

In the following example, the TAIL function is performed first to supply a string expression to the HEAD function. The effect of the tail function using the "not character set" is to search for the first valid character and return the tail of the string from that point. The whole statement serves to strip leading and trailing garbage from an input statement.

```
• S2 := "AB122784XYZ" ;
• V := HEAD(TAIL(S2, NOT "0123456789"), "0123456789") ;
                                   % RESULTS IN V = "122784"
```

Is exactly the same as the following two statements:

```
• V := TAIL(S2, NOT "0123456789");
• V := HEAD(V, "0123456789");
```

LEARNING SEQUENCE

The **ACCEPT** function displays a message on the system ODT and waits for a reply. The job is suspended until the operator responds with an ODT command (mix number AX message). The message entered by the operator will be a string.

- ST1 := ACCEPT("ENTER CURRENT ACCOUNTING PERIOD") ;

The **STRING** function converts an integer into an equivalent string representation. The absolute value of the integer is converted to a string of the specified length. If an asterisk is used in place of specifying a length, the string returned will be precisely the number of digits needed to represent the integer.

- I1 := 432 ;
- I2 := 768 ;
- ST1 := STRING(I1,5) ; % RESULTS IN **ST1** = **"00432"**
- ST2 := STRING(I1,2) ; % RESULTS IN **ST2** = **"32"**
- ST3 := STRING(I2,4) ; % RESULTS IN **ST3** = **"0768"**
- ST4 := STRING(I2,*) ; % RESULTS IN **ST4** = **"768"**

LEARNING SEQUENCE

The **TIMEDATE** function returns the system time or date in various forms that may be used in the WFL job. It could be used for communication with the operator or as part of a file name.

- STR1 := TIMEDATE (HHMMSS) ;
- S2 := TIMEDATE (YYDDD) ;
- STR3 := TIMEDATE (MMDDYY) ;
- S2 := TIMEDATE (YYYYDDD) ;
- S3 := TIMEDATE (MMDDYYYY) ;

The **SYSTEM** function allows access to the system serial number, type, and MCP level. This may be useful if the flow of a job changes depending on the type of system it is executing on.

- STR3 := SYSTEM (SERIALNUMBER) ;
- STR1 := SYSTEM (TYPE) ;
- S2 := SYSTEM (MCPLEVEL) ;

SECTION 3: EXPRESSIONS, ITERATIONS, and FUNCTIONS

UNIT 3: String Functions

PRACTICE EXERCISE

In the space provided, write the letter of the description that best defines the string function.

_____1. ACCEPT FUNCTION	a. Returns a string whose value is a substring from a given string based upon a character set.
_____2. STRING FUNCTION	b. Returns a string entered by the operator.
_____3. TAKE/DROP FUNCTIONS	c. Converts an integer expression into a string expression.
_____4. HEAD/TAIL FUNCTIONS	d. Returns a string whose value is some number of characters from the beginning or end of another string expression.

PRACTICE EXERCISE (continued)

Study the WFL job below, and then determine the values of the strings just before the end of job.

```
BEGIN JOB C ;

 STRING FYLE, S1, S2, S3, S4, S5 ;


 FYLE := "ACCTS/PAYABLE/HISTORY" ;

 S1 := FYLE/"1984" ;                  S1 = _____

 S2 := TAKE(FYLE, 4) ;                S2 = _____

 S3 := DROP(FYLE, 6) ;                S3 = _____

 S4 := HEAD(FYLE, ALPHA) ;            S4 = _____

 S5 := TAIL(S1, NOT "1234567890") ;   S5 = _____

END JOB
```

EXPRESSIONS, ITERATIONS, and FUNCTIONS

Unit 4

Numeric Functions

OBJECTIVE

Construct statements using numeric functions.

PURPOSE

Since the only way to receive data from the operator is in the form of string data, if some of that information is to be used in arithmetic computation, a transformation of this data must occur. Numeric functions provide the transformation capability.

RESOURCES

a. Student Materials	Section	3
	Unit	4
b. A Series WFL Reference Manual	Section	7

KEY WORDS

LENGTH
HEX
OCTAL
DECIMAL
INTEGER

SECTION 3: EXPRESSIONS, ITERATIONS, and FUNCTIONS

UNIT 4: Numeric Functions

LEARNING SEQUENCE

Numeric functions are prewritten procedures provided for the programmer which may accept a string or real expression and returns a real or integer value.

The **LENGTH** function returns the number of characters in a given string as a real primary.

The **HEX**, **OCTAL**, and **DECIMAL** functions are used to convert a string containing either a hex, octal, or decimal number to its equivalent real value. A run time error will occur if the string contains invalid characters for the type of field being converted.

The **INTEGER** function returns the integer portion of a real expression. Since large system WFL supports real and integer arithmetic, the INTEGER function is used to convert a real expression to an integer.

```
• S1 := "25" ;                          % ORIGINAL STRING VALUE
• R1 := LENGTH("ABCDWXYZ") ;            % RESULTS IN R1 = 8
• R2 := HEX("1F") ;                     % RESULTS IN R2 = 31
• R3 := OCTAL("37") ;                   % RESULTS IN R3 = 31
• R4 := DECIMAL(S1) ;                   % RESULTS IN R4 = 25
• I5 := INTEGER(MYSELF(ACCUMPROCTIME));
                                        % RESULTS IN I5 = NUMBER
                                        % OF WHOLE SECONDS OF
                                        % PROCESSOR TIME FOR A
                                        % JOB TASK.
```

SECTION 3: EXPRESSIONS, ITERATIONS, and FUNCTIONS

UNIT 4: Numeric Functions

LEARNING SEQUENCE

- SAMPLE JOB USING STRINGS AND FUNCTIONS

```
00001000 BEGIN JOB PARSE ;
00001100    STRING S1, S2 ;
00001200    S1 := ACCEPT ("ENTER STRING TO BE PARSED") ;
00001300    S1 := TAIL (S1, " ") ;
00001400    WHILE LENGTH(S1) > 0    DO
00001500      BEGIN
00001600        S2 := HEAD (S1, ALPHA) ;
00001700        DISPLAY S2 ;
00001800        S1 := TAIL(S1, ALPHA) ;
00001900        S1 := TAIL(S1, " ") ;
00002000      END;
00002100 END JOB
```

INPUT: THIS IS A 6 TOKEN STRING

DISPLAYS:

THIS

IS

A

6

TOKEN

STRING

SECTION 3: EXPRESSIONS, ITERATIONS, and FUNCTIONS

UNIT 4: Numeric Functions

PRACTICE EXERCISE

A. In the space provided, write the letter of the description that best defines the integer function.

_____1. HEX FUNCTION	a. Returns the number of characters within a string expression.
_____2. DECIMAL FUNCTION	b. Returns the real expression without a fractional part.
_____3. LENGTH FUNCTION	c. Returns a real primary equal to the BASE 8 number represented by the value of the string expression.
_____4. INTEGER FUNCTION	d. Returns a real primary equal to the BASE 16 number represented by the value of the string expression.
_____5. OCTAL FUNCTION	e. Returns a real primary equal to the BASE 10 number represented by the value of the string expression.

EXPRESSIONS, ITERATIONS, and FUNCTIONS

Unit 5

Job Attributes

OBJECTIVES

Construct job attribute statements

PURPOSE

To further the understanding of attributes, this unit covers some of the attributes unique to the job task and others that may have unexpected influence on the results.

RESOURCES

a. Student Materials	Section 3 Unit 5
b. A Series WFL Reference Manual	Section 3

KEY WORDS

job attribute specification
job attribute assignment
class specification
family specification
fetch specification
starttime specification
usercode specification

SECTION 3: EXPRESSIONS, ITERATIONS, and FUNCTIONS

UNIT 5: Job Attributes

LEARNING SEQUENCE

The **job attribute specification** is used to assign attributes to the job task. There are six separate categories that fall under the job attribute specification.

The **job attribute assignment** allows the assignment of task attributes to the job itself.

The **class specification** allows the job to be assigned to a particular class or job queue.

The **family specification** allows the job to equate a target family with a substitute family.

The **fetch specification** causes the job to wait for operator action before beginning execution. When the job is initiated, the operator is informed that the job contains a FETCH message. The message may be displayed using the ODT command (PF). The job will start after the ODT command (OK) is used.

The **starttime specification** sets limitations on the job as to when it may start execution. The job, when started, will be placed in the appropriate queue and await the designated time before starting.

The **usercode specification** allows the usercode and password to be set. This is retained across a Halt/Load.

- BEGIN JOB BACKUP ;

  PRIORITY = 7 ;

  QUEUE = 30 ;

  FAMILY DISK = SYSTEMS OTHERWISE DISK ;

  FETCH = "MOUNT 2 SCRATCH TAPES FOR BACKUP" ;

  STARTTIME = 20:00 ON 02/01/85 ;

  USERCODE = PRODUCTION/MYPASS ;

  COPY INVENTORY/= FROM INVENT TO INVENTBKUP(KIND=TAPE) ;

  END JOB

PRACTICE EXERCISE

Write the job attribute section of a Work Flow job that does the following:

1. Informs the operator that the user disk PAYROLL should be mounted for this job.
2. Sets the priority of the job to 8.
3. Establishes the usercode/password for this job as PAY/ONE.
4. Restricts the processing time to a maximum of three minutes.
5. Starts the job at one minute after midnight.

```
BEGIN JOB MYJOB ;
_____________________________________________
_____________________________________________
_____________________________________________
_____________________________________________
_____________________________________________
_____________________________________________
END JOB
```

EXPRESSIONS, ITERATIONS, and FUNCTIONS

Unit 6

Job Parameters

OBJECTIVE

Construct job initiating statements that pass parameters.

PURPOSE

RESOURCES

a. Student Materials	Section 3 Unit 6
b. A Series WFL Reference Manual	Section 3

KEY WORDS

parameters

SECTION 3: EXPRESSIONS, ITERATIONS, and FUNCTIONS

UNIT 6: Job Parameters

LEARNING SEQUENCE

When starting a WFL job using the ODT command (START) it is possible to pass **parameters**, or values to the job. Whatever parameters are needed must first be specified within the BEGIN JOB statement. The type and name are specified and must be either REAL, INTEGER, BOOLEAN, or STRING. The parameters in the START command are checked against the parameter list in the job and must match by type and number. The variable declared as a job parameter may only be used as an expression. It may not be used on the left side of an assignment operator.

- START PARAMEXAMPLE("PAYROLL ON PAYPACK")

```
BEGIN JOB PARAMEXAMPLE (STRING S) ;
  RUN  # S ;
END JOB;
```

- START MYJOB ("12/31/84", 2)

```
BEGIN JOB MYJOB (STRING S1, REAL R1) ;
  STARTTIME = 2:00 ON 01/01/85 ;
  TASK T1 ;
  RUN MYPROG [ T1 ] ;
    VALUE = R1 ;
  IF T1 IS COMPLETEDOK
    THEN DISPLAY "MYPROG REPORT FOR " & S1
    ELSE DISPLAY "MYPROG FAILED" ;
END JOB
```

SECTION 3: EXPRESSIONS, ITERATIONS, and FUNCTIONS

UNIT 6: Job Parameters

PRACTICE EXERCISE

A. Write a Work Flow job that will compile a COBOL program. Be able to supply the object name and source name at START time. Also include the START statement that would be used to compile OBJ/PROGB using SRC/PROGB as the source disk file.

BEGIN JOB _____;

_____;

_____;

_____;

_____;

END JOB

START _____

B. Take the job written above and test it in the lab, using the source CLASS/PROGAOLD, and an object file name of your choice.

SECTION 3: EXPRESSIONS, ITERATIONS, and FUNCTIONS

LAB : 3-1

Write and execute a Work Flow job that will accept a person's name after the job has started. It will display the lengths of the first and last name separately, no matter how many spaces are included before, after, or in between the names. **SHOW THE INSTRUCTOR THE RESULTS.**

SAMPLE INPUT:

```
bbSUSANbbbWILSONbbbb          % b MEANS A BLANK OR SPACE
```

SAMPLE OUTPUT:

```
SUSAN CONTAINS 5 CHARACTERS

WILSON CONTAINS 6 CHARACTERS
```

Modify the job just created to accept a person's name as part of the start statement. **SHOW THE INSTRUCTOR THE RESULTS.**

SECTION 3: EXPRESSIONS, ITERATIONS, and FUNCTIONS

LAB : 3-2

Write a WFL job which includes the steps below:

1. Accepts a person's name from the ODT (or a CANDE terminal), then displays the name in reverse order.

2. Accepts a number from the ODT (or a CANDE terminal), then displays that number squared. Parse the input to make sure the data is valid input.

3. Each of the following strings must be included in your job. Extract and display the string "GEORGE WASHINGTON" using each of the functions TAKE, DROP, HEAD, and TAIL at least once.

```
STR1 := "GEORACF" ;
STR2 := "XYGE WA" ;
STR3 := "123SH4IN" ;
STR4 := "GPRSTONEF" ;
```

SHOW THE INSTRUCTOR THE RESULTS.

SECTION 3: EXPRESSIONS, ITERATIONS, and FUNCTIONS

LAB : 3-3

Additional labs to be supplied by the instructor.

SECTION 4

SUBROUTINES, CONTROL, and ERROR HANDLING

SUBROUTINES, CONTROL, and ERROR HANDLING

SECTION OBJECTIVE

Construct a WFL job using subroutines, advanced task control statements, and error handling techniques.

PURPOSE

To provide more efficient code generation, subroutines allow duplicate code to be discarded. With this in mind, they also allow error handling statements to be grouped together.

UNIT OBJECTIVES

- Construct a subroutine to execute a section of repetitive code.
- Construct communication statements to instruct an operator during the job.
- Construct statements for advanced task control.
- Construct a job using Global Files or Global Data Decks.
- Identify the functions of the WFL $ options.

SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 1

Subroutines

OBJECTIVE

Construct a subroutine to execute a section of repetitive code.

PURPOSE

To keep a repetitive job simple, WFL has the capability of executing a section of code just by identifying it and calling it.

RESOURCES

a. Student Materials	Section 4 Unit 1
b. A Series WFL Reference Manual	Section 5 Section 6

KEY WORDS

subroutine
local variables
global variables
RETURN
STOP
CALL-BY-NAME
CALL-BY-VALUE

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 1: Subroutines

LEARNING SEQUENCE

A subroutine is a construct that represents a single statement or group of statements identified by an identifier. The subroutine identifier must first be defined in the declaration part of the job along with the executable statements that belong to it. Afterwards, any reference to that subroutine identifier is interpreted as a command to execute the statements associated with that identifier. Therefore, a subroutine is a shorthand method of repeating the same WFL statements several times.

```
00000100 BEGIN JOB THISJOB ;
00000200   STRING STR1 ;
00000300   SUBROUTINE SUB1 ;
00000400     BEGIN
00000500       RUN PROG/A ;
00000600         FILE PATFILE = # STR1/PATIENTS ;
00000700         FILE DRFILE ( TITLE = # STR1/DOCTORS ) ;
00000800         FILE CHARGES ( TITLE = # STR1/CHARGES ) ;
00000900       RUN PROG/B ;
00001000         FILE PATFILE = # STR1/PATIENTS ;
00001100     END ;
00001200 STR1 := "HOSPITAL1" ;
00001300 SUB1 ;
00001400 STR1 := "HOSPITAL2" ;
00001500 SUB1 ;
00001600 STR1 := "HOSPITAL3" ;
00001700 SUB1 ;
00001800 END JOB ;
```

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 1: Subroutines

LEARNING SEQUENCE

Some declarations may occur within subroutines as well as those at the job level. All declarations made in the currently executing subroutine are called **local variables**. It is possible that variables declared in the calling routine may be accessible. Those variables that are accessible are called **global variables** with reference to the currently executing routine. This means that all variables declared at the job level are global to any subroutine and local to the job itself.

```
00000100 BEGIN JOB EXAMPLE2 ;
00000200   SUBROUTINE SUB2 ;
00000300     BEGIN
00000400       INTEGER I ;
00000500       STRING ST1 ;
00000600       WHILE I LSS 7 DO
00000700         BEGIN
00000800           ST1 := STRING(I,*) ;
00000900           RUN APROG ;
00001000             FILE INTA ( TITLE = AB/ # ST1 ) ;
00001100           I := I + 1 ;
00001200         END ;
00001300     END ;
00001400   SUB2 ;
00001500   RUN BPROG ;
00001600   SUB2 ;
00001700   RUN BPROG ;
00001800 END JOB
```

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 1: Subroutines

LEARNING SEQUENCE

The **RETURN** statement is used for early termination of the subroutine and returns control to the statement following the initiation of the subroutine.

```
00000100 BEGIN JOB SUBEXAMPLE3 ;
00000150
00000200    TASK T1 ;
00000250
00000300    SUBROUTINE SUB1 ;
00000400      BEGIN
00000500       TASK T2 ;
00000600       RUN PAYROLL/REPORT1 [ T2 ] ;
00000700       IF T2 IS COMPLETEDOK
00000800                AND
00000900           FILE PAYROLL/NEWMASTER IS RESIDENT
00001000         THEN RETURN ;
00001100       RUN PAYROLL/REPORT2 ;
00001200      END ;
00001250
00001300    RUN PAYROLL/UPDTWK [ T1 ] ;
00001350
00001400    SUB1 ;
00001450
00001500    IF T1 IS COMPLETEDOK
00001600             AND
00001700        FILE PAYROLL/MNTH IS RESIDENT
00001800     THEN
00001900       BEGIN
00002000         RUN PAYROLL/UPDTMNTH ;
00002100         SUB1 ;
00002200       END ;
00002250
00002300 END JOB
```

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 1: Subroutines

LEARNING SEQUENCE

The **STOP** statement allows the early termination of an asynchronous subroutine or the job itself. Both of these will be terminated normally if no dependent tasks are currently executing. If there are tasks executing, the parent and its dependent task are terminated abnormally. An optional string expression is available for displaying a message prior to termination.

```
00000100 BEGIN JOB SUBEXAMPLE3 ;
00000150
00000200    TASK T1 ;
00000250
00000300    SUBROUTINE SUB1 ;
00000400      BEGIN
00000500       TASK T2 ;
00000600       RUN PAYROLL/REPORT1 [ T2 ] ;
00000700       IF T2 IS COMPLETEDOK
00000800                AND
00000900          FILE PAYROLL/NEWMASTER IS RESIDENT
00001000         THEN STOP
00001100         ELSE RUN PAYROLL/REPORT2 ;
00001200      END ;
00001250
00001300    RUN PAYROLL/UPDTWK [ T1 ] ;
00001350
00001400    PROCESS SUB1 ;
00001450
00001500    IF T1 IS COMPLETEDOK
00001600             AND
00001700       FILE PAYROLL/MNTH IS RESIDENT
00001800      THEN
00001850        STOP "FIRST RUN OK" ;
00001900      ELSE
00001950        BEGIN
00002000          RUN PAYROLL/UPDTWKBKUP ;
00002100          SUB1 ;
00002200        END ;
00002250
00002300 END JOB
```

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 1: Subroutines

LEARNING SEQUENCE

The use of parameters when calling a subroutine operate in a similar manner to those covered in the job. When a value is passed to a subroutine, that value may be treated in one or two ways. The first, **CALL-BY-NAME**, is used when the parameter is only a link to the original area in memory set aside for the variable being passed. Therefore, when the parameter value is changed in the subroutine, it is actually changing the value of the original variable.

```
00000100 BEGIN JOB ROLLMONTH ;
00000200  REAL DATE1, DATE2 ;
00000250
00000300  SUBROUTINE GETDATE ( REAL R, STRING TEXT ) ;
00000400    BEGIN
00000500      STRING S ;
00000550
00000600      WHILE R LEQ 0 DO
00000700        BEGIN
00000800          S := ACCEPT ( TEXT ) ;
00000900          S := TAIL( S, NOT "0123456789" ) ;
00001000          S := HEAD (S, "0123456789") ;
00001100          R := DECIMAL ( S ) ;
00001200        END;
00001250
00001300    END;
00001325
00001350
00001375
00001400   GETDATE ( DATE1, "ENTER CURRENT MONTH END DATE" ) ;
00001450
00001500   RUN A/B ;
00001600     VALUE = DATE1 ;
00001650
00001700   GETDATE ( DATE2, "ENTER NEXT MONTH END DATE" ) ;
00001750
00001800   RUN A/C ;
00001900     VALUE = DATE2 ;
00001950
00002000 END JOB
```

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 1: Subroutines

LEARNING SEQUENCE

The second, **CALL-BY-VALUE,** is used when no link exists between the parameter and the original area of memory where the value is stored. The parameter is given the value of the original variable; but when the parameter value changes, it does not modify the original variable contents.

```
00000100 BEGIN JOB SQUARES ;
00000200  REAL X := 4, Y := 7, Z ;
00000300
00000400  SUBROUTINE SQUAREIT ( REAL R VALUE, REAL S ) ;
00000500    BEGIN                    %  R = 4, S = 0
00000600      REAL T ;
00000700      R := R * R ;
00000800      T := R ;
00000900      S := T + Y ;
00001100    END;
00001200
00001300  SUBROUTINE SQUAREITAGAIN ( REAL R , REAL S ) ;
00001400    BEGIN                    %  R = 7, S = 23
00001500      REAL T ;
00001600      R := R * R ;
00001700      T := R ;
00001800      S := T + X ;
00001900    END;
00002000
00002100  SQUAREIT ( X , Z ) ;              % X =  4 , Y =  7 , Z =  0
00002200                                    % X =  4 , Y =  7 , Z = 23
00002300  SQUAREITAGAIN ( Y , Z ) ;         % X =  4 , Y =  7 , Z = 23
00002400                                    % X =  4 , Y = 49 , Z = 53
00002500 END JOB
```

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 1: Subroutines

PRACTICE EXERCISE

Study the WFL job below, and then determine the values of the variables just before the end of job.

```
BEGIN JOB D ;

 REAL C, X, Y ;

 SUBROUTINE ADDER(REAL A, REAL B VALUE) ;

   BEGIN

     C := A + B ;          C = _____   A = _____   B = _____

     A := B + C ;          A = _____   B = _____   C = _____

     B := A + C ;          B = _____   A = _____   C = _____

 % <---------------------- What are the values of the identifiers at this point?

   END ;


 X := 4 ;
 Y := 2 ;
 C := 3 ;

 ADDER(X, Y) ;

END JOB
```

ANSWERS :

C = _____

X = _____

Y = _____

A = _____

B = _____

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 1: Subroutines

PRACTICE EXERCISE (continued)

Write a WFL job that contains a subroutine that runs REPORT/ALL. Within the main body of the job include the following:

1. Run PAY/ONE on PAYPACK.
2. Run REPORT/ALL.
3. Run PAY/TWO on PAYPACK.
4. If the file PAY/MASTER is on PAYPACK, run REPORT/ALL.
5. Run PAY/THREE.
6. If PAY/THREE completes to a normal end of task, run REPORT/ALL; otherwise, run PAY/FOUR.

SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 2

Communication with the Operator

OBJECTIVE

Construct communication statements to instruct an operator during the job.

PURPOSE

Operator communication is very important to the proper operation of any system. This is meant as a review of those means of communication already discussed and as an additional means of providing instructions to an operator only if the operator needs them.

RESOURCES

a. Student Materials	Section 4 Unit 2
b. A Series WFL Reference Manual	Section 5 Section 6

KEY WORDS

fetch specification
DISPLAY
WAIT
ABORT
STOP
INSTRUCTION
IB < number >

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 2: Communication with the Operator

LEARNING SEQUENCE

At this point, many statements have been covered to display messages to an operator before the job starts, during the job, and at termination.

- PRIOR TO JOB EXECUTION
  - The **fetch specification** causes the job to wait for operator action before beginning execution. When the job is initiated, the operator is informed that the job contains a FETCH message. The message may be displayed using the ODT command (PF). The job will start after the ODT command (OK) is used.

- DURING JOB EXECUTION
  - The **DISPLAY** statement displays a string expression on the ODT.
  - The **WAIT** statement causes the job task to suspend execution until a time period elapses in seconds or the OK ODT command is entered. A message can be displayed along with the wait.
  - The **ACCEPT** statement will display a message to an operator and suspend the task until the operator enters a response.

- DURING JOB TERMINATION
  - The **ABORT** statement causes the job task to be terminated abnormally. The STOP statement causes the job task to terminate normally. In both statements, optionally a message will be displayed prior to the termination.
  - The **STOP** statement allows the early termination of an asynchronous subroutine or the job itself. Both of these will be terminated normally if no dependent tasks are currently executing. If there are tasks executing, the parent and its dependent task are terminated abnormally. An optional string expression is available for displaying a message prior to termination.

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 2: Communication with the Operator

LEARNING SEQUENCE

The **INSTRUCTION** statement allows the WFL programmer to provide indepth instructions to the operator. Up to 63 separate instructions may be provided, each with a maximum length of 1500 characters. These instructions will be displayed only if the operator enters the ODT command (**<mix #> IB < number>**). This command is not available in CANDE.

```
00000100BEGIN JOB COMPILE/TESTS ;
00000200  FETCH = "INSTRUCTIONS ARE AVAILABLE" ;
00000300  INSTRUCTION 1
00000310    TESTTAPE IS IN TAPE RACK 3 ;
00000400  COPY & COMPARE = FROM TESTTAPE TO USERS(PACK) ;
00000500  INSTRUCTION 2
00000510    IF THE COPY FROM TESTTAPE TO USERS FAILS,
00000520       PLEASE LEAVE A NOTE FOR JK ;
00000700  COMPILE TEST/17 ALGOL ;
00000800    ALGOL FILE CARD (TITLE = USERS/FILE1, KIND = DISK) ;
00000900    FILE INP (TITLE = USERS/INP/TESTFILE) ;
00001000  IF FILE TEST/17 ISNT RESIDENT THEN ABORT "BAD COMPILE";
00001100END JOB
```

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 2: Communication with the Operator

PRACTICE EXERCISE

Write a WFL job that incorporates the following functions:

1. Run PROGA with a priority of 70 and task identifier T3.
2. Display a message indicating whether PROGA completed or not.
3. If PROGA failed, abort the job with the message "JOB ABORTED". Otherwise, send a message to the ODT that it completed ok and wait for an operator response.
4. Display a message to the operator that instructions are available prior to the start of the job.
5. The instructions are:
   A. PROGA is on the USERS pack.
   B. If the job aborts, run job BKUP.

SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 3

Exception Handling and Task Control

OBJECTIVE

Construct statements for advanced task control.

PURPOSE

Under certain conditions, the task control statements learned to this point may not be the most effective. WFL provides the ON statement to handle more difficult error conditions and flows.

RESOURCES

a. Student Materials	Section 4 Unit 3
b. A Series WFL Reference Manual	Section 5 Section 6

KEY WORDS

ON TASKFAULT
AUTORECOVERY
ON RESTART
OPTION
BDBASE
BDNAME
FAULT
DSED
ARRAYS
BASE
CODE
FILES

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 3: Exception Handling and Task Control

LEARNING SEQUENCE

A task fault occurs when any task of the job does not terminate normally. This can happen either because the task was DS'ed or, if the task was a compilation, it terminated finding syntax errors in the program source file. The **ON TASKFAULT** statement allows the user to specify that the job be terminated in the event of any task failure. The ON TASKFAULT alone disables the condition so that an abnormal task termination will not have any effect on the job.

- ON TASKFAULT ;
- ON TASKFAULT, < statement > ;
- ON TASKFAULT, ABORT "TASK HAS FAILED" ;

```
00000100 BEGIN JOB TF1 ;
00000200   RUN A ;
00000300   ON TASKFAULT, DISPLAY "TF1" ;
00000400   RUN B ;
00000500   RUN C ;
00000600   ON TASKFAULT, DISPLAY "TF2" ;
00000700   RUN D ;
00000800   ON TASKFAULT ;
00000900   RUN E ;
00001000 END JOB
```

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 3: Exception Handling and Task Control

LEARNING SEQUENCE

In this example, the ON TASKFAULT statement in line 900 resets the capability. Therefore, if a failure occurs in line 700 or 800, "TF2" may or may not be displayed since these are process statements and control is passed to line 900 as soon as the tasks are initiated.

```
00000100 BEGIN JOB TF1 ;
00000200   RUN A ;
00000300   ON TASKFAULT, DISPLAY "TF1" ;
00000400   PROCESS RUN B ;
00000500   RUN C ;
00000600   ON TASKFAULT, DISPLAY "TF2" ;
00000700   PROCESS RUN D ;
00000800   PROCESS RUN E ;
00000900   ON TASKFAULT ;
00001000   RUN F ;
00001100 END JOB
```

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 3: Exception Handling and Task Control

LEARNING SEQUENCE

There are times when the operator of a system has no control over when the system will HALT/LOAD. Power outages and fluctuations, software bugs and hardware failures may all cause this situation. WFL provides for these circumstances if the system option **AUTORECOVERY** is set. The job will begin execution at the last null mix unless the statement ON RESTART is used. The ON **RESTART** statement provides the capability of controlling the restart point.

- ON RESTART ;
- ON RESTART, < statement > ;
- ON RESTART, ABORT "TASK HAS FAILED" ;

```
00000100 BEGIN JOB TF1 ;
00000200    RUN A ;
00000300    ON RESTART, DISPLAY "TR1" ;
00000400    RUN B ;
00000500    RUN C ;
00000600    ON RESTART, DISPLAY "TR2" ;
00000700    RUN D ;
00000800    ON RESTART ;
00000900    RUN E ;
00001000 END JOB
```

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 3: Exception Handling and Task Control

LEARNING SEQUENCE

In this example, the ON RESTART statement in line 900 resets the capability. Therefore, if a failure occurs in line 700 or 800, "TR2" may or may not be displayed since these are process statements and control is passed to line 900 as soon as the tasks are initiated.

```
00000100 BEGIN JOB TF1 ;
00000200   RUN A ;
00000300   ON RESTART, DISPLAY "TR1" ;
00000400   PROCESS RUN B ;
00000500   RUN C ;
00000600   ON RESTART, DISPLAY "TR2" ;
00000700   PROCESS RUN D ;
00000800   PROCESS RUN E ;
00000900   ON RESTART ;
00001000   RUN F ;
00001100 END JOB
```

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 3: Exception Handling and Task Control

LEARNING SEQUENCE

In this example, the subroutine is used for restarting the job.

```
00000100 BEGIN JOB RESTARTIT ;
00000200   TASK T1 ;
00000300   SUBROUTINE RESTARTSUB ( REAL RESTARTCODE VALUE );
00000400     CASE RESTARTCODE OF
00000500      BEGIN
00000600       (0)  :  RUN RESTART ;
00000700       (1)  :  RUN RESTART1 ;
00000800       (2)  :  RUN RESTART2 ;
00000900       ELSE :  BEGIN
00001000                START RESTARTIT ;
00001100                ABORT "INVALID VALUE, JOB RESTARTED" ;
00001200               END ;
00001000      END;
00001100   ON RESTART, RESTARTSUB( 0 ) ;
00001200   RUN PROG ;
00001300   ON RESTART, RESTARTSUB( 1 ) ;
00001400   RUN PROG1 ;
00001500   ON RESTART, RESTARTSUB( 2 ) ;
00001600   RUN PROG2 ;
00001700 END JOB
```

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 3: Exception Handling and Task Control

LEARNING SEQUENCE

The **OPTION** attribute is used to set options for the process and allows additional capabilities when a fault occurs.

```
00000100 BEGIN JOB OPTIONEXAMPLE ;
00000200   OPTION = ( BDBASE ) ;
00000300   BDNAME = PRODUCTION/WFLJOB ;
00000400   RUN PRODUCTION/PROG1 ;
00000500     OPTION = ( FAULT, DSED, ARRAYS, BASE, CODE, FILES ) ;
00000600     FILE INP ( TITLE = PRODUCTION/IN ) ;
00000700   RUN PRODUCTION/PROG2 ;
00000800     FILE FILEOUT ( TITLE = PRODUCTION/OUT ) ;
00000900 END JOB
```

If PRODUCTION/PROG1 has an internal failure or if it is externally DSed, a program dump will occur automatically. The stack, arrays, base of the stack, code, and files pertaining to that execution of the program will be included in the dump.

```
00001000 BEGIN JOB OPTIONEXAMPLE2 ;
00001100   RUN PAYROLL/PROG1 ;
00001200     OPTION = (BDBASE) ;
00001300     BDNAME = PAYROLL/CHKS ;
00001400   RUN PAYROLL/PROG2 ;
00001500 END JOB
```

Previously, BDBASE had to be used in conjunction with the attribute BDNAME. The use of these attributes allowed the WFL programmer to modify the name of a printer backup file. In the first example, it will alter the names for the job, and in the second example only the names for the associated task. With release level 3.6, the entire printing subsystem has been changed and a closer look at these attributes occurs in section 5 of the student materials.

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 3: Exception Handling and Task Control

PRACTICE EXERCISE

In the space provided, write the letter of the description that best defines the statement.

_____1.	ON RESTART	a. Causes a program dump to occur if there is an internal fault.
_____2.	ON TASKFAULT	b. Causes the flow to be passed to a statement when a fault occurs.
_____3.	FAULT	c. Causes a descendant stack for printer backup files to be initiated as a job.
_____4.	BDNAME	d. Provides the directory name for printer backup files.
_____5.	DSED	e. Causes a particular statement to be executed after a HALT/LOAD.
_____6.	BDBASE	f. Causes a program dump when terminated externally.

SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 4

Global Files and Global Data Decks

OBJECTIVE

Construct a job using Global Files or Global Data Decks.

PURPOSE

In some circumstances, it would be useful to have control over files from within the WFL job. This unit deals with the control that is available from within a WFL job.

RESOURCES

a. Student Materials	Section 4 Unit 4
b. A Series WFL Reference Manual	Section 5 Section 6

KEY WORDS

Global Files
OPEN
CRUNCH
LOCK
PURGE
RELEASE
REWIND

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 4: Global Files and Global Data Decks

LEARNING SEQUENCE

WFL has the capability of dealing with files in many ways. As presented in Section 1, WFL may copy, remove, or change file names and security. **Global Files** allow the WFL programmer to do everything with physical files that a program can do except read and write to them. Global Files may only be declared in the declaration section of the job, not in a subroutine. These files may have certain file attributes assigned to them and then the whole file may be equated to a program's internal files. This allows different programs to have different internal file names and still use the same file without removing the physical file structure. Global Files have several statements that apply to them.

- **OPEN** — Opens the global file according to MYUSE attribute.
- **CRUNCH** — Intended for disk and pack, closes the file and returns the unused space at the end of the file to the system. A crunched file may not be expanded.
- **LOCK** — Intended for disk, pack and tape, closes the file and makes disk files permanent. Tape files are rewound and made unaccessible without operator action.
- **PURGE** — Intended for disk, pack and tape, closes the file and removes disk files from the system. Tapes are marked as scratch.
- **RELEASE** — Intended for all files, closes the file, disassociates the logical from the physical file, and returns the buffers. This is the system default. **CAUTION:** This will not save newly created files unless **PROTECTION** is set to save.
- **REWIND** — Intended for disk or tape, closes the file with retention, which means that the physical file is still assigned. For disk, the record pointer is adjusted to the first record. For tape, the tape is rewound.

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 4: Global Files and Global Data Decks

LEARNING SEQUENCE

```
00000100 BEGIN JOB GLOBAL1 ;
00000200   FILE GLOBEL (KIND = DISK, TITLE = USER/GLOBAL/DATA) ;
00000300   RUN USER/PROG1 ;
00000400     FILE OUTP := GLOBEL ;          % ASSIGNED AS OUTPUT
00000500   IF GLOBEL IS RESIDENT
00000600     THEN RUN USER/PROG/REPORT ;
00000700             FILE INP := GLOBEL ;    % ASSIGNED AS INPUT
00000800 END JOB
```

Global Files may have many different and useful purposes such as described below:

- CREATE MULTIFILE TAPES
  - Write to a data tape from many programs without closing or rewinding
- INCREASE SECURITY BY MATCHING WFL JOBS TO COBOL PROGRAMS
  - Open files in WFL, read and write in COBOL
- PASS VALUES BETWEEN PROGRAMS
  - As the above example
- PURGE TAPES
  - Open the file from WFL and close it with the PURGE statement

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 4: Global Files and Global Data Decks

LEARNING SEQUENCE

Global Data Decks must be declared in the same manner as Global Files. Access to these data decks is handled by the system and the programs using these decks. There may be more than one data deck available but none may have the same name.

- BEGIN JOB GLOBAL/DATA/DECK ;

```
  DATA CARD                      %  BEGINNING OF THE DATA DECK

5
10
15
?                                %  END OF THE DATA DECK

  RUN PROG/DATA/EXAMPLE ;        %  INTERNAL FILE CARD

  RUN PROG2/DATA/EXAMPLE ;       %  INTERNAL FILE CARD

END JOB
```

- BEGIN JOB GLOBAL/DATA/DECK ;

```
  DATA CHARSET1                  %  BEGINNING OF THE DATA DECK

ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789
[]<>()*+=-$&?/.,;:
?                                %  END OF THE DATA DECK

  DATA CHARSET2                  %  BEGINNING OF THE DATA DECK

ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789ABCDE
?                                %  END OF THE DATA DECK

  RUN PROG/DATA/EXAMPLE ;        %  INTERNAL FILE CHARSET1

  RUN PROG2/DATA/EXAMPLE ;       %  INTERNAL FILE CHARSET2

END JOB
```

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 4: Global Files and Global Data Decks

PRACTICE EXERCISE

Write a WFL job that incorporates the following functions:

1. Run a program whose name is PAYROLL that uses the global file TAXINFO for input and output. The internal file name is TAXES.

2. Run a report program PAYROLLTAX that uses the same global file as input only. The internal file name is TAXINFO.

3. The global file TAXINFO resides on disk by the name of TAXDATA.

4. The report program PAYROLLTAX also uses a card file called CARDIN whose card images are to be included in the WFL job. The cards will contain the name of the company, the name of the report, and the name of the department being reported on.

SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 5

WFL Compiler $ Options

OBJECTIVE

Identify the functions of the WFL $ options.

PURPOSE

To make full use of the WFL system, it is necessary to know what is available for use.

RESOURCES

a. Student Materials	Section Unit	4 5
b. A Series WFL Reference Manual	Section	9

KEY WORDS

$
ERRORLIMIT
NEWSOURCE
INCLUDE
CODE
LIST
SET
RESET
POP

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 5: WFL Compiler $ Options

LEARNING SEQUENCE

WFL Dollar Options provide additional flexibility and control over the WFL job being produced. The **$** must be in columns 1 or 2 of the card as with regular compiler options. The number of options for WFL is restricted. The option $ **ERRORLIMIT** is used to stop the compile after a certain number of errors. The job disposition **NEWSOURCE** will determine whether a new source file will be created. If present, the new source file is created including or excluding the $ options based on the location of the $ character.

It is possible to store sections of commonly used WFL source code in a library on disk. These sections of code may then be used by just merging these library files into the WFL source. The **$ INCLUDE** is used to do this as illustrated below.

- SOURCE PROGRAM

```
- BEGIN JOB LIBRARYEX (INTEGER I1)
                    NEWSOURCE = LIBRARYEXNEW ON SYSTEMS ;

  BOOLEAN COMPILEIT ;

  IF I1 = 1

    THEN COMPILEIT := TRUE

    ELSE COMPILEIT := FALSE ;

$INCLUDE LIB/ROUTINE ;

END JOB
```

- LIBRARY FILE (LIB/ROUTINE somewhere on the system)

```
-   IF COMPILEIT

      THEN COMPILE PROGA COBOL LIBRARY GO ;

        COBOL FILE CARD (KIND = DISK, TITLE = WFL/LIB/PROGA) ;

      ELSE RUN PROGA ;
```

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 5: WFL Compiler $ Options

LEARNING SEQUENCE

Sometimes if problems are occurring and there seems to be no simple explanation, it may be useful to get a code listing of the WFL job. The **$ SET CODE** function must be used with the **LIST** option to get a code listing with the address couple information for the job. The **SET, POP** and **RESET** functions may be used to restrict the listing to only those areas desired.

- ```
  00000100 $SET LIST CODE
  00000200 BEGIN JOB WFL/TEST ;
  00000300    INTEGER I ;
  00000400
  00000500    I := 5 ;
  00000600    I := I + 2 + 3 + 4 ;
  00000700
  00000800    DISPLAY STRING(I,*) ;
  00000900
  00001000 END JOB
  ```

- CODE GENERATED BY THIS SIMPLE JOB

```
100     00000100  $SET LIST CODE

200     00000200  BEGIN JOB WFL/TEST ;

300   (01,0002) = CODE SEGMENT DESCRIPTOR
400   (02,0002) = SKELETON STACK DESCRIPTION
500   (02,0003) = DATA DECK ARRAY
600   (02,0004) = ROLLOUT TIMESTAMP SLOT
700             0000:0   NOOP                FE

800     00000300     INTEGER I ;

900        DATA POOL: SEGMENT=0005  [5]
1000     800000000014 000001200000 000400000003 0C010203E6C6 D304E3C5E2E3
1100     081400000003 07000103D9E6 C40000000000 240400000004 0E5BE2C5E340
1200     D3C9E2E340C3 D6C4C5000000 260100000002 00000041808B 371200000005
1300     17021204C4C9 E2D209E2E8E2 E3C5D4E2C5C4 04C4C9E2D2A7 000000000000
1400  (02,0005) = I

1500    00000400
```
```

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 5: WFL Compiler $ Options

LEARNING SEQUENCE

- CODE GENERATED BY THE EXPRESSIONS IN LINE 500 AND 700

```
1600   00000500     I := 5 ;

1700            0000:1   LT8               B205          LINEINFO   00000500
1800            0000:3   NAMC (02,0005)    5005
1900            0000:5   STOD              B8

2000   00000600     I := I + 2 + 3 + 4 ;

2100            0001:0   VALC (02,0005)    1005          LINEINFO   00000600
2200            0001:2   LT8               B202
2300            0001:4   ADD               80
2400            0001:5   LT8               B203
2500            0002:1   ADD               80
2600            0002:2   LT8               B204
2700            0002:4   ADD               80
2800            0002:5   NTIA              86
2900            0003:0   NAMC (02,0005)    5005
3000            0003:2   STOD              B8

3100   00000700
```

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 5: WFL Compiler $ Options

LEARNING SEQUENCE

```
3200   00000800    DISPLAY STRING(I,*) ;

3300 (01,0003) = MESSER @ (0,0028)
3400            0003:3   MKST                AE              LINEINFO   00000800
3500            0003:4   NAMC (01,0003)     6003
3600            0004:0   LT48               BE000004000003
3700 (02,0006) = TEMPORARY STRING
3800 (02,0007) = TEMPORARY STRING LENGTH
3900            0006:0   ZERO               B0
4000            0006:1   NAMC (02,0006)     5006
4100            0006:3   INDX               A6
4200            0006:4   ZERO               B0
4300            0006:5   NAMC (02,0007)     5007
4400            0007:1   STOD               B8
4500 (02,0008) = EXPRESSION TEMPORARY
4600            0007:2   NAMC (02,0008)     5008
4700            0007:4   OVRN               BB
4800            0007:5   DUPL               B7
4900 (01,0004) = CCSTRINGFUNCTION @ (0,0034)
5000            0008:0   MKST               AE
5100            0008:1   NAMC (01,0004)     6004
5200            0008:3   NAMC (02,0008)     5008
5300            0008:5   LOAD               BD
5400            0009:0   VALC (02,0005)     1005
5500            0009:2   ZERO               B0
5600            0009:3   ONE                B1
5700            0009:4   ENTR               AB
5800            0009:5   DUPL               B7
5900            000A:0   VALC (02,0007)     1007
6000            000A:2   ADD                80
6100            000A:3   NAMC (02,0007)     5007
6200            000A:5   STOD               B8
6300            000B:0   TUNU               EE
6400            000B:1   DLET               B5
6500            000B:2   ZERO               B0
6600            000B:3   LT8                B206
6700            000B:5   TUND               E6
6800            000C:0   ZERO               B0
6900            000C:1   NAMC (02,0006)     5006
7000            000C:3   INDX               A6
7100            000C:4   ENTR               AB

7200   00000900
7300   00001000  END JOB          % AND 3 MORE PAGES NOT LISTED
```

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

Unit 5: WFL Compiler $ Options

PRACTICE EXERCISE

In the space provided, write the letter of the description that best defines the statement.

_____1.	LIST	A. This option will cause the compiler to generate a listing of the code along with the source listing.
_____2.	INCLUDE	B. This element will enable an updated WFL job symbol file to be generated.
_____3.	NEWSOURCE	C. This option will enable a separate WFL source listing to be printed.
_____4.	CODE	D. This option will set the maximum number of errors allowed by the WFL compiler for a single compilation.
_____5.	ERRORLIMIT	E. This option will allow the compiler to include the contents of library files as part of the source for a compilation.

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

LAB : 4-1

Write and run a Work Flow job which includes the steps described below. You may find it helpful to write, perform and check each step separately before adding the next step to the job.

Assure that all tasks within the job have a priority of 80 unless stated otherwise. The job should be limited to 10,000 lines of print.

STEP 1: Copy the files WFL/PROGASRC and WFL/PROGESRC from _____ to _____.

STEP 2: Allow time for the operator to load the correct paper.

STEP 3: Assure the COBOL source PROGESRC is available and compile PROGE putting a copy of the object on disk. The compile should be run at a priority of 60.

STEP 4: Run PROGE if it had no syntax errors from STEP 3. The input file, internally named ORIGINAL-FILE is named WFL/WFLDATA and should be on the systems disk. The two output files, INPUT-FILE and BKUP-FILE should be named SALESDATA and SLSDATABKP respectively. SLSDATABKP is a backup copy of SALESDATA.

STEP 5: Assure the COBOL source PROGASRC is available and compile PROGA putting a copy of the object on disk. Assign a priority of 35 using a task variable. If needed, WFL/PRGASRCBKP is another copy of PROGA's source.

STEP 6: Run PROGA if there were no syntax errors. The input file, internally named ORIGINAL-FILE, is SALESDATA. The output file, internally named DISK-FILE, should be named SALESMSTR.

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

LAB : (Continued)

STEP 7: Create a loop that will call a subroutine. The subroutine should run each of the programs listed below depending on an integer variable value.

The three programs create reports and the first two need SALESMSTR as input. The three programs are:

a. WFL/PROGB

b. WFL/PROGD

c. WFL/PROGC

STEP 8: Remove all files you created within this WFL job from the user disk.

SHOW THE INSTRUCTOR THE RESULTS.

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

LAB : 4-2

Write a WFL job which includes the steps below:

STEP 1: Execute UTIL/< yourname >/PROGA and UTIL/< yourname >/PROGB simultaneously. PROGB has an output file with the internal name of AUDITDISK. File equate this to a global disk file called UTIL/< yourname >/GLOBAL.

STEP 2: Write **one** "ON TASKFAULT" subroutine to handle possible failures of the tasks run in STEP 1 above.

A. **If PROGA fails then:**

1. Run CLASS/PROGD and file equate the internal file AUDIT to the global file built in STEP 1. PROGD will use this as an input file, so be sure to set the record pointer to the beginning of the file.

2. Rerun PROGA asynchronously with PROGB. Pass a taskvalue of 2 to PROGA.

3. If PROGA fails to run the second time, stop the entire job and give the reason for termination in the job summary.

B. **If PROGB fails then:**

1. Discontinue PROGA and rerun PROGA with a taskvalue of 1 passed to it.

2. Rerun PROGB.

3. If PROGB fails the second time, stop the entire job and give the reason for termination in the job summary.

STEP 3: After PROGA and PROGB run successfully, run CLASS/PROGC.

SHOW THE INSTRUCTOR THE RESULTS.

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

LAB : 4-3

Write a WFL job containing three subroutines to perform the sections of the string function lab from SECTION 3.

STEP 1: Make a new file for this job or get the previous lab as another name.

STEP 2: If the functions are in separate jobs, use the CANDE INSERT command to combine them into one file.

STEP 3: Put each of the three functions (NAME REVERSAL, SQUARING, & GEORGE WASHINGTON) into a separate subroutine.

STEP 4: Accept the name and number in the working section and pass them as parameters to the subroutines.

STEP 5: Include at least one local variable in a subroutine.

STEP 6: Display the results from either the subroutine or the working section.

EXTRA : Instead of accepting the information in the working section, pass the information needed as a job parameter.

SHOW THE INSTRUCTOR THE RESULTS.

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

LAB : 4-4

Additional labs to be supplied by the instructor.

SECTION 4: SUBROUTINES, CONTROL, and ERROR HANDLING

LAB : 4-5

Additional labs to be supplied by the instructor.

SECTION 5

UTILITIES

UTILITIES

SECTION OBJECTIVE

Construct a WFL job using different utilities to produce the desired results.

PURPOSE

The utilities provide the user with a means to obtain various information about the system and a way to manage the system resources. There are some basic utilities that almost every user needs to know.

UNIT OBJECTIVES

- Construct a WFL job to print out a file using the printing subsystem.
- Construct a WFL job to copy a file from one media to another using SYSTEM/DUMPALL.
- Construct a WFL job that uses SYSTEM/FILEDATA to generate reports on disk usage.
- Construct a WFL job that will copy files to a backup location using SYSTEM/FILECOPY.
- Construct a WFL job that will execute SYSTEM/LOGANALYZER to generate reports from system logging.
- Identify the uses of other utilities available on large systems.

UTILITIES

Unit 1

Printing Subsystem

OBJECTIVE

Construct a WFL job to print out a file using the printing subsystem.

PURPOSE

To keep the machine from becoming I/O bound on slow speed devices, such as a printer or punch, the MCP has some options that allows the output to these devices to be redirected to disk or tape. The printing subsystem is provided to manage the efficient use of these devices.

RESOURCES

a. Student Materials — Section 5, Unit 1

b. A Series Print System

c. A Series Printing Utilities

KEY WORDS

LPBDONLY
CPBDONLY
BACKUPBYJOBNR
SB
DLBACKUP
DL
OU
PS SHOWREQUEST
PS DELETE
DS
CL
PS STOP
PS OK
PS FORCE
SYSTEM/BACKUP

SECTION 5: UTILITIES

Unit 1: Printing Subsystem

LEARNING SEQUENCE

Printer and punch type peripheral devices are usually very slow in their handling of data. If many programs are trying to access a printer or punch, and only one may have control over it any point in time, then several programs are just waiting. This creates a problem in that the I/O cannot keep up with the processor which is more commonly referred to as I/O bound. To alleviate this situation, the printed or punched output from any or all programs may be redirected to disk or tape through the systems options **LPBDONLY** and **CPBDONLY**. The default for these options are to backup or put the files on disk or tape. A benefit derived from sending output files to disk or tape is that the utility may print these files at the first opportunity. It will also drive the device at close to its maximum output thus making better use of system resources.

The backup files are linked in a queue structure that provides two selection capabilities based on another system option **BACKUPBYJOBNR**. This option allows the operator to decide whether files will be output by job number or by the amount of lines generated by the job.

To get a job to execute properly and to get the desired results, it may be necessary to control the environment of the job, its tasks, and the output generated. Task attributes and file attributes are used for this purpose. With the level 3.6 release, many of these have enhanced capabilities when applied to printing. In addition, other attributes have been created to provide capabilites not available to the previous backup system.

SECTION 5: UTILITIES

Unit 1: Printing Subsystem

LEARNING SEQUENCE

These are some of the system options and attributes that were available for level 3.5 software and are still applicable to level 3.6 software.

SYSTEMS OPTIONS:

- **LPBDONLY** % OP+ SET - SPOOLED
  OP- RESET - DIRECT
- **CPBDONLY**
- **BACKUPBYJOBNR** % OP+ SET - BY JOB NUMBER
  OP- RESET - BY VOLUME
- **NOSUMMARY** % OP+ SET - NO SUMMARY IF A NORMAL EOJ OCCURS

TASK ATTRIBUTES:

- **BDBASE** % INITIATES A SEPARATE STACK
- **BDNAME** % PREFIX FOR BACKUP FILE NAME
- **JOBSUMMARY** % DEFAULT NOSUMMARY (SYSTEM OPTION)

FILE ATTRIBUTES:

- **BACKUPKIND** % BACKUP MEDIA TYPE
- **FORMID** % SPECIFIES THE TYPE OF FORM TO BE USED
- **KIND** % ORIGINAL DEVICE
- **PRINTERCONTROL** % INDICATES THE FILE NAME OF A PRINTER CONTROL FILE
- **PRINTERKIND** % TYPE OF PRINTER - IMAGE OR LP

SECTION 5: UTILITIES

Unit 1: Printing Subsystem

LEARNING SEQUENCE

Several ODT commands allow the operator flexibility as to where to store backup files. The ODT command (**SB**) allows the operator to equate any backup media type to another media type or **DLBACKUP** for better control of the system. This also allows the print system to find all backup at one location. The DLBACKUP option specifies that the system should look for a particular family based on the information given by the ODT command (**DL**). If a particular unit is not available, the MCP will respond with a message marked **RSVP**. The ODT command (**OU**) allows the operator to assign another output unit in place of the unavailable unit.

Other ODT commands allow the operator to control the process. The ODT command (**SP**) will show the current files waiting to be printed. The ODT command (**EP**) eliminates the print queue if AB is set to zero. To stop a file that is printing, there are three ODT commands available. The ODT command (**DS**) will terminate the current print task and remove the print file. The ODT command (**CL**) will terminate the current task, remove the print file, and reset the exception flags for that device. The ODT command (**QT**) will allow an operator to stop the task while not removing the print file. The task must be restarted manually at a later time unless the print queue is rebuilt. If you are stopping the task to print a special job, you might want to set AB to zero and initiate the task independently of AUTOBACKUP. To initiate a separate copy of AUTOBACKUP not under control of the AB command, the ODT command (**PB**) is used.

Since the printing subsystem must find the backup files being created and be able to know which files belong to which job, the filenames are in the following format.

```
* B D / J O B  N U M B E R / T A S K  N U M B E R / F I L E N A M E

* B D / 0 0 0 _____ / 0 0 0 _____ / 0 0 0 _____

* B D / 0 0 0 4 2 5 5 / 0 0 0 4 2 6 0 / 0 0 0 L I N E O U T
```

The attribute BDNAME is, therefore, used to modify the prefix of the printer backup filename. If modified, the printing subsystem will not be able to find the printfile automatically. Other means may be used to print it out such as the PB statement in WFL, WFL PB from CANDE, or the ODT command (**?PB**). These statements will initiate **SYSTEM/BACKUP** which has capability to do more specialized print request than does the printing subsystem. It allows one to specify the number of copies, special formatting, and where the output is to go. In addition, reference may be made by directory name or file name rather than by job number or mix number.

SECTION 5: UTILITIES

Unit 1: Printing Subsystem

LEARNING SEQUENCE

These are some of the ODT commands that were available for level 3.5 software and are still applicable to level 3.6 software. Some however have been implemented differently because of the new printing subsystem.

ODT COMMANDS:

- **SB** — % SUBSTITUTE BACKUP
  CHANGE MEDIA TYPE
- **DL** — % DISK LOCATION
  EQUATE MEDIA TO A FAMILY
- **OU** — % OUTPUT UNIT
  RESPONSE TO RSVP
- AB — % NEW 3.6 IMPLEMENTATION
  - WILL BE DE-IMPLEMENTED
- SP — % NEW 3.6 IMPLEMENTATION
  - WILL BE DE-IMPLEMENTED
- EP — % NEW 3.6 IMPLEMENTATION
  - WILL BE DE-IMPLEMENTED
- **QT** — % QUIT
  QUIT AND SAVE
- **CL** — % CLEAR
  QUIT AND REMOVE, CLEAR DEVICE
- **DS** — % DISCONTINUE
  QUIT AND REMOVE
- PB — % NEW 3.6 IMPLEMENTATION
  - WILL BE DE-IMPLEMENTED

- **?PB** — % PRINT BACKUP
  INITIATES SYSTEM/BACKUP

SECTION 5: UTILITIES

Unit 1: Printing Subsystem

LEARNING SEQUENCE

If the BDBASE option is set in addition to the BDNAME attribute, then a separate descendant stack is initiated for the print job. This means that the printout is treated as a separate job by the system and may become separated from the original job output since there is no guarantee that they will go to the same destination or will be printed at the same time.

```
BEGIN JOB WFLBACKUP ;
  RUN MYPROG/ONE ;
    OPTION = ( BDBASE ) ;
    BDNAME = CONCEPTS/CLASS ;          %  DL LOCATION
  PB "CONCEPTS/CLASS ON EDUCATION" ;
  RUN MYPROG/TWO ;
END JOB ;
```

```
JOB OUTPUT                          %  FROM MYPROG/TWO
  ( R W D ) * B D /0007299/0007300/000LINEOUTTWO
```

```
BDBASE OPTION OUTPUT                %  FROM MYPROG/ONE
  ( R W D ) C O N C E P T S / C L A S S /0007225/000LINEOUTONE
```

SECTION 5: UTILITIES

Unit 1: Printing Subsystem

LEARNING SEQUENCE

There are new system options and attributes for level 3.6 software which include the new printing subsystem. The implementation of the new printing subsystem offers new or enhanced task attributes and file attributes to control printed and punched output.

The Printing Subsystem offers the following **new task attributes:**

JOBSUMMARYTITLE	- DESIGNATES TITLE OF BACKUP FILE TO RECEIVE JOB SUMMARY.
NOJOBSUMMARYIO	- INHIBITS WRITING OF JOB SUMMARY INFORMATION TO THE JOB CODE FILE.
PRINTDEFAULTS	- SPECIFIES DEFAULT VALUES FOR PRINTING-RELATED FILE ATTRIBUTES.

The Printing Subsystem offers the following **new file attributes:**

1. **For routing and scheduling of print requests:**

AFTER	- DEFERS PRINTING TO LATER TIME.
DESTINATION	- SPECIFIES DESTINATIONS WITH OPTIONAL COPY COUNT PER DESTINATION. [DEFAULT VALUE IS SET FROM **DESTNAME**, WHICH CANDE EXTRACTS FROM CANDEDESTNAME IN THE USERDATAFILE.]
PRINTDISPOSITION	- SPECIFIES WHEN TO LOGICALLY QUEUE THE FILE FOR PRINTING.
PRINTERKIND	- INDICATES GENERIC PRINTER (IMAGE PRINTER, LINE PRINTER, OR DONTCARE).

LEARNING SEQUENCE

2. For backup file control:

PRINTCOPIES	- INDICATES NUMBER OF COPIES TO PRINT AT DESTINATION.
SAVEBACKUPFILE	- SPECIFIES WHETHER A BACKUP FILE SHOULD BE REMOVED OR NOT AFTER PRINTING.
SECURITYTYPE	- SET TO PRIVATE IF BACKUP FILE TITLE INCLUDES A USERCODE.
TITLE	- RETURNS ACTUAL BACKUP FILE NAME.
USERBACKUPNAME	- ALLOWS USER-SPECIFIED BACKUP FILE NAME.

3. For printing control:

BANNER	- PRECEDES FILE PRINTING BY BANNER PAGE.
FORMID	- INDICATES TYPE OF PAPER OR KIND OF FORM REQUIRED. [A STRING VALUE UP TO 100 CHARACTERS LONG.]
NOTE	- ALLOWS USER TO SUPPLY TEXT FOR BANNER PAGE.
PRINTERCONTROL	- INDICATES THE FILE NAME OF THE PRINTER CONTROL FILE.
TRANSFORM	- SPECIFIES LIBRARY ENTRYPOINT FOR TRANSFORMING DATA ON ITS WAY FROM BACKUP FILE TO PRINTER.

SECTION 5: UTILITIES

Unit 1: Printing Subsystem

LEARNING SEQUENCE

A new printing subsystem that replaces the autobackup facility is available on the Mark 3.6 release. The new subsystem supports the backup file creation and automatic printing features that were previously provided by Autobackup, as well as providing capabilities for backup file creation, routing, and printing control. Remote printing has been integrated into the new subsystem to make routing to remote printers as easy as routing to site printers and to allow remote printers to be controlled with the same operator commands that work for site printers.

All the old operator commands that were supported by Autobackup are still accepted and cause equivalent actions in the new Printing Subsystem. The only old command without a direct equivalent is EP. The new PS DELETE ALL command is roughly equivalent to EP except that it also removes any files queued for printing. The equivalent new syntax for the old Autobackup commands is shown below. Acceptable abbreviations are shown in upper case characters. The system automatically translates the old commands to the new commands for you, except for the EP command, which results in an informative error message.

Old syntax	New syntax
AB	PS SERVers
AB <number>	PS SERVers = <number>
AB <device>	PS DEVices + <device>
AB - <device>	PS DEVices - <device>
EP	PS DELete ALL
FORM <device>	PS DEVices <device>
FORM <device> <text>	PS CONFIGure <device> FORMid <text>
SP	PS SHowrequests

On level 3.6, some new ODT commands allow the operator to control the process. The ODT command (**PS SHOWREQUEST**) will show the current files waiting to be printed. The ODT command (**PS DELETE**) eliminates entries from the print queue and the associated backup files. To stop a file that is printing, there are three ODT commands available. The ODT command (**DS**) will terminate the current print task and remove the print file. The ODT command (**CL**) will terminate the current task, remove the print file, and reset the exception flags for that device. If you are stopping the task to print a special job, you might want to use the ODT command (**PS STOP**) which will allow an operator to suspend the printing while not removing the print file. The ODT command (**PS OK**) will restart printing of a stopped file. When a print request is initiated at the wrong time, it may be rescheduled by the ODT commands (**PS REQUEUE**) and (**PS MODIFY**). To initiate an immediate print request, the ODT command (**PS FORCE**) is used.

SECTION 5: UTILITIES

Unit 1: Printing Subsystem

LEARNING SEQUENCE

The Printing Subsystem offers the following **new operator commands.**

NEW OPERATOR COMMANDS:

PS

ADDFILES	ADD BACKUP FILES AND CREATE PRINT REQUESTS FOR THEM.
CONFIGURE	IDENTIFY DEVICE CHARACTERISTICS SUCH AS VOLUME LIMITS, FORMID, AND TRANSFORM FUNCTION.
DEFAULT	CONTROL PRINT-TIME DEFAULTS FOR ENTIRE SYSTEM, SUCH AS PRINTER SELECTION BY PRINTERKIND FOR PRINT JOBS VERSUS STANDALONE JOB SUMMARY FILES.
DELETE	DELETE PRINT REQUEST(S) AND REMOVE ANY ASSOCIATED BACKUP FILES.
DEVICES	LIST PRINTING DEVICES AND THEIR CONFIGURATIONS.
FORCE	FORCE REQUEST TO PRINT AS SOON AS POSSIBLE BY GIVING IT AN ARBITRARILY HIGH PRIORITY.
MODIFY	MODIFY ATTRIBUTES (INCLUDING DESTINATION) OF PENDING PRINT REQUEST.
OK	RESUME PRINTING ON A DEVICE THAT WAS STOPPED BY AN OPERATOR.
REQUEUE	STOP DEVICE AND REQUEUE PRINT REQUEST.
SERVERS	VIEW/CHANGE NUMBER OF PRINT SERVERS ALLOWED.
SHOWREQUESTS	DISPLAY PRINT REQUEST LIST (OPTIONALLY SHOW ONLY REQUESTS THAT ARE COMPLETED, WAITING, PRINTING, SCHEDULED, OR EXCEPTIONS). [ANY SHOWREQUESTS COMMAND MAY BE LIMITED TO REQUESTS FOR A PARTICULAR USERCODE.]
SKIP	SKIP FORWARD OR BACKWARD WITHIN A FILE OR SKIP FILES COMPLETELY DURING PRINTING.
STOP	TEMPORARILY STOP PRINTING ON A DEVICE.

SECTION 5: UTILITIES

Unit 1: Printing Subsystem

LEARNING SEQUENCE

A brief summary of the new printing subsystem appears below. The printing subsystem offers the following capabilities:

1. A programmatic interface through file attributes that gives the user control over backup file creation, routing, and printing.

2. The ability to set defaults for printing-related file attributes for a job, task, or MARC session through the task attribute PRINTDEFAULTS.

3. Extended operator control over printing devices and print jobs through the set of ODT commands that begin with the prefix PS.

4. The ability to manually create print requests with the PRINT statement in WFL.

5. Enhanced capabilities for viewing, copying, removing, and printing backup files via the Backup Processor utility that is available through MARC or CANDE.

6. The ability to optionally print job summary information, store it in a backup file for later analysis, or prevent job summary information from being written to the job code file through the task attributes JOBSUMMARY, JOBSUMMARYTITLE, and NOJOBSUMMARYIO, respectively.

7. Remote printing that is integrated with site printing and controlled by the same file attributes and operator commands.

SECTION 5: UTILITIES

Unit 1: Printing Subsystem

LEARNING SEQUENCE

The user/operator should be aware of differences in the following areas:

1. New programs/libraries

a. The PRINTSUPPORT function must be SLed to SYSTEM/PRINT/SUPPORT, a system library that maintains most of the state for the Printing Subsystem.

b. SYSTEM/PRINT/ROUTER is a process that will stay in the mix to handle asynchronous communications with the Printing Subsystem. It is responsible for creating printing requests, responding to ODT commands, and monitoring changes in printer status.

c. A new Backup Processor Utility that is called for CANDE or MARC users is provided by the codefile SYSTEM/PRINT/BACKUP/PROCESSOR. This replaces the internal BACKUPPROCESSOR that CANDE previously used.

d. Remote printing requires creation of a PRINTING window via the COMS UTILITY window and installation of the program titled SYSTEM/PRINT/REMOTE/SERVER. This program handles multiple remote printers and uses the library SYSTEM/PRINT/REMOTE/LIB to maintain state.

e. Site printing is done by a stack per printer that is labelled SERVER/<unit name>/R#<request number>/J#<job number> rather than the old naming convention AUTOPRINT/<unit name>/<job number>.

f. The stack that does tape printing follows the old naming convention with one exception: AUTOPRINT is replaced by TAPEPRINT.

SECTION 5: UTILITIES

Unit 1: Printing Subsystem

LEARNING SEQUENCE

2. New system files

a. Information regarding backup files and print requests is maintained in the system file SYSTEM/BACKUPFILELIST on the DL BACKUP family. If this file is not present at Halt/Load, the Printing Subsystem waits on a "NO FILE" condition. This wait allows the operator to copy in the file from another pack or from tape, or to enter "<mix number> OF" and have the system create the file by searching the directory of all disk packs for files that begin with the prefix "*BD" or "*BP". If this system file is present, the disk pack directory will NOT be searched for backup files.

Note that Autobackup was used to search the disk pack directory after Halt/Load or when the AB count went from zero to non-zero. Due to the existence of SYSTEM/BACKUPFILELIST, the disk pack directory is no longer searched automatically.

b. Information regarding printer characteristics is maintained in the system file SYSTEM/PRINTERINFO on the DL BACKUP family. If this file is not present at Halt/Load, the Printing Subsystem waits on a "NO FILE" condition, allowing the operator to copy in the file from another pack or from tape, or to enter "<mix number> OF" and have the system create the file.

SECTION 5: UTILITIES

Unit 1: Printing Subsystem

LEARNING SEQUENCE

3. Unit selection

a. Rather than considering ABed devices as "preferred" in the sense that Autobackup used, the new Print Subsystem uses the concept that printers can be added to or deleted from the "default printer pool". This pool of printers is used for servicing requests that do not specify a destination. A printer that does not belong to the default pool is used only when a request specifically names that printer as its destination.

b. Print requests are examined by the Printing Subsystem and split into separate requests if the files have different resource requirements. A request is not assigned to a Print Server stack unless the required resource is provided by that Server. If a request needs a resource that is unavailable or in use, the request is marked as waiting in the Print Request List, but it does not cause any task to wait. Therefore, Print Server stacks do not wait for "REQUIRES FORM: XYZ", for example.

c. When a Print Server stack is initiated, it only handles requests for a specific printer and never tries to re-assign itself to a different printer. The fact that requests are split according to resource requirements makes it possible for Servers to be dedicated to one device and prevents a Server from having to wait for "REQUIRES LP", for example. If a Server finishes printing a request and no more requests are compatible with that printer, the Server stack terminates.

SECTION 5: UTILITIES

Unit 1: Printing Subsystem

LEARNING SEQUENCE

Marc menus are the easiest way to execute the new commands. The following menus are samples of what you will see while executing the new commands from a privileged usercode. Limited options will appear on non-privileged user screens.

```
        MARC - MENU-ASSISTED RESOURCE CONTROL          11:14 AM        MARC
Action: [                                                              ]
        HOme PRev GO PArent COmnd                    (Press SPCFY for Help)

INTRO   Intro to MARC          SYS     System Control      NEWS   System News
JD      Job Display            CONFIG  System Config       DATE   Date and Time
JC      Job Control                                        USER   Usercode/Password
JQ      Job Queues             PK      Disk Pack           FILE   File Management
PS      Printing System        DK      Head-per-trk Disk   LIBS   System Libraries
                               MT      Magnetic Tape
RUN     Run A Task             LP      Line Printer        DC     DataComm Control
START   Start WFL Job          IP      Image Printer       BNA    BNA Commands
UTIL    System Utilities       CR      Card Reader         COMS   COMS Displays
                               CP      Card Punch          CC     COMS Control
MEM     Memory Management                                  SEND   Send Messages
SP      Special Programs       HC      Host Control        SC     Session Control
DUMPS   Dumps                  PROC    Processors          ON     Change Window
LOG     Logging                MM      Memory Modules      CANDE  Cande Window
SWAP    Swapper                OTHER   Other Devices

Choice: [                                                                  ]
```

```
        PS - PRINTING SYSTEM                           11:18 AM        MARC
Action: [                                                              ]
        HOme PRev GO PArent COmnd                    (Press SPCFY for Help)

STOP    Stop A Printer From Printing     CPBD    Show/Set CPBDONLY Option
OK      OK A Printer To Print            BYJOB   Show/Set BACKUPBYJOBNR
REQUE   Requeue A Request                DL      Specify Backup Family
SHOW    Show Print Requests
SKIP    Skip Printing A File             DEVICE  Show Printing Devices
FORCE   Force Printing Of A Request      SERV    Show/Set Print Servers
DELETE  Delete A Print Request           ADD     Add Printer to Default Pool
CONFIG  Configure A Print Device         REMOVE  Remove Printer From Pool
DECON   Deconfigure A Print Device
ADDF    Add Files To BACKUPFILELIST      SB      Show Backup Substitution
MODIFY  Modify Print Requests            CB      Change Backup Substitution
PRINT   Print Backup Files               DEF     Show/Set Default PRINTERKIND
PBMT    Print Tape Files                 JOBSUM  Show/Set JOBSUMMARY attribute
BACK    View/control printer files       TITLE   Show/Set JOBSUMMARYTITLE
LPBD    Show/Set LPBDONLY Option         PRDEF   Show/Set PRINTDEFAULTS

Choice: [                                                                  ]
```

SECTION 5: UTILITIES

Unit 1: Printing Subsystem

LEARNING SEQUENCE

The following screens offer examples of what you will see for specific requests.

```
        PSSHOW - SHOW PRINT REQUESTS                    11:22 AM
Action: [                                               (Press SPCFY for Help)]
        HOme PRev GO PArent COmnd

If Desired, Enter ONE Of The Following Options: . . . . . [    ]
   C = Show Completed Print Requests
   W = Show Waiting Print Requests
   S = Show Scheduled Print Requests
   E = Show Exception Requests
   P = Show Printing Requests
   Leave Blank = Show All Requests
(Enter USERCODE to limit display)     [                    ]
```

```
        FORCE - FORCE PRINTING OF A FILE                11:25 AM
Action: [                                               (Press SPCFY for Help)]
        HOme PRev GO PArent COmnd

Enter Request Number . . . . . [    ]
        or
Enter Job Number . . . . . [    ]
```

The **PRINT** command is new to level 3.6 and may be used from the ODT, CANDE, MARC, or WFL to print printer backup files. It provides a more flexible means of initiating print request than does SYSTEM/BACKUP. A user may specify file attributes and modifiers to be applied to the backup file when it is printed. These attributes and modifiers may also be changed any time before the print request is acted upon.

A new backup processor utility is provided for CANDE or MARC users called **SYSTEM/PRINT/BACKUP/PROCESSOR**. A new CANDE option **INTBACKUTIL**, decides which backup processor CANDE will use. When set to true, the internal backup processor is used and when set to false, it allows the MCP to decide which utitility to use.

SECTION 5: UTILITIES

Unit 1: Printing Subsystem

LEARNING SEQUENCE

The PRINT statement may be executed from MARC, CANDE, or WFL to create a print request. This command is used to print backup files only! It may be used to print a single backup file or an entire directory of backup files. File attributes of the backup files may be altered at execution time by specifying them in the print request. The following samples are written for execution from a WFL job. To make them work from MARC or CANDE insert the word WFL in front of the word PRINT.

To print 5 copies of the backup file MYPROG/PRINTOUTS

```
PRINT MYPROG/PRINTOUTS (PRINTCOPIES = 5) ;
```

To print the backup file BIG/REPORT after 8:00 p.m.

```
PRINT BIG/REPORT (AFTER = "2000") ;
```

To print the backup file MYSOURCE double spaced.

```
PRINT MYSOURCE; PRINTDEFAULTS = (DOUBLESPACE = TRUE) ;
```

To print the backup file PAYROLL/CHECKS on a printer with checks loaded and print 2 copies of the backup file PAYROLL/REPORTS on LP4.

```
PRINT PAYROLL/CHECKS (FORMID= "CHECKS"),
      PAYROLL/REPORTS(DESTINATION="LP4",PRINTCOPIES=2) ;
      PRINTDEFAULTS = (HEADER = UNCONDITIONAL) ;
```

SECTION 5: UTILITIES

Unit 1: Printing Subsystem

LEARNING SEQUENCE

On level 3.6 the BDNAME attribute may be used by itself without the option BDBASE. The *BD is replaced with the value of the usercode and the value assigned to BDNAME.

```
BEGIN JOB WFLBACKUP ;
  RUN MYPROG/ONE ;
    BDNAME = CONCEPTS/CLASS ;
END JOB ;
```

```
( R W D ) C O N C E P T S / C L A S S /0004888/0004893/000LINEOUT
```

SECTION 5: UTILITIES

Unit 1: Printing Subsystem

LEARNING SEQUENCE

For special forms, it is fairly simple to restrict the output to a single device or to load the form needed and start the output.

```
BEGIN JOB WFLFORMS ;
  RUN MYPROG/ONE ;
    FILE PRT(FORMID = "CHECKS") ;
END JOB
```

ON LEVEL 3.5

- CAUSES THE MESSAGE - MIX NUMBER REQUIRES FM: CHECKS
- RESPOND WITH - MIX NUMBER **FM** LP4

  OR BEFOREHAND - **FORM** LP5 "CHECKS"

ON LEVEL 3.6

- NO LONGER CAUSES THE MESSAGE - MIX NUMBER REQUIRES FM: CHECKS
- MARKS THE PRINT REQUEST AS WAITING RATHER THAN THE TASK - CHECK THE PRINT REQUEST LIST
- RESPOND WITH - PS DEVICES <DEVICE>

  OR BEFOREHAND - PS CONFIGURE<DEVICE> FORMID<TEXT>

SECTION 5: UTILITIES

Unit 1: Printing Subsystem

PRACTICE EXERCISE

Write a WFL job that executes the program PAY2 that outputs 2 printer files called OUT1 and OUT2. OUT1 must use three ply carbonless paper for its output, while OUT2 prints on a special type of check called CKTYPE2. The job requires that 5 copies of OUT1 be printed and that both backup files be kept for future use. These backup files are to be placed under the directory PAYHISTORY.

UTILITIES

Unit 2

System/Dumpall

OBJECTIVE

Construct a WFL job to copy a file from one media to another using SYSTEM/DUMPALL.

PURPOSE

There are many times that files may be received on foreign media. It may be necessary to modify these files as they are copied to be usable on this system.

RESOURCES

a. Student Materials — Section 5, Unit 2

b. A Series System Software Utilities — Section 4

KEY WORDS

PARAMETER MODE
CARD MODE
INTERACTIVE MODE

SECTION 5: UTILITIES

Unit 2: System/Dumpall

LEARNING SEQUENCE

The SYSTEM/DUMPALL utility is a rather comprehensive media conversion program. It is capable of copying files from one media to another, listing files on the line printer, or generating card decks. These files may be labeled or unlabeled on either input or output. Any FILEKIND may be copied from any media device to any media device. If it has a standard label, it is used; otherwise, the user must supply the information to the utility. The utility may be used for changing the attributes of a file during the copy process.

SYSTEM/DUMPALL executes in three basic modes of operation. In all three modes, commands are required to perform the operations desired.

- **PARAMETER MODE**
  - RUN SYSTEM/DUMPALL (" ")
    - % COMMANDS ARE PASSED AS PARAMETERS

- **CARD MODE**
  - RUN SYSTEM/DUMPALL ("CARD")

    FILE CARD (TITLE = _____ ON _____)
    - % COMMANDS ARE PASSED AS CARD IMAGES

- **INTERACTIVE MODE**
  - RUN SYSTEM/DUMPALL ("INTER")
    - % COMMANDS ARE PASSED INTERACTIVELY FROM A REMOTE TERMINAL

SECTION 5: UTILITIES

Unit 2: System/Dumpall

LEARNING SEQUENCE

The following job executes SYSTEM/DUMPALL in the **parameter mode** from a WFL job. It may also be executed from CANDE in the same manner.

```
100  BEGIN JOB UTIL3;
120   %
130    RUN *SYSTEM/DUMPALL
140      ("L (RWD)CONCEPTS/CLASS/UTIL3");
150   %
160  END JOB
```

The output from the job above is shown below:

```
100 B6900:2372  DUMPALL  VERSION 34.180.032  TUESDAY, 02/05/85 02:39 PM.
200
300
400
500 L (RWD)CONCEPTS/CLASS/UTIL3
600
700 FILE ATTRIBUTES FOR: FIN  TITLE=(RWD)CONCEPTS/CLASS/UTIL3 ON
    SYSTEMSED  HOSTNAME=TRAINING1  KIND=PACK  INTMODE=EBCDIC
    EXTMODE=EBCDIC  FILETYPE=0  MINRECSIZE=0  MAXRECSIZE=15
    BLOCKSIZE=420  FRAMESIZE=48  FILEUSE=IO  BUFFERS=2
    TRANSLATE=FULLTRANS PROTECTION=SAVE  POPULATION=1  AREAS=1
    AREALENGTH=15120  FLEXIBLE  LASTRECORD=7  FILEKIND=JOBSYMBOL
    ROWSINUSE=1  CRUNCHED USERINFO=000000000000
    CREATIONDATE=02/05/85(85036)  LASTACCESSDATE=02/05/85(85036)
    CYCLE=1  VERSION=0  SAVEFACTOR=30 SECURITYTYPE=PRIVATE
    SECURITYUSE=IO  PACKNAME=SYSTEMSED

1200
1300
1400      1E! BEGIN JOB UTIL3;
1500      2E!  %
1600      3E!  RUN *SYSTEM/DUMPALL
1700      4E!    ("L (RWD)CONCEPTS/CLASS/UTIL3");
1800      5E!
1900      6E!  END JOB

2000 EOF - FILE CONTAINS 6 RECORDS
2100 6 RECORDS PROCESSED
```

SECTION 5: UTILITIES

Unit 2: System/Dumpall

LEARNING SEQUENCE

The following job executes SYSTEM/DUMPALL in the **card mode** from a WFL job. It may also be executed from CANDE in the same manner.

```
BEGIN JOB DUMPALLCARD1 (STRING S1) ;
 RUN *SYSTEM/DUMPALL("CARD") ;
     FILE CARD (TITLE=#S1 ON SYSTEMSED)

END JOB
```

The parameters to SYSTEM/DUMPALL may be placed in an ordinary CANDE data file. The contents of the data file may contain any of the input parameters available for by SYSTEM/DUMPALL. Some examples are:

ATTributes <file title> (file attribute<s>)

List the attributes of the file specified.

CAT <file title> THEN <file title> GIVING <file specification>

Concatenates the first file to the second file creating a new file.

COPY <file title> TO <file title> (file attributes)

Copies the first file to the second file name using the attributes specified to create the output file.

LIST <file title> RECORD 5 H

List record 5 of the file specified in hex.

SECTION 5: UTILITIES

Unit 2: System/Dumpall

LEARNING SEQUENCE

The following example shows the use of Dumpall using the card mode and global files to dynamically pass the name of a file created in a program. Indirectly, it also shows how to build a file name in an Algol program for a printer backup file. The same procedure may be used in WFL to generate a file name if the internal name of the file is known.

```
BEGIN JOB DUMPALLCARD2 ;
 DESTNAME  = B20FINAL ;
 STRING S1  ;
 FILE PASSIT (KIND=DISK, MAXRECSIZE=80, UNITS= CHARACTERS);
  S1 := "THIS IS A TEST STRING" ;
  RUN OBJECT/TEST/PROGA(S1) ;
  RUN *SYSTEM/DUMPALL("CARD") ;
    FILE CARD := PASSIT ;
END JOB
```

```
$LEVEL 2
PROCEDURE TESTIT (S1) ;
  ARRAY   S1 [*] ;
  BEGIN
  STRING S2, S3 ;
  FILE PRT (KIND = PRINTER) ;
  FILE PASSIT(KIND=DISK, MAXRECSIZE=80, UNITS=CHARACTERS,
              PROTECTION=SAVE) ;
  THRU  5 DO WRITE(PRT,<A6>,S1) ;
  S3 := "LISTAN *BD" CAT "/"
        "000" CAT  STRING(MYSELF.JOBNUMBER,*) CAT "/"
        "000" CAT  STRING(MYSELF.MIXNUMBER,*) CAT "/"
        "000" CAT  "PRT" ;
  WRITE (PASSIT,/,S3);
  DISPLAY (S1);
  DISPLAY (S3);
  WAIT((5)) ;
 END.
```

SECTION 5: UTILITIES

Unit 2: System/Dumpall

LEARNING SEQUENCE

The following job executes SYSTEM/DUMPALL in the **interactive mode**. The items that are highlighted are the commands that are transmitted from a terminal. The interactive mode is especially useful to the programmer that wishes to check on a record that was just updated by a new program. It is not limited to specific file formats as is CANDE.

```
RUN *SYSTEM/DUMPALL("INTER")
 RUNNING 2525
 ?
B6900:2372  DUMPALL  VERSION 34.180.032  WEDNESDAY, 02/06/85 01:49 PM.
PLEASE ENTER DUMPALL COMMAND

LIST CONCEPTS/CLASS/UTIL1

FILE ATTRIBUTES FOR: FIN  TITLE=(RWD)CONCEPTS/CLASS/UTIL1 ON SYSTEMSED
HOSTNAME=TRAINING1  KIND=PACK  INTMODE=EBCDIC  EXTMODE=EBCDIC  FILETYPE=0
MINRECSIZE=0  MAXRECSIZE=15  BLOCKSIZE=420  FRAMESIZE=48  FILEUSE=IO
BUFFERS=2 TRANSLATE=FULLTRANS  PROTECTION=SAVE  POPULATION=1  AREAS=1
AREALENGTH=15120  FLEXIBLE  LASTRECORD=7  FILEKIND=JOBSYMBOL  ROWSINUSE=1
CRUNCHED  USERINFO=000000000000  CREATIONDATE=02/05/85(85036)
LASTACCESSDATE=02/06/85(85037)  CYCLE=1  VERSION=0  SAVEFACTOR=30
SECURITYTYPE=PRIVATE  SECURITYUSE=IO  PACKNAME=SYSTEMSED


CONT

        1E! BEGIN JOB UTIL1;
(0000068)E!                 00000100...90

CONT

        2E! %
(0000068)E!                 00000200...90

CONT

        3E!  RUN SYSTEM/FILEDATA
(0000068)E!                 00000300...90
```

LEARNING SEQUENCE

The commands shown on this page show the same record in different data formats:
R = Real, D = Decimal, E = EBCDIC, H = HEX, A = ASCII, and O = Octal.

R

```
       3R!   -3088730162960.0 !   -1.8861720541E-4 ! -1.74319633488E+23 !
(0000003)R!     -133634.007843 !   -275955859520.0 !    -275955859520.0 !
(0000006)R!    -275955859520.0 !   -275955859520.0 !    -275955859520.0 !
(0000009)R!    -275955859520.0 !   -275955859520.0 !    -275955859520.0 !
(0000012)R!    -275955859520.0 !   -275955904752.0 !  -7.6531949077E-19 !..15
```

D

```
       3D! REAL          ! ALPHA         ! ALPHA         ! ALPHA         !
(0000004)D! -275955859520 ! -275955859520 ! -275955859520 ! -275955859520 !
(0000008)D! -275955859520 ! -275955859520 ! -275955859520 ! -275955859520 !
(0000012)D! -275955859520 ! -275955904752 ! ALPHA         !..15
```

E

```
       3E!  RUN SYSTEM/FILEDATA
(0000068)E!                 00000300...90
```

H

```
       3H! 40D9E4D540E2 ! E8E2E3C5D461 ! C6C9D3C5C4C1 ! E3C140404040 !
(0000048)H! 404040404040 ! 404040404040 ! 404040404040 ! 404040404040 !
(0000096)H! 404040404040 ! 404040404040 ! 404040404040 ! 404040404040 !
(0000144)H! 404040404040 ! 40404040F0F0 ! F0F0F0F3F0F0 !...180
```

A

```
       3A! @???@??????a????????@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
(0000068)A! @@@@@@@@@@@@@@????????...90
```

O

```
       3O! 2015474465240342 ! 7216134361352141 ! 6154472361342301 !
(0000048)O! 7074050020040100 ! 2004010020040100 ! 2004010020040100 !
(0000096)O! 2004010020040100 ! 2004010020040100 ! 2004010020040100 !
(0000144)O! 2004010020040100 ! 2004010020040100 ! 2004010020040100 !
(0000192)O! 2004010020040100 ! 2004010020170360 ! 7417036074770360 !...240
```

SECTION 5: UTILITIES

Unit 2: System/Dumpall

LEARNING SEQUENCE

The commands on this page are used to locate a particular record within a file.

E

```
        3E!  RUN SYSTEM/FILEDATA
(0000068)E!                  00000300...90
```

RECORD 1

```
        1E! BEGIN JOB UTIL1;
(0000068)E!                  00000100...90
```

SKIP+2

```
        3E!  RUN SYSTEM/FILEDATA
(0000068)E!                  00000300...90
```

SKIP-1

```
        2E! %
(0000068)E!                  00000200...90
```

NEXT

```
        3E!  RUN SYSTEM/FILEDATA
(0000068)E!                  00000300...90
```

AGAIN

```
        3E!  RUN SYSTEM/FILEDATA
(0000068)E!                  00000300...90
```

CONT

```
        4E!          STATION=139;
(0000068)E!                  00000400...90
QUIT
9 RECORDS PROCESSED
END OF LIST ROUTINE
PLEASE ENTER NEXT COMMAND
QUIT
```

The first QUIT exits the list command and the second QUIT exits DUMPALL.

SECTION 5: UTILITIES

Unit 2: System/Dumpall

PRACTICE EXERCISE

Write a WFL job to execute SYSTEM/DUMPALL that will list the file USER/ONE/DATA ON PRODUCTION. After listing the file, copy the file to a tape called BKUP. Execute DUMPALL only **once** from the job.

UTILITIES

Unit 3

System/Filedata

OBJECTIVE

Construct a WFL job that uses SYSTEM/FILEDATA to generate reports on disk usage.

PURPOSE

This utility provides the capability to generate reports about the files and their storage on a system.

RESOURCES

a.	Student Materials	Section 5 Unit 3
b.	A Series System Software Utilities	Section 6

KEY WORDS

SECTION 5: UTILITIES

Unit 3: System/Filedata

LEARNING SEQUENCE

The SYSTEM/FILEDATA utility produces various reports regarding files. The reports may give such information as a hierarchical list of files, a map of file layout, a layout of all permanent files and the space around them, the attributes of a file or files, a list of files on tape or tapes, catalogue information for a file or files, or creating a "report" file that is usable by library maintenance. The kinds of reports that may be produced also include user-defined reports.

There are three ways in which SYSTEM/FILEDATA may be invoked:

- ODT COMMANDS
- WFL COMMANDS
- CANDE COMMANDS

The WFL job below executes SYSTEM/FILEDATA to find the attributes CREATIONDATE AND LASTACCESS of the files in the directory CONCEPTS under the usercode RWD. The output is directed to the terminal.

```
00000100 BEGIN JOB UTIL1 ;
00000200 %
00000300  RUN SYSTEM/FILEDATA
00000400    ("A:DIR = (RWD)CONCEPTS ON SYSTEMSED LASTA CRE SCREEN") ;
00000450         STATION=139 ;
00000500 %
00000600 %
00000700 END JOB ;
```

The output received at the terminal is shown on the next page. This type of redirection of the output is dependent on the device being used and the structure of the report.

LEARNING SEQUENCE

```
          BURROUGHS LARGE SYSTEMS 2372 REPORT
        OF 02/05/85 AT 14:43:18.  VERSION 35.200

FAMILY = SYSTEMSED.  ON UNIT      50
SERIAL NUMBER = 814114    CREATED  3/14/84.
                                          SELECTED FILE ATTRIBUTES

(RWD)
. CONCEPTS
. . CLASS
. . . 3
. . . . DATA : CREATIONDATE= 7/31/84 @ 15:03:44
                LASTACCESSDATE= 1/14/85 @ 18:56:46
. . . JOB
. . . . 1 : CREATIONDATE= 7/31/84 @ 15:13:31
             LASTACCESSDATE= 1/14/85 @ 18:47:13
. . . WFL : CREATIONDATE= 7/26/84 @ 16:09:58
            LASTACCESSDATE= 2/01/85 @ 14:40:01
. . . WFL2 : CREATIONDATE= 7/26/84 @ 17:09:15
             LASTACCESSDATE= 1/14/85 @ 18:48:26
. . . NEWBB : CREATIONDATE= 7/26/84 @ 17:14:45
              LASTACCESSDATE= 1/14/85 @ 18:56:16
. . . PROGA : CREATIONDATE= 7/26/84 @ 17:14:40
              LASTACCESSDATE=10/08/84 @ 13:27:33
. . . PROGB : CREATIONDATE= 3/01/77 @ 00:00:00
              LASTACCESSDATE= 7/26/79 @ 00:00:00
. . . UTIL1 : CREATIONDATE= 2/05/85 @ 09:33:50
              LASTACCESSDATE= 2/05/85 @ 14:43:13
. . . UTIL3 : CREATIONDATE= 2/05/85 @ 11:37:01
              LASTACCESSDATE= 2/05/85 @ 14:39:08
. . . FILECOPY : CREATIONDATE= 7/31/84 @ 15:42:16
                 LASTACCESSDATE= 1/14/85 @ 19:02:07
. . . PROGANEW : CREATIONDATE= 3/01/77 @ 00:00:00
                 LASTACCESSDATE= 1/14/85 @ 18:53:46
. . . PROGBNEW : CREATIONDATE= 3/01/77 @ 00:00:00
                 LASTACCESSDATE= 1/14/85 @ 18:54:42
. . . TASKVALUE : CREATIONDATE=10/08/84 @ 13:32:14
                  LASTACCESSDATE= 1/14/85 @ 18:55:42

          SELECTED FILE ATTRIBUTES

          FILE ATTRIBUTES INPUT WAS:

"A:DIR = (RWD)CONCEPTS ON SYSTEMSED LASTA CRE SCREEN"
```

LEARNING SEQUENCE

The example below failed because the report being generated had a line width of 132 characters. In this instance, there are other options available.

```
00000100BEGIN JOB UTIL1 ;
00000200%
00000300 RUN SYSTEM/FILEDATA
00000400  ("F:DIR = (RWD)CONCEPTS ON SYSTEMSED NA TTY") ;
00000450          STATION=139 ;
00000500%
00000600%
00000700END JOB ;
```

```
              LISTDIRECTORY INPUT WAS:

    "F:DIR = (RWD)CONCEPTS ON SYSTEMSED NA TTY"

**ERROR**   OUTPUT REQUIRES GREATER LINE WIDTH.
```

The following example illustrates a way to retrieve the information from a remote terminal using a WFL job and the CANDE command BACK.

```
00000100 BEGIN JOB UTIL1 ;
00000200 %
00000210   OPTION =(BDBASE) ;
00000220   BDNAME = RWD ;
00000300   RUN SYSTEM/FILEDATA
00000400     ("F : DIR = (RWD)CONCEPTS/CLASS ON SYSTEMSED NA PR") ;
00000500 %
00000600 END JOB ;
```

Once the job has been executed, then you may look at the printer backup files designated by RWD with the BACK command.

```
BACK RWD ON EDUCATION          % ASSUMES THAT BACKUP FILES ARE
                                 GOING TO EDUCATION

  LIST                         % SUB COMMAND OF BACK
                                 ALLOWS YOU TO LOOK AT A BACKUP
                                 FILE AT A TERMINAL
```

SECTION 5: UTILITIES

Unit 3: System/Filedata

LEARNING SEQUENCE

```
100
200              BURROUGHS LARGE SYSTEMS 2372 REPORT
                OF 02/05/85 AT 15:09:39.  VERSION 35.200
300
400 FAMILY = SYSTEMSED.
500              HIERARCHICAL DIRECTORY STRUCTURE.                PAGE 1
600
700                                          (RWD)
800                                          . CONCEPTS
900                                          . . CLASS
1000                                         . . . 3
1100      1          DATA                    . . . . DATA
1200                                         . . . JOB
1300      2          JOBSYMBOL               . . . . 1
1400      3          JOBSYMBOL               . . . WFL
1500      4          JOBSYMBOL               . . . WFL2
1600      5          ALGOLSYMBOL             . . . NEWBB
1700      6          COBOLCODE               . . . PROGA
1800      7          ALGOLCODE               . . . PROGB
1900      8          JOBSYMBOL               . . . UTIL1
2000      9          JOBSYMBOL               . . . UTIL3
2100     10          JOBSYMBOL               . . . WFL2B
2200     11          JOBSYMBOL               . . . WFL3A
2300     12          JOBSYMBOL               . . . WFL3B
2400     13          JOBSYMBOL               . . . WFL4A
2500     14          JOBSYMBOL               . . . UTIL11
2600     15          DATA                    . . . FILECOPY
2700     16          COBOLSYMBOL             . . . PROGANEW
2800     17          COBOLSYMBOL             . . . PROGAOLD
2900     18          ALGOLSYMBOL             . . . PROGBNEW
3000                                         . . . PROGBNEW
3100     19          ALGOLSYMBOL             . . . . PATCH
3200     20          JOBSYMBOL               . . . TASKVALUE
3300
3400
3500
3600                    LISTDIRECTORY INPUT WAS:
3700
3800        "F : DIR = (RWD)CONCEPTS/CLASS ON SYSTEMSED NA PR"
```

BURROUGHS LARGE SYSTEMS 2372 REPORT OF 02/25/86 AT 13:14:08. VERSION 36.130

FAMILY = SYSTEMSED. ON UNIT 51
SERIAL NUMBER = 814114 CREATED 3/03/85.

HIERARCHICAL DIRECTORY STRUCTURE. PAGE 1

FILE NUMBER	CREATION DATE	DATE LAST ACCESSED	FILEKIND	STATUS	CLASS	CONTROLLED SEGMENTS	HIERARCHICAL NAMES
							*BD
							. 0000290
							. . 0000291
1	2/25/86	2/25/86	BACKUPPRINTER	CRUNCH	A	2970	. . . 000LINE
							*DTS
2	3/20/84	12/18/85	DCALGOLSYMBOL	CRUNCH	A	1358	. MF
							. JOHN
3	5/03/85	5/03/85	DATA		P	1008	. . TEST
4	5/01/85	5/01/85	DATA		P	1000	. JUNK
							. MT983
5	5/06/85	5/06/85	DATA		P	2040	. . PRINTERFILE
6	9/04/85	9/04/85	DATA	CRUNCH	P	30	. STAT1
7	9/04/85	9/04/85	DATA	CRUNCH	P	30	. STAT3
8	5/03/85	5/03/85	DATA	CRUNCH	P	30	. JOHNIN
9	4/19/85	4/19/85	DATA	CRUNCH	P	30	. JUNKIN
10	9/04/85	9/04/85	DATA	CRUNCH	P	30	. VBSTAT
11	9/04/85	9/04/85	DATA	CRUNCH	P	30	. VBSTATUS3
						5586	#
							*WFL
							. RUN
							. . COMS
12	8/14/85	12/16/85	JOBSYMBOL	CRUNCH	A	14	. . . TEST
2716	12/14/82	12/14/82	COBOLCODE	CRUNCH	P	27	. . . . OBJECT
2717	8/13/82	12/14/82	COBOLSYMBOL	CRUNCH	P	28	. . . SCREENPRINT
						97	. . #
							. . PATCHES
2718	2/07/86	2/07/86	DCALGOLSYMBOL	CRUNCH	P	56	. . . B20
2719	7/25/85	8/09/85	DCALGOLCODE	CRUNCH	A	27	. . UTILITY
							. . UTILITY
2720	7/25/85	2/09/86	DCALGOLCODE	CRUNCH	A	27	. . . COMS
							. . . COMS
2721	7/25/85	7/25/85	DCALGOLSYMBOL	CRUNCH	A	42	. . . . SYMBOL
2722	12/22/82	12/22/82	DCALGOLCODE	CRUNCH	A	27	. . . DEBUG
2723	7/25/85	7/25/85	DCALGOLSYMBOL	CRUNCH	A	42	. . . SYMBOL
						138	. . #
2724	7/25/85	8/09/85	ALGOLSYMBOL	CRUNCH	P	14	. . PRINTDOC
						11308	. #
							. USERDATAFILE
							. . ALL
2725	11/04/85	2/14/86	DATA	CRUNCH	A	322	. . . LIST
2726	2/24/86	2/24/86	SEQDATA		P	980	. . CURRENT
2727	7/02/85	2/24/86	SEQDATA	CRUNCH	P	14	. DEFAULTDEFINES
						222042	#

TOTAL SEGMENTS SHOWN = 1560659

LISTDIRECTORY INPUT WAS:

"F:PR;"

BURROUGHS LARGE SYSTEMS 2372 REPORT OF 02/25/86 AT 13:14:08. VERSION 36.130

FAMILY = SYSTEMSED. ON UNIT 51
SERIAL NUMBER = 814114 CREATED 3/03/85.

AREA LAYOUT OF FILES. PAGE 1

FILE NUMBER	AREA		AREACLASS	FAMILY INDEX	SEGMENT ADDRESS	SIZE IN SEGMENTS	SEGMENTS ON FAMILY	FILENAME
1	ROW	0		1	1121809	700		*BD/0000290/0000291/000LINE.
	ROW	1		1	1125421	700		
	ROW	2		1	1126625	700		
	ROW	3		1	1127829	700		
	ROW	4		1	1129537	170		
							2970 FAMILY 1	
2	ROW	0		1	1175553	504		*DTS/MF.
	ROW	1		1	1176057	504		
	ROW	2		1	1180999	350		
							1358 FAMILY 1	
3	ROW	0		1	1304412	1008		*DTS/JOHN/TEST.
							1008 FAMILY 1	
4	ROW	0		1	1258285	1000		*DTS/JUNK.
							1000 FAMILY 1	
5	ROW	0		1	1296771	1020		*DTS/MT983/PRINTERFILE.
	ROW	1		1	1297791	1020		
							2040 FAMILY 1	
6	ROW	0		1	460338	30		*DTS/STAT1.
							30 FAMILY 1	
7	ROW	0		1	541905	30		*DTS/STAT3.
							30 FAMILY 1	
8	ROW	0		2	310268	30		*DTS/JOHNIN.
							30 FAMILY 2	
9	ROW	0		1	543517	30		*DTS/JUNKIN.
							30 FAMILY 1	
10	ROW	0		2	302393	30		*DTS/VBSTAT.
							30 FAMILY 2	
11	ROW	0		1	184214	30		*DTS/VBSTATUS3.
							30 FAMILY 1	
12	ROW	0		1	203407	14		*WFL/RUN/COMS/TEST.
							14 FAMILY 1	
13	ROW	0		1	666399	42		*WFL/COMS/GEMCOS/FORMATLIBRARY.
							42 FAMILY 1	
14	ROW	1		1	388427	182		*WFL/LINC.

BURROUGHS LARGE SYSTEMS 2372 REPORT OF 02/25/86 AT 13:14:08. VERSION 36.130

FAMILY = SYSTEMSED. ON UNIT 51
SERIAL NUMBER = 814114 CREATED 3/03/85.

THE DISK CHECKERBOARD. PAGE 135

SPACE OCCUPIED BY LOCKED FILES

FAMILY INDEX	BASE ADDRESS	END ADDRESS	LENGTH		AREA	FILENAME
2	360945	361126	182	ROW	1	(RWD)LINC/WFL/LINC11.
2	361127	361326	200	ROW	4	(ADDSDB)DATADICTIONARY/AUDIT2.
2	361327	361356	30	ROW	106	(SYSTEMSED)IDC/CONFIG/021086.
2	361368	361661	294	ROW	0	(DLG)GEMCOSCLASS/OMNI/COMS/INSTALLATION.
2	361662	362021	360	ROW	16	*BOOK/DATADICTIONARY/ENGLISH.
2	362022	362051	30	ROW	15	(SYSTEMSED)IDC/CONFIG/022186.
2	362052	362081	30	ROW	17	(SYSTEMSED)IDC/CONFIG/022186.
2	362082	362111	30	ROW	19	(SYSTEMSED)IDC/CONFIG/022186.
2	362112	362141	30	ROW	21	(SYSTEMSED)IDC/CONFIG/022186.
2	362142	362145	4	ROW	0	(RWD)WFLCLASS/WFL/EC/TIMECARD.
2	362150	362205	56	ROW	9	(SYSTEMSED)SMFII/EXAMPLES/QUERY/WORKLOAD/-GLOSSARY/SYMBOL.
2	362206	362209	4	ROW	0	(DMJ)EASDB/LINCCONTRO.
2	362210	362219	10	ROW	0	(JCA)EDITOR/OPTIONS.
2	362220	362223	4	ROW	0	(DMJ)EASDBII/LINCCONTRO.
2	362224	362226	3	ROW	0	(RWD)WFLCLASS/WFL/EC/MSTRBKUP.

TOTAL IN-USE SEGMENTS ON FAMILY 2 = 333208

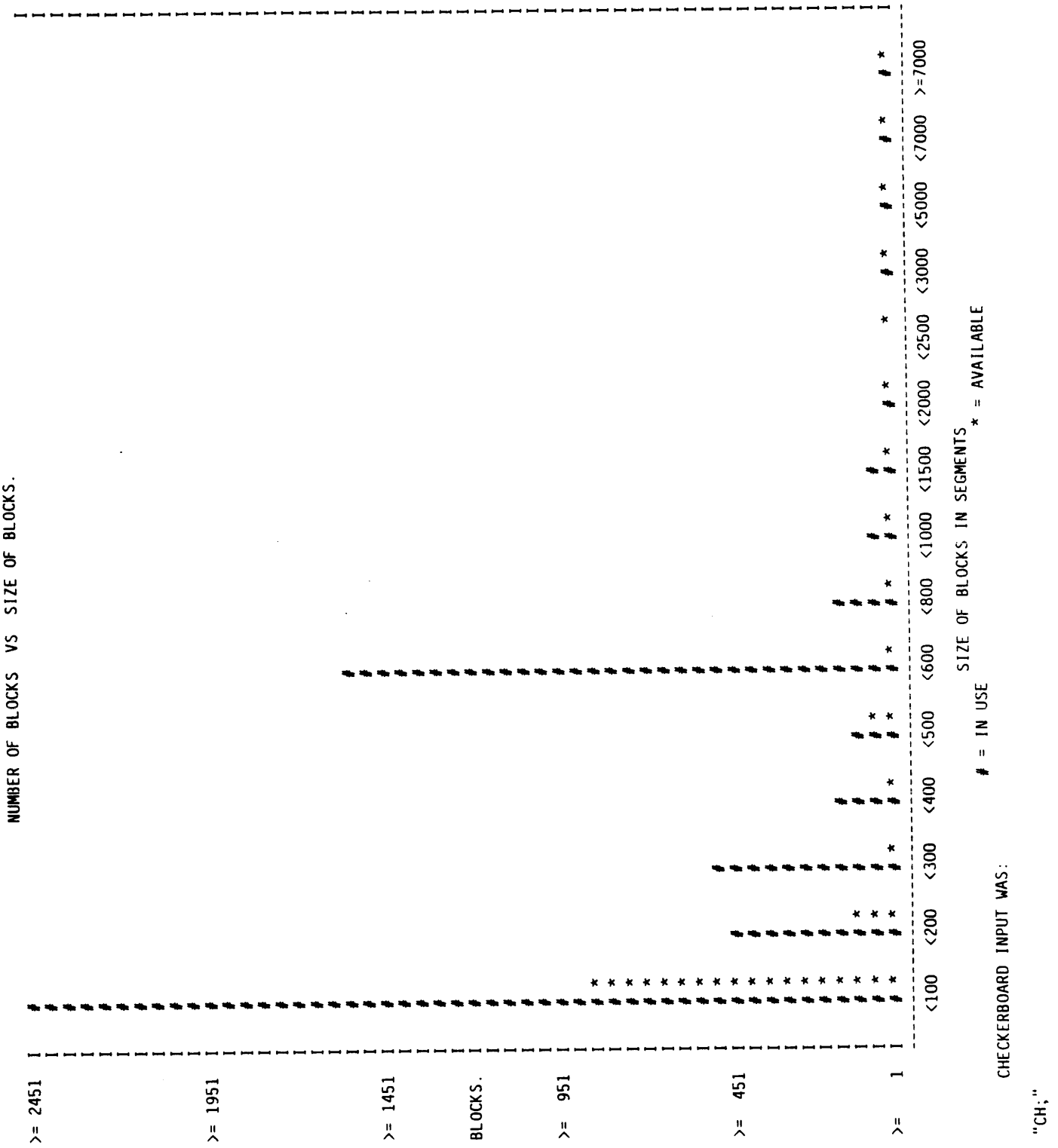
SPACE BETWEEN ROWS

FAMILY INDEX	BASE ADDRESS	END ADDRESS	LENGTH
2	361357	361367	11
2	362146	362149	4
2	362227	THRU END OF FAMILY.	

TOTAL CHECKERBOARDED SEGMENTS ON FAMILY 2 = 29019

SECTION 5: UTILITIES

Unit 3: System/Filedata



SECTION 5: UTILITIES

Unit 3: System/Filedata

PRACTICE EXERCISE

Write a WFL job that will print the attributes for the files under the usercode CEC, a hierarchical list of those file names, and a structure map of the family EDUCATION using SYSTEM/FILEDATA.

UTILITIES

Unit 4

System/Filecopy

OBJECTIVE

Construct a WFL job that will copy files to a backup location using SYSTEM/FILECOPY.

PURPOSE

This utility allows the user to generate a WFL job that will copy files using Library Maintenance routines based on a time/date function.

RESOURCES

a. Student Materials	Section 5 Unit 4
b. A Series System Software Utilities	Section 5

KEY WORDS

SECTION 5: UTILITIES

Unit 4: System/Filecopy

LEARNING SEQUENCE

This utility is intended to simplify the library maintenance requirements of the user site. This is accomplished by allowing the user to specify the files and/or kinds of files that are to be copied. The user may request that the utility create a card deck of the copy requests. SYSTEM/FILECOPY will create a copy deck tailored to the specifications supplied by the user.

SYSTEM/FILECOPY will use the supplied specifications relating to the location and/or types of files to be copied and create a copy deck. At this point, it will either start the copy at that time or save the copy deck for future use. Once a deck is created, it may be used repeatedly without executing SYSTEM/FILECOPY again. If you wish the copy deck to be dynamic in nature, then the option is provided to execute SYSTEM/FILECOPY from a WFL job and execute the deck generated automatically.

```
BEGIN JOB FILECOPY/JOB ;
  DESTNAME = RJESYSCEC ;
  USER = 0718WFL/CLASS ;
  FAMILY DISK = EDUCATION OTHERWISE DISK ;

  RUN *SYSTEM/FILECOPY ;
    DATA CARD
      UPDATED AFTER 01/28/85              %  TIME FRAME
      FILES ( (0718WFL)=                  %  WHAT FILES
              FROM EDUCATION(PACK) )
              TO BKUP(TAPE)               %  WHERE TO
      DECKLABEL LW/FILECOPY/DECK          %  NAME OF JOB CREATED
      LOCKDECK                            %   PERMANENT
      NOZIP                               %  DO NOT EXECUTE
?                                         %  END OF DATA DECK

  RUN *SYSTEM/DUMPALL("L LW/FILECOPY/DECK;L LW/FILECOPY/JOB");
                                          %  LIST JOB CREATED

END JOB
```

The preceding example would build a WFL source deck called LW/FILECOPY/DECK. This is to a permanent file and will not automatically execute. The files to be copied are all the files that have been updated after 1/28/85 on the family EDUCATION under the usercode 0718WFL. They will be copied to a tape called BKUP.

SECTION 5: UTILITIES

Unit 4: System/Filecopy

LEARNING SEQUENCE

In the next example, the job is executed at the bottom of this page showing job responses. On the next page is the output generated by FILECOPY utility with the WFL source that was generated on the page after that.

```
00000100BEGIN JOB UTIL3 ;
00000200 %
00000210 OPTION = (BDBASE) ;
00000220 BDNAME = RWD ;
00000300 RUN *SYSTEM/FILECOPY ;
00000400  EBCDIC CARD
00000500   CREATED  AFTER 05/23/84
00000550     FILES ((RWD)CONCEPTS/CLASS/=  FROM SYSTEMSED(PACK))
00000600     TO DISK(PACK)
00000700    DECKLABEL (RWD)COPYJOB
00000800   LOCKDECK
00000900   NOZIP
00001000?
00001100 RUN *SYSTEM/DUMPALL
00001200   ("L (RWD)CONCEPTS/CLASS/UTIL3; L (RWD)COPYJOB") ;
00001220
00001240 PB "RWD ON EDUCATION" SAVE ;
00001260
00001300 END JOB ;
```

If you start this job from a terminal, you see the following results.

```
ST
  RUNNING 1789
JOB 1790 IN QUEUE 50


1790 BOJ  UTIL3
1790/1791 BOT  *SYSTEM/FILECOPY
1791 (RWD)COPYJOB REMOVED ON SYSTEMSED PK50.
1792/1792 EOJ  JOB COPY
1790/1791 EOT  *SYSTEM/FILECOPY
1790/1793 BOT  *SYSTEM/DUMPALL ON SYSTEMSED
1790/1793 EOT  *SYSTEM/DUMPALL ON SYSTEMSED
1790/1794 BOT  *SYSTEM/BACKUP
```

SECTION 5: UTILITIES

Unit 4: System/Filecopy

LEARNING SEQUENCE

The following data is generated from FILECOPY itself as the copy deck is being generated.

MIX NUMBER 1790/1791

```
100
200       BURROUGHS LARGE SYSTEMS 2372 FILECOPY DUMP
            OF 2/05/85 VERSION 3.5.200
300
400
500             CREATED  AFTER 05/23/84
600               FILES ((RWD)CONCEPTS/CLASS/=  FROM SYSTEMSED(PACK))
700               TO DISK(PACK)
800              DECKLABEL (RWD)COPYJOB
900             LOCKDECK
1000             NOZIP
1100
1200
1300
1400 SUMMARY OF TASK 1, -CREATED-, WITH REQUESTED AND DEFAULT
      OPTIONS.
1500
1600 FILES WHICH PASSED THE TEST CREATED AFTER  5/23/84 WERE
      UTILIZED.
1700 UTILIZED FILES WILL BE COPIED TO DISK
1800 AND THE 'WFL' FILE SAVED AS (RWD)COPYJOB ON DISK. NOT ZIPPED.
1900 OUTPUT FILE(S) CALLED DISK.
2000
2100 14 FILES WERE REQUESTED FOR COPYING.
2200
```

SECTION 5: UTILITIES

Unit 4: System/Filecopy

LEARNING SEQUENCE

If only the second listing from DUMPALL is extracted, you can see that a WFL job has been created to do the copying necessary to complete the task. Make note of the last entry generated. DUMMY is the filler used when there are no more files that match the parameters given.

MIX NUMBER 1790/1793

```
3600 FILE ATTRIBUTES FOR: FIN  TITLE=(RWD)COPYJOB ON SYSTEMSED
     HOSTNAME=TRAINING1  KIND=PACK  INTMODE=EBCDIC
     EXTMODE=EBCDIC  FILETYPE=0 MINRECSIZE=0  MAXRECSIZE=15
     BLOCKSIZE=450  FRAMESIZE=48  FILEUSE=IO  BUFFERS=2
     TRANSLATE=FULLTRANS  PROTECTION=SAVE  POPULATION=1 AREAS=1
     AREALENGTH=6750  FLEXIBLE  LASTRECORD=22  FILEKIND=DATA
     ROWSINUSE=1  CRUNCHED  USERINFO=000000000000
     CREATIONDATE=02/05/85(85036)  LASTACCESSDATE=02/05/85(85036)
     CYCLE=1  VERSION=0  SAVEFACTOR=10  SECURITYTYPE=PRIVATE
     SECURITYUSE=IO  PACKNAME=SYSTEMSED
4100
      1E! ?BEGIN JOB COPY; CLASS= 50; FAMILY DISK = DISK ONLY;
      2E! ?COPY          % FROM SYSTEM 2372:  GENERATED ON  2/05/85
      3E! %   DUMPING CRITERIA USED WAS CREATED AFTER  5/23/84
      4E!       %SOURCE IS (RWD)CONCEPTS/CLASS/= ON SYSTEMSED
      5E! (RWD)CONCEPTS/CLASS/3/DATA,                %DATA  7/31/84
      6E! (RWD)CONCEPTS/CLASS/JOB/1,                %JOBSYMBOL  7/31/84
      7E! (RWD)CONCEPTS/CLASS/WFL,                  %JOBSYMBOL  7/26/84
      8E! (RWD)CONCEPTS/CLASS/WFL2,                 %JOBSYMBOL  7/26/84
      9E! (RWD)CONCEPTS/CLASS/NEWBB,                %ALGOLSYMBOL  7/26/84
     10E! (RWD)CONCEPTS/CLASS/PROGA,                %COBOLCODE  7/26/84
     11E! (RWD)CONCEPTS/CLASS/UTIL1,                %JOBSYMBOL  2/05/85
     12E! (RWD)CONCEPTS/CLASS/UTIL3,                %JOBSYMBOL  2/05/85
     13E! (RWD)CONCEPTS/CLASS/WFL2B,                %JOBSYMBOL  10/12/84
     14E! (RWD)CONCEPTS/CLASS/WFL3A,                %JOBSYMBOL  7/27/84
     15E! (RWD)CONCEPTS/CLASS/WFL3B,                %JOBSYMBOL  10/12/84
     16E! (RWD)CONCEPTS/CLASS/WFL4A,                %JOBSYMBOL  7/27/84
     17E! (RWD)CONCEPTS/CLASS/FILECOPY,             %DATA  7/31/84
     18E! (RWD)CONCEPTS/CLASS/TASKVALUE,            %JOBSYMBOL  10/08/84
     19E! %      14 FILES WILL BE DUMPED.
     20E! DUMMY/LASTFILEFROM/SYSTEMSED  %NON-EXISTANT FILE WITHOUT
            TRAILING COMMA
     21E! FROM SYSTEMSED (KIND=PACK)
     22E! TO DISK ;
     23E! ?END JOB
```

SECTION 5: UTILITIES

Unit 4: System/Filecopy

LEARNING SEQUENCE

The example of SYSTEM/FILECOPY presented below will find those files whose last access date was before 01/01/86. The files considered are those in the directory (WFL)/ALGOL on the family SYSTEMSED. Since this is a copy utility, the destination is disk. The name of the job source to be created is (WFL)ALGOLRM and it is made permanent through the use of the LOCKDECK clause. The NOZIP statement prevents the job from being executed.

```
BEGIN JOB FILECOPYREMOVE;
 RUN *SYSTEM/FILECOPY;
  DATA CARD
   EXPIRED AFTER 01/01/86
   FILES ((WFL)ALGOL/= FROM SYSTEMSED(PACK))
   TO  DISK(PACK)
   DECKLABEL (WFL)ALGOLRM
   LOCKDECK
   NOZIP
?
END JOB
```

The beauty of this utility is that you may execute the job created (on the next page) to copy the older files that have not been used for a while and then modify the job itself to remove the files. All that is necessary is to change the copy statement to a remove statement using the files already provided.

SECTION 5: UTILITIES

Unit 4: System/Filecopy

LEARNING SEQUENCE

```
?BEGIN JOB COPY; CLASS= 50; FAMILY DISK = DISK ONLY;                          %00000000
?COPY                    % FROM SYSTEM 2372:  GENERATED ON  2/25/86           %00000100
%   DUMPING CRITERIA USED WAS EXPIRED AS OF  1/01/86                          %00000200
      %SOURCE IS (WFL)ALGOL/= ON SYSTEMSED                                    %00000300
(WFL)ALGOL/LAB/EX/16,                    %ALGOLSYMBOL  4/24/85                %00000400
(WFL)ALGOL/LAB/16/B,                     %ALGOLSYMBOL  4/24/85                %00000500
(WFL)ALGOL/LAB/WFL/D/ALGOL/DATABWPB2,    %DATA  2/16/82                       %00000600
(WFL)ALGOL/LAB/WFL/D/ALGOL/DATAPROB1,    %DATA  5/07/84                       %00000700
(WFL)ALGOL/LAB/WFL/D/ALGOL/DATAPROB2,    %DATA  2/23/82                       %00000800
(WFL)ALGOL/LAB/WFL/D/ALGOL/UNSORTEDDATA, %DATA  2/25/82                       %00000900
(WFL)ALGOL/LAB/WFL/D/CLASS/COBOL,        %DATA 10/16/84                       %00001000
(WFL)ALGOL/LAB/WFL/D/CLASS/SCORES,       %DATA 10/23/84                       %00001100
(WFL)ALGOL/LAB/WFL/D/CLASS/SCORES/GOOD,  %DATA 10/15/84                       %00001200
(WFL)ALGOL/LAB/WFL/D/CLASS/SCORES/SAVE,  %DATA 10/16/84                       %00001300
(WFL)ALGOL/LAB/WFL/D/SCORES,             %DATA  5/07/84                       %00001400
(WFL)ALGOL/LAB/WFL/D/SPORTS,             %DATA  2/24/82                       %00001500
(WFL)ALGOL/LAB/WFL/D/SPORTSNAMES,        %DATA  5/07/84                       %00001600
(WFL)ALGOL/LAB/WFL/D/STUDENTINFO,        %DATA  2/24/82                       %00001700
(WFL)ALGOL/LAB/WFL/1,                    %ALGOLSYMBOL  2/08/85                %00001800
(WFL)ALGOL/LAB/WFL/2,                    %ALGOLSYMBOL  2/08/85                %00001900
(WFL)ALGOL/LAB/WFL/3,                    %ALGOLSYMBOL  2/08/85                %00002000
(WFL)ALGOL/LAB/WFL/4,                    %ALGOLSYMBOL 11/13/85                %00002100
(WFL)ALGOL/LAB/WFL/13,                   %ALGOLSYMBOL  2/08/85                %00002200
(WFL)ALGOL/LAB/WFL/14,                   %ALGOLSYMBOL  7/25/85                %00002300
(WFL)ALGOL/LAB/WFL/4A,                   %ALGOLSYMBOL  2/08/85                %00002400
(WFL)ALGOL/LAB/WFL/5A,                   %ALGOLSYMBOL  2/08/85                %00002500
(WFL)ALGOL/LAB/WFL/7A,                   %ALGOLSYMBOL  2/08/85                %00002600
(WFL)ALGOL/LAB/WFL/7B,                   %ALGOLSYMBOL  2/08/85                %00002700
(WFL)ALGOL/LAB/WFL/10B,                  %ALGOLSYMBOL  2/08/85                %00002800
(WFL)ALGOL/LAB/WFL/10C,                  %ALGOLSYMBOL  2/08/85                %00002900
(WFL)ALGOL/LAB/WFL/11B,                  %ALGOLSYMBOL  5/02/85                %00003000
(WFL)ALGOL/LAB/WFL/12C,                  %ALGOLSYMBOL  2/08/85                %00003100
(WFL)ALGOL/LAB/WFL/16B,                  %ALGOLSYMBOL  7/25/85                %00003200
(WFL)ALGOL/LAB/WFL/LAB2,                 %ALGOLSYMBOL 10/30/84                %00003300
(WFL)ALGOL/LAB/WFL/LAB2/OUTPUT,          %DATA 10/18/84                       %00003400
(WFL)ALGOL/LAB/WFL/LAB3,                 %ALGOLSYMBOL 10/29/84                %00003500
(WFL)ALGOL/LAB/WFL/SAMPLE/BILL/TEST,     %ALGOLSYMBOL  3/11/85                %00003600
(WFL)ALGOL/LAB/WFL/SAMPLE/BIND,          %ALGOLSYMBOL  3/11/85                %00003700
(WFL)ALGOL/LAB/WFL/SAMPLE/INIT/DATABASE, %ALGOLSYMBOL  3/11/85                %00003800
(WFL)ALGOL/LAB/WFL/SAMPLE/INIT/CONTROLFILE,%ALGOLSYMBOL  3/11/85              %00003900
(WFL)ALGOL/LAB/WFL/SAMPLE/SEEK/DATA,     %DATA 10/16/84                       %00004000
(WFL)ALGOL/LAB/WFL/SAMPLE/TEST/IO,       %ALGOLSYMBOL  3/11/85                %00004100
(WFL)ALGOL/LAB/WFL/SAMPLE/TEST/EOF,      %ALGOLSYMBOL  3/11/85                %00004200
(WFL)ALGOL/LAB/WFL/SAMPLE/TEST/SCAN,     %ALGOLSYMBOL  3/11/85                %00004300
(WFL)ALGOL/LAB/WFL/SAMPLE/TEST/SEEK,     %ALGOLSYMBOL  3/11/85                %00004400
(WFL)ALGOL/LAB/WFL/SAMPLE/TEST/ERNIE,    %ALGOLSYMBOL  3/11/85                %00004500
(WFL)ALGOL/LAB/WFL/SAMPLE/TEST/PWORD,    %ALGOLSYMBOL  3/11/85                %00004600
(WFL)ALGOL/LAB/WFL/SAMPLE/TEST/DELIMS,   %ALGOLSYMBOL  3/11/85                %00004700
(WFL)ALGOL/LAB/WFL/SAMPLE/TEST/FORMAT,   %ALGOLSYMBOL  3/11/85                %00004800
(WFL)ALGOL/LAB/WFL/SAMPLE/TEST/POINTER,  %ALGOLSYMBOL  3/11/85                %00004900
(WFL)ALGOL/LAB/WFL/SAMPLE/TEST/ANYFAULT, %ALGOLSYMBOL  3/11/85                %00005000
(WFL)ALGOL/LAB/WFL/SAMPLE/TEST/CONTROLFILE,%ALGOLSYMBOL  3/11/85              %00005100
(WFL)ALGOL/LAB/WFL/SAMPLE/TEST/LASTRECTEST,%ALGOLSYMBOL  3/11/85              %00005200
(WFL)ALGOL/LAB/WFL/SAMPLE/TEST/CURSORPOSITION,                                %00005300
%ALGOLSYMBOL  3/11/85                                                         %00005400
(WFL)ALGOL/LAB/WFL/SAMPLE/TEST/FINDLASTRECORD,                                %00005500
%ALGOLSYMBOL  3/11/85                                                         %00005600
%     51 FILES WILL BE DUMPED.                                                %00008500
DUMMY/LASTFILEFROM/SYSTEMSED  %NON-EXISTANT FILE WITHOUT TRAILING COMMA       %00008600
FROM SYSTEMSED (KIND=PACK)                                                    %00008700
TO DISK ;                                                                     %00008800
?END JOB                                                                      %00008900
```

SECTION 5: UTILITIES

Unit 4: System/Filecopy

PRACTICE EXERCISE

Write a WFL job to copy all the files updated since yesterday (5:00 p.m.) under the usercode used in the lab. The backup should be placed on a tape called CLASSBKUP and the job should start at 10:00 p.m. on the day being backed up.

UTILITIES

Unit 5

System/Loganalyzer

OBJECTIVE

Construct a WFL job that will execute SYSTEM/LOGANALYZER to generate reports from system logging.

PURPOSE

This utility allows the user to find out information about the system by accessing the system log files.

RESOURCES

a. Student Materials	Section	5
	Unit	5
b. A Series System Software Site Management	Section	7

KEY WORDS

SECTION 5: UTILITIES

Unit 5: System/Loganalyzer

LEARNING SEQUENCE

The utility SYSTEM/LOGANALYZER program is more of a report generator than it is an analyzer. Its purpose is to allow a user to produce hardcopy reports using the entries in the SYSTEM/SUMLOG file. Since all system activities are entered into that file, it is possible for an installation to produce reports based on any of those entries. It is, therefore, possible to obtain reports such as all the activities for a particular job, for a given time frame, or for a group of operator inputs.

```
LOG 0800 OPERATOR DSED
  RUNNING 3198
                                35.200.141 LOGANALYZER
02/07/85,  13:28:03, MCP: SYSTEM/MCP35200.  35.200 ,  SYSTEM SERIAL:  2372
REQUEST IS:  0800 OPERATOR DSED
DSED STOP.
* UNRECOGNIZED REQUEST.
```

```
LOG 0800 OPERATOR DS
  RUNNING 3199
                                35.200.141 LOGANALYZER
02/07/85,  13:28:23, MCP: SYSTEM/MCP35200.  35.200 ,  SYSTEM SERIAL:  2372
REQUEST IS:  0800 OPERATOR DS
LOG FILE IS:*SYSTEM/SUMLOG ON SYSTEMSED.
(LOG FILE IS 4488 RECORDS LONG)
LOG CONTAINS RECORDS FROM FEB 7, 1985 07:00:13 TO FEB 7, 1985 13:28:25
(CURRENT
     LOG)

   10:28:33        ->3009  OPERATOR ENTERED: DS.
   10:31:18        ->3018  OPERATOR ENTERED: DS.
   10:27:25        ->2998  OPERATOR ENTERED: DS NONE.
   10:24:51        ->3003  OPERATOR ENTERED: DS.
   10:30:28        ->3006  OPERATOR ENTERED: DS.
   11:10:27        ->3138  OPERATOR ENTERED: DS.
   11:30:02        ->3158  OPERATOR ENTERED: DS.
   11:41:24        ->3169  OPERATOR ENTERED: DS.
   12:16:12        ->3184  OPERATOR ENTERED: DS.
   12:29:07        ->3185  OPERATOR ENTERED: DS.
   12:29:47        ->3188  OPERATOR ENTERED: DS.
   12:41:04        ->3189  OPERATOR ENTERED: DS.

NORMAL TERMINATION FOR LOGANALYZER PROGRAM.
```

SECTION 5: UTILITIES

Unit 5: System/Loganalyzer

LEARNING SEQUENCE

LOG 1200 BOJ

```
  RUNNING 3202
                                35.200.141 LOGANALYZER
02/07/85,  13:32:42, MCP: SYSTEM/MCP35200.  35.200 ,  SYSTEM SERIAL:   2372
REQUEST IS:  1200 BOJ
LOG FILE IS:*SYSTEM/SUMLOG ON SYSTEMSED.
(LOG FILE IS 4541 RECORDS LONG)
LOG CONTAINS RECORDS FROM FEB 7, 1985 07:00:13 TO FEB 7, 1985 13:32:44
(CURRENT

     LOG)


FEB 7, 1985

   12:33:35    BOJ  3191  "?BEGIN JOB;".
                          JOB ENTERED SYSTEM: FEB 7, 1985 12:33:35 FROM
WFL 35
    .200
                          QUEUE: 50
                          ORIGINATING LSN: 1  MCS: 0
                          PRIORITY: 50

FEB 7, 1985
   12:34:07    BOJ  3193  "AUTOPRINT/LP00004/ 3191".
                          QUEUE: 0
                          ORIGINATING UNIT: 0
                          PRIORITY: 80

FEB 7, 1985
   12:41:21    BOJ  3194  "AUTOPRINT/LP00004/ 3183".
                          QUEUE: 0
                          ORIGINATING UNIT: 0
                          PRIORITY: 80

NORMAL TERMINATION FOR LOGANALYZER PROGRAM.
```

SECTION 5: UTILITIES

Unit 5: System/Loganalyzer

LEARNING SEQUENCE

LOG 1205 MSG

```
  RUNNING 3206
                                35.200.141 LOGANALYZER
02/07/85,  13:38:25, MCP: SYSTEM/MCP35200.  35.200 ,  SYSTEM SERIAL:  2372
REQUEST IS:  1205 MSG
LOG FILE IS:*SYSTEM/SUMLOG ON SYSTEMSED.
(LOG FILE IS 4616 RECORDS LONG)
LOG CONTAINS RECORDS FROM FEB 7, 1985 07:00:13 TO FEB 7, 1985 13:38:26
(CURRENT
     LOG)

   12:10:49          2896  (TOP50TM11)CANDE/TEXT670 CHANGED TO
(TOP50TM11)BEST1
    /LOG ON TOP50 PK44.
   12:36:54          2896  (TOP50TM2)CORK/DESIGN/DOC REMOVED ON TOP50
PK44.
   12:36:54          2896  (TOP50TM2)CANDE/TEXT660 CHANGED TO
(TOP50TM2)CORK
    /DESIGN/DOC ON TOP50 PK44.
   12:23:13          3186  STACK EXTENDED FROM 2111 TO 2233 WORDS.
   12:23:30          3186  UNDEFINED VARIABLE FILE @ 067:0027:2 @
(21613020)*
   12:23:36          3187  STACK EXTENDED FROM 2111 TO 2233 WORDS.
   12:31:57          3187  UNDEFINED VARIABLE FILE @ 063:0022:3 @
(21613020)*
   12:12:33          3184  STACK EXTENDED FROM 2111 TO 2233 WORDS.
   12:16:12        ->3184  OPERATOR ENTERED: DS.
   12:16:12          3184  OPERATOR DSED @ (38924670)*
   12:16:33          3185  STACK EXTENDED FROM 2111 TO 2233 WORDS.
   12:29:07        ->3185  OPERATOR ENTERED: DS.
   12:29:07          3185  OPERATOR DSED @ (38924670)*
   12:29:14          3188  STACK EXTENDED FROM 2111 TO 2233 WORDS.
   12:29:47        ->3188. OPERATOR ENTERED: DS.
   12:29:47          3188  OPERATOR DSED @ (38924670)*
   12:29:58          3189  STACK EXTENDED FROM 2111 TO 2233 WORDS.
   12:41:04        ->3189  OPERATOR ENTERED: DS.
   12:41:05          3189  OPERATOR DSED @ (34176380)*
   12:34:05          3192  BD/0003145/0003190/000OUTFILE REMOVED ON
EDUCATION

NORMAL TERMINATION FOR LOGANALYZER PROGRAM.
```

SECTION 5: UTILITIES

Unit 5: System/Loganalyzer

LEARNING SEQUENCE

LOG 1000 COMMENT

```
  RUNNING 3205
                                35.200.141 LOGANALYZER
02/07/85,  13:38:05, MCP: SYSTEM/MCP35200.  35.200 ,  SYSTEM SERIAL:  2372
REQUEST IS:  1000 COMMENT
LOG FILE IS:*SYSTEM/SUMLOG ON SYSTEMSED.
(LOG FILE IS 4599 RECORDS LONG)
LOG CONTAINS RECORDS FROM FEB 7, 1985 07:00:13 TO FEB 7, 1985 13:38:06
(CURRENT
     LOG)

NO ENTRIES FOUND FOR THIS REQUEST.
```

LOG 1205 JOB 3191

```
  RUNNING 3203
                                35.200.141 LOGANALYZER
02/07/85,  13:34:01, MCP: SYSTEM/MCP35200.  35.200 ,  SYSTEM SERIAL:  2372
REQUEST IS:  1205 JOB 3191
LOG FILE IS:*SYSTEM/SUMLOG ON SYSTEMSED.
(LOG FILE IS 4558 RECORDS LONG)
LOG CONTAINS RECORDS FROM FEB 7, 1985 07:00:13 TO FEB 7, 1985 13:34:03
(CURRENT
     LOG)


FEB 7, 1985
   12:33:35     BOJ  3191  "?BEGIN JOB;".
                           JOB ENTERED SYSTEM: FEB 7, 1985 12:33:35 FROM
WFL 35
    .200
                           QUEUE: 50
                           ORIGINATING LSN: 1  MCS: 0
                           PRIORITY: 50

   12:33:36     BOT  3192  SYSTEM/BACKUP.
                           CODE COMPILED: MAR 14, 1984 10:01:55 BY
DCALGOL 34

--- AND MUCH MORE ---
```

SECTION 5: UTILITIES

Unit 5: System/Loganalyzer

PRACTICE EXERCISE

Write a WFL job to check on all the information about a job with the mix number of 2983.

UTILITIES

Unit 6

Additional Utilities

OBJECTIVE

Identify the uses of other utilities available on large systems.

PURPOSE

There are several other utilities that may be of use on the system. This unit will identify some of these.

RESOURCES

a. Student Materials — Section 5, Unit 6

b. A Series System Software Utilities

c. A Series System Software Site Management

KEY WORDS

SECTION 5: UTILITIES

Unit 6: Additional Utilities

LEARNING SEQUENCE

There are many more utilities that are available to the large system user than just the five presented in the previous units. The following is a brief description of some of the additional utilities available that may be of use.

- **SYSTEM/COMPARE**

  This utility will compare any number of pairs of files. The comparison is on a bit by bit basis within each record. When the comparison between two records fails, the records are listed on the line printer. Some of the options allow for the comparison of sequence numbers and maximum error limits.

- **SYSTEM/DCSTATUS**

  The SYSTEM/DCSTATUS utility program is intended to give an installation the ability to produce run-time reports and/or "snapshots" of the data communications sub-system operations. It is possible to obtain reports for stations, lines, quad-line adapters/clusters, NSP/DCP or all of the previous elements.

  With this utility, the user may define where the reports are to go. The default report medium is the line printer. However, the reports may be directed to a terminal. In that case, the output format will be modified for the 72 character line width.

  Reports may be generated not only on the currently active network definition files but on other network definition files. However, when non-active files are being accessed, there are some restrictions on what may be reported.

  It should be understood that in order to interpret the reports generated by SYSTEM/DCSTATUS an understanding of NDLLII/NDL and NSP/DCP is required.

SECTION 5: UTILITIES

Unit 6: Additional Utilities

LEARNING SEQUENCE

- **SYSTEM/DUMPANALYZER**

  This utility is used to analyze full system dumps. When an error is detected in the system that was not caused by a user program (task), the MCP will cause all of memory to be written out to a tape. This tape will be labeled "MEMORY/DUMP". This tape then becomes the input file to SYSTEM/DUMPANALYZER. Dumpanalyzer has several ways it may be executed.

  - The simplest method is the ODT command (**DA**). With this command SYSTEM/DUMPANALYZER is entered into the mix and will look for the input file labeled MEMORY/DUMP. When the analysis is finished, it is written out to the line printer.

  - SYSTEM/DUMPANALYZER may also be run using the ODT as one of the input files. "OPTIONS" command tells the utility which of dumpanalyzer options are to be used. The output will go to the line printer.

  - The third method available is to run the utility from a remote terminal. When running from a remote terminal, the output may be directed to the line printer and/or the terminal.

- **SYSTEM/MAKEUSER**

  This utility is used by the installation in order to establish controls over access to the system by local and/or remote users. This is accomplished by the execution of SYSTEM/MAKEUSER, which will create or update a data base known as SYSTEM/USERDATAFILE. This data base will contain a list of valid usercodes and other kinds of data relative to the users of the system. The security system, of which SYSTEM/MAKEUSER is a major facet, is designed to meet the following goals:

  - Provide a single file with information about all users that is easily maintained by the installation management.

  - Supply file and system security that can be applied to both batch and data comm users.

  - Allow installations to easily tailor the file to their own needs.

  - Make the file easily accessible through both random and serial accessing methods.

LEARNING SEQUENCE

- **SYSTEM/MAKEUSER (continued)**

  All access to the USERDATAFILE is controlled and provided by procedures of the MCP. These procedures are responsible for the enforcement of the system security through the use of the information in the USERDATAFILE.

- **SYSTEM/GUARDFILE**

  This utility is a further extention of security for use by installations requiring a higher level of restricted access to certain files by various users. SYSTEM/GUARDFILE is used to create a file that describes the relationship between various files and various users and/or programs.

  The GUARDFILE is examined by the MCP whenever a file or data base, secured by the GUARDFILE, is opened. Thus only those that have been entered into the file will be granted access; all others will be denied access to the file.

- **SYSTEM/RLTABLEGEN**

  This utility program gives the installation management the ability to define to the system any non-standard tape labels. This is done so they can be handled automatically by the MCP's label recognition procedure (READALABEL).

  The utility is given a description of the desired non-standard labels. With these descriptions, the program will build tables for each label. Then the program will bind these tables to the MCP. With this new MCP, tapes with these installation defined labels may be used in the same manner as tapes with standard label formats.

- **SYSTEM/PATCH**

  This utility program is used to merge one or more patch decks into a single patch deck which can be used for the primary input file of a compiler. The program merges all input records by sequence numbers in the order received by the utility. If duplicate sequence numbers are received, the last one received will be the active record.

SECTION 5: UTILITIES

Unit 6: Additional Utilities

PRACTICE EXERCISE

Match the system utility name with its correct description given below.

_____1.	SYSTEM/DUMPANALYZER	A.	Allows for "snapshots" of the operation of a data communication subsystem.
_____2.	SYSTEM/PATCH	B.	Used to describe the types of non-standard tape labels being used on the systems.
_____3.	SYSTEM/RLTABLEGEN	C.	Used to do a bit by bit comparison of two files.
_____4.	SYSTEM/DCSTATUS	D.	Used to set up and control the security file to be applied to a system.
_____5.	SYSTEM/COMPARE	E.	Used for controlled access to files, by defining who may have access to what and how.
_____6.	SYSTEM/MAKEUSER	F.	Used after a memory dump has occurred in order to help find the cause.
_____7.	SYSTEM/GUARDFILE	G.	Used to merge multiple patch files into a single patch file.

SECTION 5: UTILITIES

LAB :

Through the use of ODT commands, CANDE commands, or WFL jobs complete the following task using SYSTEM/BACKUP, SYSTEM/DUMPALL, SYSTEM/FILEDATA, SYSTEM/FILECOPY, or SYSTEM/LOGANALYZER.

1. **SYSTEM/BACKUP:** Through the ODT, CANDE terminal, or a WFL job:

   A. Run OP/PRINT/PRACTICE, but store the printer backup files under a BDNAME in your directory (INITIALS/_____).

   B. Use SYSTEM/BACKUP to print your file.

2. **SYSTEM/DUMPALL:** Run SYSTEM/DUMPALL through WFL, the ODT, or a CANDE terminal(INTERACTIVELY) to accomplish each of the following.

   A. Starting with the third record in the file, list **OP/PROG/3/DATA** in hex and alpha.

   B. Copy **OP/PROG/3/DATA** to **<YOUR INITIALS>/PROG/3/DATA**, with a new record size of 15 words and a new block size of 30 words.

   C. Verify (via DUMPALL, FILEDATA, OR CANDE LFILES) that the above file was copied with the proper attributes.

   D. Concatenate the files **OP/JOB/1/A** and **OP/JOB/1/B** in a new file called **<YOUR INITIALS>/JOB/1**.

3. **SYSTEM/FILEDATA:** Through the ODT or a WFL job, print a listing of the creation data and last access date of a all files under your directory in the lab usercode.

SECTION 5: UTILITIES

UTILITIES LAB (continued)

4. **SYSTEM/FILECOPY:** Write a WFL job to run FILECOPY as follows:

   A. Copy to a tape all files created under the lab usercode on the lab family since Monday of last week.

   B. Store the WFL job created as **LAB/<YOUR INITIALS>/FILECOPY.**

   C. Do not allow FILECOPY to initiate the job, so that you can edit the job before you start it yourself.

   D. Use CANDE or DUMPALL to study the job created. You may start the job if you have a scratch tape available for output.

5. **LOGANALYZER:** Through the ODT or a CANDE terminal, run SYSTEM/LOGANALYZER against the current SYSTEM/SUMLOG to print (or display) all the DSs and PRIORITY changes entered by the operator.

APPENDIX A

ALTERNATIVE LABS

APPENDIX A

WORKFLOW LAB PROJECT 1

PURPOSE:

To write and debug a WFL job using the run statement, embedded data, parameters, and the DUMPALL utility.

GENERAL NOTES:

(1) Cause all system messages to be displayed on your terminal.
(2) USERCODE = _____ PASSWORD = _____
(3) FAMILY DISK = _____ OTHERWISE _____ .
(4) Name your CANDE work file as follows:

TESTWFL/<YOURNAME>/LAB1

(5) Set the DESTNAME attribute to the appropriate setting.

DIRECTIONS:

(1) Using CANDE, create a file of type JOB. If you forget to use type JOB, CANDE will default to type SEQDATA, and you will be unable to start the WFL.

(2) The purpose of this lab is to run SYSTEM/DUMPALL passing it the parameter "CARD". DUMPALL will expect input batch commands to be entered via the FILE CARD.

(3) The DUMPALL commands that this lab requires are COPY, LIST, LAN, and CAT.

A. COPY FROM TESTWFL/<YOURNAME>/LAB1 RECORDS 1,4,7-10, AND 11, creating the file TESTWFL/<YOURNAME>/LAB1/DATA1.

B. Also, COPY from TESTWFL/<YOURNAME>/LAB1 RECORDS 1-5,7 and 10, creating the file TESTWFL/<YOURNAME>/LAB1/DATA2.

C. List file TESTWFL/<YOURNAME>/LAB1/DATA1 in EBCDIC.

D. List file TESTWFL/<YOURNAME>/LAB1/DATA2 in ALPHA, and NUMERIC format.

E. Concatenate TESTWFL/<YOURNAME>/LAB1/DATA1 and TESTWFL/<YOURNAME>/LAB1/DATA2 creating a new file with a different BLOCKSIZE titled TESTWFL/<YOURNAME>/LAB1/DATA3.

F. List the resulting file in STEP E in EBCDIC format.

APPENDIX A

WORKFLOW LAB PROJECT 2

PURPOSE:

To write and debug a WFL job using compilers and task variables.

GENERAL NOTES:

(1) Cause all system messages to be displayed on your terminal.
(2) USERCODE = _____ PASSWORD = _____
(3) FAMILY DISK = _____ OTHERWISE _____ .
(4) Name your CANDE work file as follows:

TESTWFL/<YOURNAME>/LAB2

(5) Set the DESTNAME attribute to the appropriate setting.

DIRECTIONS:

(1) Using CANDE, get the source files CLASS/PROGANEW and CLASS/PROGBNEW that exist on DISK as PROGANEW/<YOURNAME> and PROGBNEW/<YOURNAME>, respectively.

(2) Patch and compile the COBOL source PROGANEW/<YOURNAME> as follows:

A. Name the object file "OBJECT/PROGA/<YOURNAME>" and store it in your library for possible subsequent executions.

B. As embedded data, include the patch below in your WFL job:

```
COL:  1      8  12  16        27
      000070 77  Z   COMP-4    VALUE 4.
```

C. Create a new COBOL source disk file called NEWAA/<YOURNAME>.

D. Obtain a hardcopy printout and verify that the patch has been inserted correctly.

DIRECTIONS: (continued)

(3) Patch and compile the ALGOL source file "PROGBNEW/<YOURNAME>" as follows:

A. Name the object file "OBJECT/PROGB/<YOURNAME>" and store it in your library for possible subsequent executions.

B. Create a disk file thru CANDE called PATCHFILE/<YOURNAME> with the following record stored in: **(REMEMBER TO MAKE THIS PATCHFILE TYPE ALGOL)**

```
COL:          1
      00001400  FOR I:= 1 STEP 1 UNTIL 35 DO
```

C. Create a new source disk file called NEWBB/<YOURNAME>.

D. Obtain a hardcopy printout and verify that the patch has been inserted correctly.

If PROGANEW fails to compile, run the ALGOL object file "OBJECT/CLASS/PRINTB2".

If PROGBNEW fails to compile, display an appropriate message.

APPENDIX A

WORKFLOW LAB PROJECT 3

PURPOSE:

To use library maintenance routines and gain more experience in compiling programs and using IF statements.

GENERAL NOTES:

(1) Cause all system messages to be displayed on your terminal.
(2) USERCODE = _____ PASSWORD = _____
(3) FAMILY DISK = _____ OTHERWISE _____ .
(4) Name your CANDE work file as follows:

TESTWFL/<YOURNAME>/LAB3

(5) Set the DESTNAME attribute to the appropriate setting.

DIRECTIONS:

(1) Simultaneously copy the ALGOL files CLASS/SQRT AND CLASS/PRINTA as CLASS/SQRT/<YOURNAME> and CLASS/PRINTA/<YOURNAME>, respectively. These files reside on the pack "EDUCATION" and will be copied to the pack "EDUCATION".

(2) Compile CLASS/SQRT/<YOURNAME> to the library. Call your object file "OBJECT/CLASS/SQRT/<YOURNAME>. Create a new source file called NEW/CLASS/SQRT/<YOURNAME>. Include the following patch for your source file in your WFL job. This patch will be your primary input file.

```
3000 $SET LIST MERGE NEW
3100 INTEGER I;                                          /
        / 00000250
```

Your secondary source file is the file you copied in STEP 1.

Obtain a hardcopy printout also.

*** NOTE—YOU DON'T WANT TO COMPILE CLASS/SQRT/<YOURNAME> UNTIL THE FILE HAS BEEN COPIED INTO YOUR LIBRARY (STEP 1).**

APPENDIX A

DIRECTIONS: (continued)

(3) If CLASS/SQRT/<YOURNAME> didn't compile successfully, then abort the job. If CLASS/SQRT/<YOURNAME> did compile successfully, do the following:

(A) Display an appropriate message. Run the file "OBJECT/CLASS/SQRT/<YOURNAME>". Use the STATION attribute to get the results of the execution of this program on your terminal. During execution of this program, the terminal operator will supply integer values to the program. When you have entered all of your values in **(ENTER A VALUE AND XMIT)**, key in **?END**.

(B) Compile the ALGOL program CLASS/PRINTA/<YOURNAME> to your library. **(THE PRIMARY INPUT FILE IS YOUR SOURCE FILE CLASS/PRINTA/<YOURNAME>.)** Name your object file "OBJECT/CLASS/PRINTA/<YOURNAME>".

(C) If CLASS/PRINTA/<YOURNAME> compiled successfully, run the file "OBJECT/CLASS/PRINTA/<YOURNAME>". If the object doesn't complete successfully, run OBJECT/CLASS/SUMVALUE.

APPENDIX A

WORKFLOW LAB PROJECT 4

PURPOSE:

To write and debug a WFL job using strings as parameters to the job.

GENERAL NOTES:

(1) Cause all system messages to be displayed on your terminal.
(2) USERCODE = _____ PASSWORD = _____
(3) FAMILY DISK = _____ OTHERWISE _____ .
(4) Name your CANDE work file as follows:

TESTWFL/<YOURNAME>/LAB4

(5) Set the DESTNAME attribute to the appropriate setting.

DIRECTIONS: PART ONE

(1) Write a WFL job that will compile any source program disk file creating an object file with the word "OBJECT" appended to the source file name.

(2) Pass only the source file name as a parameter to the job. (HINT): This compile deck will require the use of a global file, interrogation of the file attribute FILEKIND, and string functions.

(3) To test your WFL, start this job, passing the ALGOL source file name "CLASS/PRINTB" as the above mentioned source file name. You might want to display a message if the source compiles successfully.

DIRECTIONS: PART TWO

Name your CANDE work file as follows:

TESTWFL/<YOURNAME>/LAB4A

(1) Part two is a simple extension to the compile deck created from the directions above. The purpose is to give you practice constructing a subroutine. Convert TESTWFL/<YOURNAME>/LAB4 into a subroutine, which can be invoked from the main body of the job. Pass a second parameter to this subroutine which will be used as the task variable for the compile.

APPENDIX A

WORKFLOW LAB PROJECT 5

PURPOSE:

To write a series of subroutines which will give you practice in running the utilities SYSTEM/FILEDATA, SYSTEM/LOGANALYZER, AND SYSTEM/BACKUP. You will get additional practice in interrogating file and task attributes, and with string concatenation.

GENERAL NOTES:

(1) Cause all system messages to be displayed on your terminal.
(2) USERCODE = _____ PASSWORD = _____
(3) FAMILY DISK = _____ OTHERWISE _____ .
(4) Name your CANDE workfile as follows:

TESTWFL/<YOURNAME>/LAB5

(5) Set the DESTNAME attribute to the appropriate setting.

DIRECTIONS:

(1) Using CANDE get TESTWFL/<YOURNAME>/LAB3 as TESTWFL/<YOURNAME>/LAB5.

(2) Using CANDE, insert the compile subroutine from LAB4A in the declaration section.

(3) Have all compiles in the WFL invoke the compile subroutine.

***** NOTE *****

Instead of compiling CLASS/SQRT/<YOURNAME> use NEW/CLASS/SQRT/<YOURNAME> which will enable you to compile using a single source.

(4) Write a subroutine named FYLEDATA that will receive one string parameter containing the name of a file. Generate a FILEDATA attributes report containing the following information: FILEKIND, BLOCKSIZE, RECORDSIZE, CREATION DATE, AND SECURITY.

***** NOTE *****

This subroutine should be invoked every time you have a successful compile.

APPENDIX A

DIRECTIONS: (continued)

(5) Write a subroutine named LOGIT that will receive one task parameter. Generate a LOGANALYZER task report by extracting the task number from the parameter.

***** NOTE *****

This subroutine should be invoked if a compilation terminates abnormally, or if the program OBJECT/NEW/CLASS/SQRT/<Y.N> terminates abnormally. To test the subroutine, terminate OBJECT/NEW/CLASS/SQRT/<Y.N.> by entering **?<MIX NUMBER>DS**, instead of ?END

This subroutine should display one message containing the following information:

- CURRENT SYSTEM TIME
- TASK NUMBER
- TASK NAME
- HISTORY TYPE
- HISTORYCAUSE

(6) Generate all printer backup under the directory beginning with your name. Print out any listings at the end of the job using SYSTEM BACKUP and double spacing the output.

APPENDIX A

WORKFLOW LAB PROJECT 6

PURPOSE:

To write a WFL job to check taskvalues that were set in a program and to pass values to another program. This lab will give you some indication as to WFL's capability to communicate with a program.

GENERAL NOTES:

(1) Cause all system messages to be displayed on your terminal.
(2) USERCODE = _____ PASSWORD = _____
(3) FAMILY DISK = _____ OTHERWISE _____ .
(4) Name your CANDE workfile as follows:

TESTWFL/<YOURNAME>/LAB6

(5) Set the DESTNAME attribute to the appropriate setting.

DIRECTIONS:

Write a WFL job to run the object file "OBJECT/CLASS/SET/TASKVALUE". In this program, taskvalues are set. The taskvalues set are 1 or 2, depending on program logic. You are to check these values in your WFL job and perform the following steps:

(1) If the taskvalue = 1, run the object file "OBJECT/CLASS/PRINTB", passing it a value of 3. Use the display statement to indicate that you successfully completed the execution of this program. Use the SOURCESTATION attribute to get your results of the execution of "OBJECT/CLASS/PRINTB" to your station.

(2) If the taskvalue = 2, then display a message indicating that the taskvalue = 2 at this point.

***** NOTE *****

If you look at the program that set the taskvalue, you will notice that the action taken for a taskvalue of 2 will not occur. However, depending on program logic, we could have fired off another program, or done something else at that point.

A SERIES AND B 5/6/7000 WORKFLOW AND UTILITIES

BIBLIOGRAPHY

REFERENCE MATERIALS

A Series Work Flow Student Guide	
A Series Work Flow Language Ref. Manual	1169802
A Series System Software Utilities	1170024
A Series I/O Subsystem Ref. Manual	1169984
A Series Print System	1169919
A Series Printing Utilities	1169950
A Series System Software Site Management	1170008
A Series ODT Reference Card	5013907
A Series ODT Reference Manual	1169612
A Series CANDE Reference Card	5014533
A Series CANDE Reference Manual	1169869
The Complete CANDE Primer	Gregory
The B 6700 WFL Primer	Gregory